IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 7, 2020, accepted November 5, 2020, date of publication November 9, 2020, date of current version November 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3036938

A Deep Reinforcement Learning Approach for the Patrolling Problem of Water Resources Through Autonomous Surface Vehicles: The Ypacarai Lake Case

SAMUEL YANES LUIS, DANIEL GUTIÉRREZ REINA, AND SERGIO L. TORAL MARÍN
Department of Electronic Engineering, Technical School of Engineering, University of Seville, 41004 Sevilla, Spain

Corresponding author: Samuel Yanes Luis (syanes@us.es)

This work was supported in part by the Universidad de Sevilla under the contract "Contratos de acceso al Sistema Español de Ciencia, Tecnología e Innovación para el desarrollo del programa propio de I+D+i de la Universidad de Sevilla," by the Spanish "Ministerio de Ciencia, innovación y Universidades, Programa Estatal de I+D+i Orientada a los Retos de la Sociedad" through the Project "Despliegue Adaptativo de Vehículos no Tripulados para Gestión Ambiental en Escenarios Dinámicos" under Grant RTI2018-098964-B-I00, and in part by the regional government Junta de Andalucía through the Projects "Despliegue Inteligente de una red de Vehículos Acuáticos no Tripulados para la monitorización de Recursos Hídricos" under Grant US-1257508 and "Despliegue y Control de una Red Inteligente de Vehículos Autónomos Acuáticos para la Monitorización de Recursos Hídricos Andaluces" under Grant PY18-RE0009.

ABSTRACT Autonomous Surfaces Vehicles (ASV) are incredibly useful for the continuous monitoring and exploring task of water resources due to their autonomy, mobility, and relative low cost. In the path planning context, the patrolling problem is usually addressed with heuristics approaches, such as Genetic Algorithms (GA) or Reinforcement Learning (RL) because of the complexity and high dimensionality of the problem. In this paper, the patrolling problem of Ypacarai Lake (Asunción, Paraguay) has been formulated as a Markov Decision Process (MDP) for two possible cases: the homogeneous and the non-homogeneous scenarios. A tailored reward function has been designed for the non-homogeneous case. Two Deep Reinforcement Learning algorithms such as Deep Q-Learning (DQL) and Double Deep Q-Learning (DDQL) have been evaluated to solve the patrolling problem. Furthermore, due to the high number of parameters and hyperparameters involved in the algorithms, a thorough search has been conducted to find the best values for training the neural networks and the proposed reward function. According to the results, a suitable configuration of the parameters allows better results for coverage, obtaining more than the 93% of the lake surface on average. In addition, the proposed approach achieves higher sample redundancy of important zones than other common-used algorithms for non-homogeneous coverage path planning such as Policy Gradient, lawnmower algorithm or random exploration, achieving an 64% improvement of the mean time between visits.

INDEX TERMS Deep reinforcement learning, monitoring, path planning, autonomous surface vehicle, patrolling, complete coverage.

I. INTRODUCTION

Ypacaraí Lake is the largest body of water in Paraguay with more than 60 km^2 of navigable surface (Fig. 1). It is located between the cities of San Bernardino (eastwards), Areguá (westwards) and Ypacaraí (southwards) as the main source of water supplying in the area. Within the years, its importance has become bigger with tourism since it has been used as a recreational lake for people to swim and for water sports. Its importance is also related to the natural life developed in the wetlands of the basin surrounding of the lake. Nonetheless, in the past 40 years, the continuous expansion of the agriculture in the surroundings of the lake, the lack of sewerage systems in the near cities and the disposals of wastes from industries located at the shore, among other factors, have caused in the lake an abnormal eutrophication process [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Moinul Hossain.

FIGURE 1. Ypacarai Lake.

FIGURE 2. Prototype of ASV designed for the exploration of Ypacaraí Lake in [1].

This artificial eutrophication (non natural enrichment of the waters) causes the flourish of cyanobacteria and since 2012, it has been in an aggressive form with several blooms of green-blue algae. These colonies of algae come with fetid smells and the generation of toxins like microsystis, harmful for wildlife and humans, even deadly in some occasions [2]. It should be noted that the nature of cyanobacteria outbreaks is dynamic and quite chaotic, so the prediction of where and when they will arise is not a trivial process [3]. Furthermore, the bloom behavior is not static and changes its size with time, which makes the task of measuring the quality of the water an arduous process. This contamination problem is an interdisciplinary challenge in which diverse engineering and scientific profiles must work together. The solutions should come from the help of multidisciplinary techniques that not only focus on reducing waste disposals and infrastructure investments, but also on monitoring and studying the current state of the lake to find an efficient way to reverse the situation.

It is vital to monitor efficiently the state of the lake in order to have an updated image of the biological status of algae blooms. This contamination map allows to analyze the performance of the environmental measures taken by the authorities and researchers. However, the manual monitoring task takes a lot of effort and human resources since it requires constant travels from the shore to the main blooms with motor boats and manual sampling of waters. In [1] the use of Autonomous Surface Vehicles (ASV) equipped with water quality sensors is proposed to substitute the manual sampling. These vehicles, which are similar to ships but smaller and without a crew to operate them, have been widely used for environmental monitoring tasks [4]. They present many advantages compared with ordinary ships, such as being cheaper in fuel expenses, more environment-friendly due to the use of electric motors that can be connected to a battery and a solar panel power system, and of course, the ability to operate without an on-board crew.

Monitoring applications of water resources based on ASV have been proposed in the last few years as a promising approach [4]. Every ASV has the ability to be operated via remote control (RC) and/or autonomously by travelling along a path of selected waypoints. These points are normally generated following a coverage criterion and allow the ASV to take samples with its sensors along the established path. For instance in [1], an ASV has been designed and constructed to explore and monitor Ypacaraí Lake (Fig. 2). It is equipped with several water quality sensors such as PH-meters, dissolved oxygen, and Oxide-Potential Reduction (OPR) sensors, among others. These sensors provide the ASV the ability to estimate the contamination levels (in terms of cyanides concentration and oxygen concentration levels) along the path planned.

The monitoring task ultimately falls on a path planner, which takes as input the map of the lake and generates a set of waypoints to visit in a way that maximizes the coverage of the surface. Depending on the mission objectives, a large amount of possible paths could be taken into consideration. In this paper, the path planning for monitoring tasks of Ypacarai Lake is addressed under two perspectives: i) the patrolling problem with homogeneous importance, which assumes that every zone of the lake is equally important; thus, the ASV should visit all waypoints with the same frequency, and ii) the patrolling problem with a non-homogeneous importance, considering different relative importance for every waypoint of the lake. The former case is a myopic approach, where non previous information on the lake is considered; consequently, the ASV should explore the whole lake to build a contamination map of it. In the latter case, prior information is considered to create an underlying contamination map of the lake; therefore, the path planning algorithm leverages this model to plan the routes of the ASV and provides higher frequency to these areas that are more probable to be contaminated.

It must be defined the difference between a *Complete Coverage* task and the *Patrolling Problem*. In the former, an episodic task is achieved, where once every zone is covered, then the mission ends. In the latter, the aim is to revisit periodically every zone (attending to its importance and given a certain amount of timesteps) without neglecting any location for a long time. Consequently, in the Ypacaraí monitoring task, the Patrolling Problem is targeted considering the relative importance between the zones (given by the contamination map already mentioned). In this way, visiting

every zone will be as necessary as revisiting important zones after a while.

The proposed strategy addresses the problem of finding an efficient patrolling of the lake by training the agent with a Deep Reinforcement Learning (DRL) algorithm called Deep Q-Learning [5] and its evolution Double Deep Q-Learning [6], in a discretized representative map of the Ypacaraí Lake. This is accomplished by modelling the problem as a Markov Decision Process (MDP) with a tailored reward function for every movement of the agent. Thus, the learning process is focused on obtaining a better policy in each episodic iteration.

The main contributions of this work are (a) the application of deep reinforcement learning for monitoring the quality of water resources like the Ypacarai Lake with an ASV (a DDQL approach with convolutional neural network representation and a tailored and non-homogeneous reward function are proposed for an efficient patrolling) and (b) a comparison between our approach, other reinforcement learning approaches (Policy Iteration, DQL) and other common path planning algorithms (lawn-mower and randomized).

This paper continues as follows: Section II includes the main related works of the present work. Section III describes the problem addressed in this paper. The proposed approach based on Deep Reinforcement Q-learning is detailed in Section IV. In Section V, results of the hyperparametrization, performance of the algorithm, and a comparison with other strategies are presented. Finally, in Section VI, the conclusion and future works are included.

II. RELATED WORK

Since there is not a unique and systematic approach to achieve the patrolling task, it is mandatory to consider optimization algorithms such as Evolutionary Algorithms [1], [2]) or Reinforcement Learning Algorithms [7] to deal with the complexity of the problem. These approaches, based on artificial intelligence foundations, return a near-optimal performance and have been demonstrated as useful techniques for a wide variety of problems [8]. There are many proposed solutions for the patrolling problem as this is a recurrent task in robotics [9], [10]. Depending on the problem, there is a difference between the exploration with only one agent and the multi-agent paradigm. In the latter, it is mandatory to ensure the trajectories avoid the collision between the robots. In [11] and in [12] a multi-agent approach has obtained good results.

In [13] the problem is modelled as a Travelling Salesman Problem (TSP), which can be solved with optimization algorithms like GA. In [14] a cyclic lawn mower solution is proposed with an evolutionary approach to optimise the coverage path planning with a multi-agent perspective. In [1], the particular case of Ypacaraí Lake path planning is addressed by formulating the coverage problem as a Hamiltonian circuit, with remarkable results; nonetheless, it is an off-line algorithm and, consequently, has no reactivity to model uncertainties. In [15], authors compared a Hamiltonian and Eulerian formulation for the resolution of the Ypacaraí coverage problem, resulting in efficient paths given by a genetic algorithm optimization. Moreover, in [16], the patrolling problem of Ypacarai lake is addressed by a GA approach and a TSP model with a variable number of waypoints. The main idea is to select the waypoints according to the areas with higher interest.

Other heuristic approaches lay on Reinforcement Learning techniques [7], [11], [17]–[19]. Using Reinforcement Learning algorithms for path planning in robotics has become a trendy approach since [5] proposed Deep Q-Learning as a solution for a human-like intelligence in Atari 2600 games. Deep Reinforcement Learning algorithms such as Deep Q-Learning and its subsequent algorithms like Double Deep Q-Learning [6], Dueling Deep Q-Learning [20], etc, have been used by many authors to achieve good performance in the *complete coverage*, patrolling problem [7], [18], [21] and classic path planning and trajectory generation [19]. The main goal of the complete coverage problem is to obtain the minimal number of waypoints which guides the agent around the scenario by covering every single zone. This is the problem addressed in [22] and [18] by using DQL algorithms. Furthermore, some authors are interested in the solution of non-homogeneous patrolling. This problem is approached by [7] and [11] and result in reactive ways to deal with zones of different importance and an agent with sensors.

Every DRL approach has an environment, a reward function, a state definition and at least one agent with its action space [23]. The environment will define the problem and the contour conditions. Many authors defined the environment in their coverage problems as a two dimension grid map with a representative cell resolution for every scenario. This assumption is adequate for many cases such as in [22] and [19], where the agent moves within a plane. Each cell is usually identified with a label with one of the following categories: visited, unvisited or obstacle [22]. In addition, an interest value with a range of possible values can be given to each cell [11]. Regarding the scenario, it can be stochastic [21], with random events, or be completely deterministic [19].

There are also multiple ways to represent the state of the scenario in a MDP depending on the observability of the environment. The observability of the state lays on how the information of the scenario is given to the agent, i.e. how the scenario internal state affects the observable information; therefore, there are two possible MPD types: Full Observable MDP (the complete state of the scenario is accessible) or Partially Observable MDP (the state must be estimated or is partially unavailable). There are different approaches to a partially-observable state, like in [7] which assumes uncertainties in the state acquisition (all the information of the scenario is not available at every moment because it comes from a camera) and faces temporal dependencies. Other authors define the state as fully-observable and accessible always. A common representation of the state for RL approaches is a RGB image of the observed scenario. This is a very compact

TABLE 1. Summary of the state of the art.

Ref	Scenario	State	Reward function	Agent	RL Algorithm	Estimation function	Hyperparameters
[22]	- Complete Coverage. - 2D gridmap with 4 cell types: visited, not visited, obstacle and agent position.	- RGB image of the scenario. - Fully Observable.	- Constant reward for every new visited cell and for end of task. - Constant penalization for every action and illegal action (collision).	- Mono-agent (Tetromino-like). - 11 actions: 4 traslational, 7 transformative	-Async. Advantage Actors Critic with buffer replay.	- 3x Conv. Layer (8x8x32) with ELU. - Dense (512) with ELU. - LSTM (256) with sigmoid.	- Learn. rate = 1E-3 - Discount factor = 0.9 - Exp. Buffer Size = 20E3 - Epochs = 10E3
[17]	- Complete Coverage. - 3D Cartesian scenario without obstacles.	- Vector of Cartesian coordinates of every agent. - Fully Observable.	- Constant reward for coverage and for end of task. - Constant penalization for every action and illegal action (collision).	- Multi-agent (3 drones) - 6 actions (up, down, north, south, east, west)	- Novel Advantage Actor Critic.	- Gradient descent without Neural Network.	- Learn. rate (Actor) = 2.5E-3 - Learn. rate (Critic) = 5E-2 - Epochs = 10E3
[7]	- Patrolling variable interest zones. - 2D gridmap with variable interest values for each cell.	- Gray-scale image of the visible scenario from the agent's camera. - Partially Observed.	- Variable reward depending on the interest of each cell. Each cell increases its interest with time.	- Mono-agent (1 drone) - 12 actions: 6 traslational, 4 for camera movement and 2 (zoom in/out)	- Double Deep Q Learning.	- Conv. Layer (8x8x16) with ReLU. - 2x Dense (1024) with ReLU. - Dense (12) with linear activation.	- Num of steps = 20
[18]	- Complete Coverage with energy consumption efficiency. - 2D gridmap with 5 cell types:landing cell, take-off cell, obstacle, visitable and objective cell.	- 5 channel image (one channel for cell type)	- Constant reward for every new objective visited cell. - Constant penalization for every action, illegal action and when there is no battery left.	- Mono-agent (1 drone) - 5 actions: 4 traslational and landing action.	- Double Deep Q Learning.	- 2x Conv. Layer (16x16x5) with ReLU. - 4 x Dense (256) with ReLU.- Dense (5).	- Exp. Buffer Size = 10E3 - Batch size = 128 - Learn. rate = 1E-3 - Discount factor = 0.95 - Epochs = 10E3
[21]	- Complete Coverage with area sweeping. - 2D gridmap with time increasing uncertainty cells and robot position.	- RGB image of the scenario.(Partially Observed due to the uncertainty)	- Reward depending on the stochastic events detected in the scenario.	- Mono-agent. - 5 actions: 4 traslational (north, south, east and west).	- Double Deep R Leaning (novel DQL method).	- Conv. Layer (32x32x5) with ReLU. - Max Pool 2x2 - 2x Conv. Layer (16x4x4) with ReLU. - Dense (500) with ReLU.	- Learn. rate = 1E-4 - Discount factor = 0.95
[11]	- Complete Coverage. - 3D scenario and 2D interest gridmap.	- Gridmap of consumed interest and position of every agent.	- Continuous reward function based on the consumption of the interest of each agent. (agents position is mapped in the interest gridmap)	- Multi-agent. - 8 actions: 4 traslational (north, south, east and west and 4 diagonals movs).	- Q-Learning novel approach.	- Gradient descent without Neural Network.	- Learn. rate = 1 - Discount factor = 0.99
[24]	- Complete Coverage.-2D Occupancy grid (from SLAM algorithm).	- Set of points in map representing map frontiers to discover and agent's position.	- Continous reward function depending on the information recopilated along the path to reduce travelled distance.	- Mono-agent. - Travel to every frontier point compounds possible actions.	-Async. Advantage Actors Critic.	Multiple Sequential Parallel Convolutional Neural Nerworks and a final LSTM layer for time dependencies.	- Learn. rate = 1E-4 - Discount factor = 0.99 - Epochs = 3570

and intuitively representation but requires a method to extract the encoded information in the image like a Convolutional Neural Network (CNN). Another representation of the state could be the mere position of the agent. It is the case of [17], which uses the Cartesian coordinates of every agent as a state observation, avoiding the need of CNNs but at the cost of a more intensive learning process.

Regarding the type of Deep Reinforcement Learning algorithm used by authors, there are two main directions: on the one hand the DQL approach, which is a Value Optimization algorithm i.e. tends to find the optimal behavioral policy by finding the optimal value function $V(s)$ for every state s (like in [7], [18] and [21]). On the other hand, the Policy Optimization family, which tends to optimize directly the optimal behavioral policy noted as $\pi^*(s)$ that returns the best action taken in every step given s (used in [22] and [24]).

Besides both approaches (Value Iteration and Policy Iteration) have been proved equally efficient with some differences, while Policy Iteration methods have, in general, a nice stability for every non-stochastic problem [25], they suffer of a high variance in terms of learning (a higher deviation from the total expected reward) and a worse sample efficiency compared to the Value Iteration methods. For such methods as DQL and DDQL, a good estimation of the $V(s)$ could be achieved faster (for a fixed number of epochs) and tends to avoid the local minima problem [23] (as a consequence of a high sample efficiency). These characteristics might suggest Q-Learning (and others Value Iteration related) as a good algorithm to deploy for a faster (and even online) training.

There are also novel approaches like in [19], where a combination of DQL and a parallel exploration structure improves the performance of the path planning and the learning convergence speed. Other novel approach, propposed in [26], implements an interactive Deep Learning algorithm combining a DQL approach and interactive RL for path following in underwater unmanned vehicles. In Table 1 a summary of the state of the art is presented for convenience of the reader. It contains the main characteristics of the proposed solutions in the current literature.

In this work, a grid map scenario for a RL approach has been designed, as usual in robotic path planning algorithms, with a RGB representation of the state. There is a substantial difference between this work and others like [22] and [17] because the state not only reflects the scenario morphology (obstacles and visitable zones) but also the importance of every cell based on the time of the last visit and the absolute importance of the cell in a graphic way. Also a tailored and non-homogeneous reward function has been designed, similar to [7], whereas other works (like [22] or [19]) use a temporal-non-dependant reward with constant reward for each action. Another difference found in this work is the hyperparametrization that has been conducted. Some papers propose learning parameters based on trial-error or in an intuitive idea of the problem needs ([7], [17] and [11]). In this paper, a detailed study of how parameters (hyperparameters and reward function parameters) affect the final performance has been presented. As the Reinforcement Learning approach with a model-free algorithm does not

need a model of the scenario, this algorithm will serve for arbitrary maps, unlike the previous works based on genetics algorithms [2], [14], [15].

III. STATE OF THE PROBLEM

A. ASSUMPTIONS

For the exploration task an ASV prototype is used (Fig. 2). This prototype has several sensors as said before, and will serve the purpose of navigating the Ypacaraí Lake while sampling the water quality and its contamination level. The ASV has a hardware-software architecture that will provide the ability to follow a sequence of waypoints (previously computed and interpolated by a trajectory generator, such as an spline interpolator). The closed-loop guarantees a robust reference tracking of given waypoints and it is implemented in a PixHawk4 autopilot. Thus, once the low-level architecture is ready, the exploration task is achieved by selecting the waypoints the ASV must visit for the best efficiency. It is necessary to remark the assumptions of the application in order to design a sufficiently accurate environment. The assumptions for the environment are:

- Ypacaraí Lake has a total area of 60 km^2 with an average depth of 1.31 m. Different representations of the space can be used to obtain different levels of accuracy in the generated path like octrees discretization (for non-homogeneous resolution) [27] or a simple grid discretization. For the simplicity and computational efficiency, the map has been discretized in squared cells of 580 m $\times$ 580 m, which is a representative size of the sensor effective area and the variables resolution (temperature, PH, dissolved oxygen, ...). The smaller the area, the more accurate will be the scenario but more difficult to compute each iteration of the algorithm. With this representation, a map of 25×19 cells is obtained (see Fig. 3), with a total amount of 179 visitable cells and 296 illegal cells (not-visitable ones). It is not convenient to expand excessively the resolution because the time of execution for the training could easily explode.
- The inner control loop guarantees the reference tracking due to (i) the robust control designs and (ii) the sufficiently-big size of very cell (580 m $\times$ 580 m), which allows positioning errors due to drifts or disturbances.
- Every movement duration is assumed to be the same and in the simulations each timestep equals a movement.
- The following algorithm focuses on the path planning i.e. the computation of a sequence of waypoints to follow. A local trajectory generator and autopilot (PixHawk4) will ensure the movement [28].
- An illegal action (a waypoint in a non-navigable zone) is rejected in simulation and also in the real application by not performing this particular action (the ASV will remain in the same place).
- Besides in the simulation no obstacle in the interior of the Lake is considered as the obstacle sensors in the ASV (Lidar + Camera) provide the ability to avoid them (with a reactive local path planner) and perform effectively the commanded action by the RL path planner propposed.
- The ASV can move in one of the eight different directions of the adjoining cells (N, E, S, W and NE, SE, NW, SW directions) each step. This is representative of the movement capacities of the ASV.

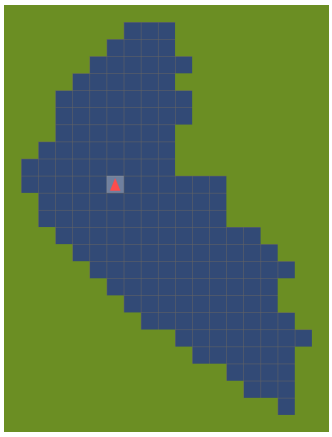

FIGURE 3. Representation of the scenario. Visitable cells are in blue, illegal cells are in green and the agent as the red triangle.

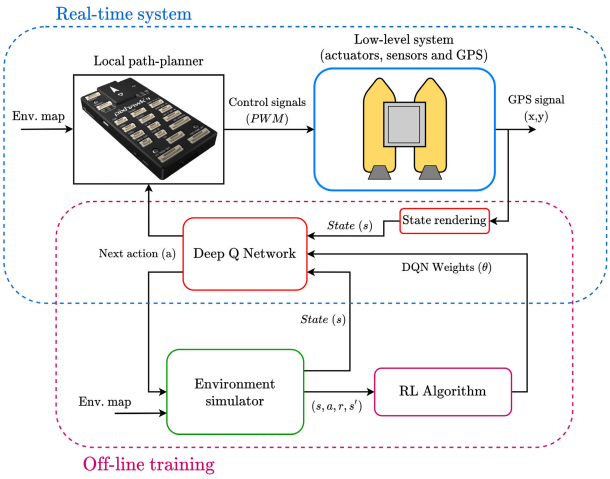

FIGURE 4. Diagram of the ASV control in the real case (up) and the reinforcemet learning loop (down).

A control diagram is presented in Fig. 4. Note that once the off-line training is completed, the real-time system uses the trained pathplanner (Deep Q Network). The state will be rendered thanks to the GPS coordinates and a map of the scenario.

B. THE PATROLLING PROBLEM

The patrolling problem can be described in terms of an undirected graph $G(V, E)$ with each discretized cell V as a node and each edge E will represent the distance between nodes with a non-metric assumption (diagonal movements are treated the same as non-diagonal ones for simplicity) [13]. Given such graph, the aim of the agent is to minimize the average *idleness* of each node, i.e. visit regularly every zone in order to reduce it. The idleness is more known as the *mean*

time between visits T and could be weighted depending on the importance of each cell. It is clear that the number of possible solutions for the coverage task is very high and none of them is trivial (NP-hard problem). This problem is a candidate to a formulation in terms of a Markov Decision Problem (which is indeed formulated as a graph) and in the next section this formulation is used to establish a mathematical framework. The patrolling problem is said to be solved once the path planner is able to choose the movements that results in a minimum mean time between visits. Once resolved, the reference tracking is a task for the low-level loop which controls the actuators (Fig. 4).

In terms of the real environment, the main challenge is to develop a path planner capable of providing the waypoints (the next waypoint, given a position of the ASV in the discrete map) that will result, after a number of movements (timesteps), in the minimization of the average waiting time of every cell depending on the importance criteria of every zone. Once the training is complete, the policy of the ASV will decide every timestep the next move given its state in the map. Regarding the real implementation, the path planner will decide the next waypoint, once there, the sensor sampling is done and a new waypoint is requested.

At this point two variants of the patrolling problem can be considered attending to the importance of every zone:

- **Homogeneous patrolling:** it assumes that every cell in the map is equally important and the homogeneous coverage of the lake is pursued.
- **Non-homogeneous patrolling:** in this case, there are zones with more importance than others. Therefore, these relevant areas should be revisited with a higher frequency (depending on the importance given to each zone).

In the case of Ypacarai Lake, it is well-known that there are high-concentrated contamination areas (e.g. the blooms) with special interest for researchers [1], [2], [15]. These areas match with recreative port areas or industrial zones near the shore like *Puerto de San Blas*, at the southern part of the lake. Therefore, the non-homogeneous patrolling case is more suitable for the Ypacarai scenario.

In general, from the optimization point of view, given a fixed number of movements, the agent pursues the minimisation of the mean time between visits while trying to cover the maximum number of cells. In our case, the number of movements is considered 300 in both patrolling cases. This number of steps is more than enough to cover the lake given the previously resolution of the cells and generate meaningful paths to follow by an ASV. The larger number of steps, the bigger the paths and the more complexity the solution has.

IV. PROPOSED APPROACH

A. IMPORTANCE MATRICES AND REWARD FUNCTION

The reward function proposed is a function based on interest gradient, similar to [7], which is particularly tailored to pursue an efficient patrolling. Two matrices of 25×19 cells are defined. On the one hand, $\mathcal{R}_{abs}$ will define the absolute importance of the lake zones in every cell within a range of [0, 255]. For example, a value of 0 means that the cell is not important at all and a value of 255 means this specific cell has the maximum importance and must be visited. On the other hand, the temporal visited mask $\mathcal{W}$ is defined. This mask values go from 0 to 1 and will weight every cell with position (i, j) (row and column, respectively) of $\mathcal{R}_{abs}$ depending on for how long the cell has not been visited. $\mathcal{W}$ will be updated as follows:

$$\mathcal{W} = \begin{cases} \mathcal{W}(i,j)^{t+1} \\ = min[\mathcal{W}(i,j)^t + \delta, 1] & \text{if } p_{agent} \neq [i,j] \\ \mathcal{W}(i,j)^{t+1} = 0 & \text{if } p_{agent} = [i,j] \end{cases} \quad (1)$$

The parameter δ is a refresh parameter fixed at 0.055, which indicates how many steps must pass until a cell of $\mathcal{W}$ recovers its full value (20 steps in this case). This value defines the threshold between good and unnecessary redundancy. As a value of 20 steps is on average approximately 10 km, it is considered that after this distance the cell is again a candidate for revisiting. It has been observed that a higher value for δ will push short-term trajectories as the importance regenerates quickly and an excessively low value will cause erratic behavior once most of the map is covered since the interest is slowly regenerated. The relative importance matrix R will be the element-wise product of $\mathcal{W}$ and $\mathcal{R}_{abs}$:

$$R = \mathcal{W} \odot \mathcal{R}_{abs} \quad (2)$$

In this way, $\mathcal{W}$ weights the relative importance of every zone ($\mathcal{R}_{abs}$) depending on whether the cell has been visited recently or not. For example, lets imagine a very important cell $[i, j]$ with an absolute value of 255 in $\mathcal{R}_{abs}(i, j)$. When the agent visits this cell, the $\mathcal{W}(i, j)$ value will drop to 0, which means that in the next step the absolute importance of the cell (in $R(i, j)$) will be 0. Within time, this value will be incremented to the maximum again. In order to avoid revisiting a certain cell many times, the absolute importance of a cell is decremented with every visit. By settling a maximum desirable of 5 visits for a maximum importance cell (255 in R_{abs}), each visit will subtract *a fifth* of the maximum value 255, which is subtracting 51 to $R_{abs}(i, j)$, being (i, j) the position of the agent. The maximum number of visits as 5 is chosen to be a good value for useful revisiting in interesting zones; thus, a higher value will result in excessively redundant paths.

Finally, the reward function will depend on the gradient of the relative interest matrix R from a position of the agent (i, j) in the state s to the next position (i', j') given by the action a:

$$\rho(s, s') = R(i', j') - R(i, j) = \nabla R \quad (3)$$

Because the values of $\rho(s, s')$ are in the range of $[R_{min}, R_{max}] = [0, 255]$ (R image values for the worst and best case respectively), a linear transformation has been applied to parametrize the maximum and minimum reward given in the best and worst ∇R case. Thus, it is translated the R importance domain to the reward domain (the real value used in (8) for an easier reward function's parameter tuning. The

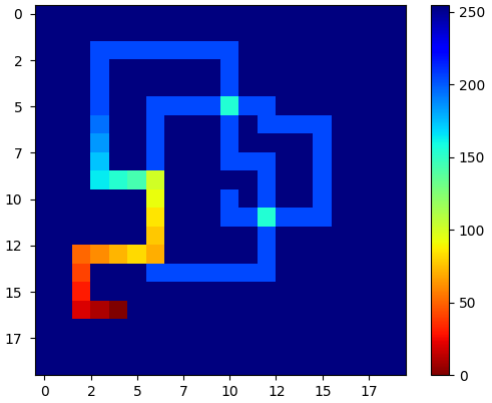

FIGURE 5. Heat map representation of R.

parameters r_{max} and r_{min} define the maximum and minimum reward given to the agent and $C_{illegal}$ is the punishment value for invading a non-visitable cell. The final reward function is described as follows:

$$r = \begin{cases} \dfrac{r_{max} - r_{min}}{R_{max} - R_{min}} [\nabla R - R_{min}] + r_{min} \text{ if } legal \\ -C_{illegal} \text{ if } illegal \end{cases} \quad (4)$$

The reward function will benefit the movement to higher importance zones like unexplored ones or cells not visited for a while. In this way, the cells with a high idle time will be attractive to the agent, fostering the movement to interesting and high-uncertainty zones (a desirable behavior in patrolling). The interest-gradient reward is also useful to balance the interest between a recently-visited but important cell and an unvisited but low-important one: the gradient-based approach pushes the agent to move to an unvisited zone instead of revisiting a just-visited cell. The matrix R can be represented as a heat map (see Fig. 5) where the coldest zones indicate high relative interest and the hottest ones represent low relative interest. Notice that illegals cells values do not matter as they are not visitable.

B. STATE REPRESENTATION

The state representation of the scenario affects the performance of the learning process because it defines what information is available for the agent to learn. In this paper up to three different state representations have been analyzed. The three state representation are:

- **State as the position of the agent:** Only the position of the agent in the map is considered in a two-component vector representation. It is a partial observation of the full state, considering only the position (i, j) of the agent in the map, without any further information of the already visited or illegal zones.
- **RGB image with only the visited cells (RGB 1):** Besides the obstacle (green) cells and the agent position (red), the visitable cells appear in two colors: white for already visited and black for those which remain unvisited (as in Fig. 6a). Illegal (unvisitable) cells are in green.
- **RGB image with full information of the cells (RGB 2):** A RGB image with the obstacles, agent position and gray-scale visitable cells. Each visitable cell will have a gray tone depending on its absolute interest in the previously explained relative interest matrix R. A just recently visited cell will be white and an unvisited cell or a cell with long time without a visit will be black (as in Fig. 6b).

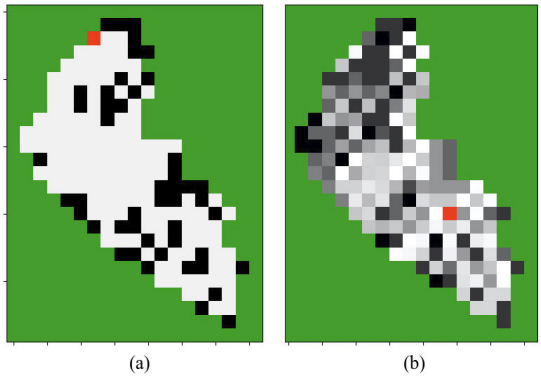

FIGURE 6. Two different state representation. RGB 1 in (a) and RGB 2 in (b).

C. DEEP Q-LEARNING

Deep Q-Learning is a reinforcement learning algorithm that approximates the Q function with a Deep Neural Network (DNN) in the context of a MDP. This function, known also as value-action function, returns the value that a state s has for every possible action a in terms of the future expected reward r for the next N steps.

$$Q(s, a) = \mathbb{E}\left(\sum_{t=0}^{N} r_t \gamma^t\right) \quad (5)$$

where $\gamma \in [0, 1]$ is the discount factor that determines the trade-off between short-term and long-term gains. The aim of Q-Learning algorithm is to find the optimal value for $Q(s, a)$ by learning the relation between an action-state pair (s, a) and the next state s' using the reward r of the transition. The optimal policy, once the Q-Learning algorithm has optimized the Q values is given by:

$$\pi^*(s) = \max_a Q(s, a) \quad (6)$$

In Deep Q-Learning a neural network is used to avoid the high number of possible values for the state. Depending on the state and the number of actions, the Q function dimension could explode (especially when the state has a high dimension like an RGB matrix), thereby a neural network is used and trained to compute the optimal value of $Q(s, a)$ by the *Time Difference* (TD) method [23] as

$$Q(s, a)^{t+1} = Q(s, a)^t + \alpha \times \left(r + \gamma \max_{a'} Q(s', a')^t - Q(s, a)^t\right). \quad (7)$$

In every training step, the $Q(s, a)$ value is compared within its target value $Q(s', a)$, its loss function is evaluated and the Q function is updated by taking a gradient-descent step.

Within this method an experience replay is used to avoid the overfit problem in DNN besides to ensure the effective training of the network by using enough experiences (s, a, r, s') from previous episodes [29]. By random batch of previous store samples, the agent learns from a wider range of transitions.

In [6], it is analyzed how the Deep Q-Learning algorithm sometimes overestimates the Q values. This effect could result in an unstable optimization (as *catastrophic forgetting*) or a bad performance of the agent. As a consequence, Double Deep Q-Learning is proposed as a feasible option to avoid the overoptimistic estimation of Q values. The only difference is that another estimation function is used to obtain the greedy target value of $Q(s, a)$ in (7). The target function Q^* is merely updated every M steps with the weights of the Q function (like in (8)). This is proven to enhance the stability in the learning in many cases. The full learning pseudocode for both DQL/DDQL is provided in Algorithm 1.

$$Q(s, a)^{t+1} = Q(s, a)^t + \alpha \times \left(r + \gamma \max_{a'} Q^*(s', a')^t - Q(s, a)^t \right). \tag{8}$$

D. NEURAL NETWORK

Since the the positional state representation is different in dimension from the RGB representations (the first one is a simple two-component vector and the other two are a RGB image), two architectures of deep neural networks are proposed.

- With the state as the Cartesian position of the agent in the map, a dense (fully-connected) neural network is used. It has an input layer (of two neurons), two hidden layers (of 1024 neurons each) and an output layer (of eight neurons, one for each possible action). The activation function is in every layer a ReLU function (see Fig. 7).
- With state as a RGB image of the scenario a convolutional neural network is used. An initial convolutional layer of eight 5×5 filters will extract the features of the image and a dense neural network of two hidden layers (1024 neurons each) and an output layer (8 neurons) will determine the Q values based on those features (see Fig. 8).

In both cases a Huber Loss function has been used for the gradient descent estimation of $Q(s, a)$ function. This function is used in Reinforcement Learning as it faces less convergence issues than the well-known Root Mean-Squared Error function (RMSE) [30]. Whereas RMSE is quadratic growing in all its domain, Huber Loss function is only quadratic to a certain point and then linear, so a big step will not cause exploding gradients (outliers robustness).

Algorithm 1: DQL/DDQL With Buffer Replay Algorithm

```
Initialize replay memory M to capacity N;
Initialize action-value function Q with random weights θ;
if DDQL is activated then
    Initialize target function Q* with random weights θ⁻;
end
epoch ← 0;
while epoch ≤ epoch_MAX do
    step ← 0;
    Reset the environment and observe the initial state s;
    while step ≤ step_MAX do
        With probability ϵ select a random action a
          otherwise select a = argmax_a Q(s, a; θ)
        Execute action a and observe reward r and next
          state s'
        Store transition (s, a, r, s') in M
        if M ≥ Batch Size then
            Sample random Batch Size (s, a, r, s')
              experiences from M;
            for every (s, a, r, s') sampled do
                if DDQL is activated then
                    q* ≈ r + γ max_a' Q*(s', a'; θ⁻);
                else
                    q* ≈ r + γ max_a' Q(s', a'; θ);
                end
                Perform gradient descent step on
                  Loss(q* − Q(s, a))
            end
        end
        step ← step + 1;
    end
    epoch ← epoch + 1;
    if DDQL is activated then
        Every T epochs, updates Q* weights θ⁻ ← θ ;
    end
end
```

E. EPSILON-GREEDY POLICY

In order to achieve good results in the training, a moving ϵ-greedy policy is implemented for the behavioral policy of the agent. The behavioral policy is the way that an agent chooses the next action in every step of an episode. In an ϵ-greedy policy, the agent has an ϵ (with $\epsilon \in [0, 1]$) probability to choose a random action (exploring the value-action pair domain) and a $1 - \epsilon$ possibility to choose a greedy action i.e. $a_{next} = \max_a Q(s, a)$. At the beginning of the learning, ϵ is higher to ensure a sufficient exploration and as the episodes go by, ϵ is decremented at a constant rate to a minimum (to always ensure exploration). At the end of the learning, the vast majority of the actions are taken greedily.

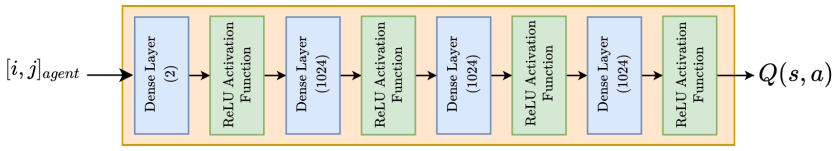

FIGURE 7. Neural Network architecture for positional state representation.

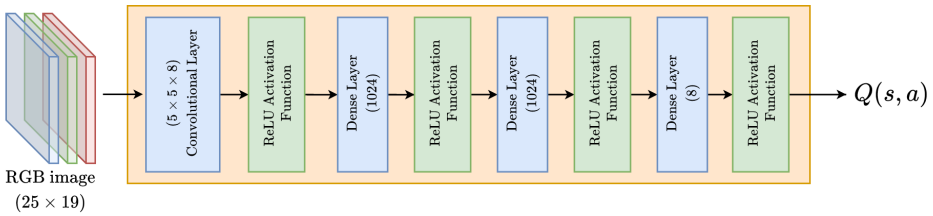

FIGURE 8. Convolutional Neural Network for RGB state representation.

V. PERFORMANCE EVALUATION

This section is divided into multiple parts: first, the environment simulator is presented with the hyperparameters and the performance metrics used to evaluate the proposed approach. Second, the best state representation and reward parameters are chosen by tuning DQL algorithm in the homogeneous patrolling case. In this analysis, with the best model parameters, both DQL and DDQL results for the homogeneous patrolling task are presented. Then, the non-homogeneous patrolling case is addressed for both DQL and DDQL algorithms. In the next part, a hyperparametrization analysis is addressed for DDQL algorithm since it is proven that the DDQL outperforms DQL. Lastly, the best selected results of DDQL are compared with other path planning approaches with a final discussion.

A. ENVIRONMENT SIMULATOR, MODEL PARAMETERS AND PERFORMANCE METRICS

A simulator for the environment has been implemented in Python. The environment is designed to be compatible with *OpenAI Gym*[1] environments, vastly used in Reinforcement Learning researches. The DNNs for Q functions were created using the *PyTorch*[2] library with GPU enhance and *NumPy*[3] for matrix operations. The simulator is able to compute the eight possible actions, and when it is reseted, it initializes all the interest matrices and restarts the agent in a random position of the visitable domain of cells. For every action executed, the scenario returns the actual state s, the reward of the action r and the interest matrices for the analysis. The code can be found in a Github repository.[4] All the simulations were executed in Ubuntu 20.04 with the following hardware specifications: AMD Ryzen 9 3900 (3.8 GHz) CPU, Nvidia RTX 2080S-8GB GPU and 16GB RAM memory.

[1]https://gym.openai.com/
[2]https://pytorch.org/
[3]https://numpy.org/
[4]https://github.com/derpberk/YpacaraiReinforcementLearning

TABLE 2. Model parameters.

Symbol	Definition
$C_{illegal}$	Penalization for an illegal action i.e. attempting to move to an obstacle cell.
r_{max}	Maximum reward possible for visiting a new cell of the maximum interest (255).
r_{min}	Minimum reward possible for visiting a cell which has been visited in within the last action.

TABLE 3. Training hyperparameters.

Symbol	Definition
α	Learning rate. The step size at each iteration while moving towards a minimum of the loss function in the DNN.
Batch Size	Amount of experiences (s, a, s', r) that are sampled from the experience replay to train the Q function.
ϵ-decay	The amount subtracted from ϵ in every epoch.
γ	Discount factor. States the importance of the future reward. A near-one value means the future reward matters more than the short-term reward. It is fixed to 0.99.
Memory Size	Number of experiences recorded in the experience replay and ready to be sampled. It is fixed to 10.000.

All the hyperparameters and scenario parameters are summarized in Tables 2 and 3.

For the performance evaluation, three different metrics are proposed:

- **Coverage**: The coverage κ of the map will be defined as the ratio between the total interest of the visited cells and the total interest of the lake cells. A high value of coverage is desired.

$$\kappa = \frac{\sum_{\forall (i,j) visited} [\mathcal{R}_{abs}(i,j)]}{\sum \mathcal{R}_{abs,visitable}} \tag{9}$$

- **Effective average visiting time**: It is defined by the weighted mean time between visits on average in every cell of the map and divided by κ to be able to compare with different coverage rates. A low value means a good

TABLE 4. Parameters for training in the state evaluation.

Parameter	Value
α	1E-3
Batch Size	250
Mem. Size	10E3
ϵ-decay	7E-4
γ	0.99
Steps	300
Epochs	1500
r_{max}	1
r_{min}	-0.5
$C_{illegal}$	-5

performance of the algorithm because, on average, every cell does not have to wait for long time to be recovered (reducing the uncertainty knowledge of the map).
Let $\tau_k^{(i,j)}$ be the time between a visit in (i, j) cell in the k step:

$$\tau_k^{(i,j)} = t_k^{(i,j)} - t_{k-1}^{(i,j)} \tag{10}$$

Given $\mu_\tau^{(i,j)}$ as the mean of $\tau_k^{(i,j)}$, the average visiting time of the map T_{mean} is defined as:

$$T_{mean} = \frac{1}{m} \sum_{\forall(i,j)} \left[\mu_\tau^{(i,j)} \frac{\mathcal{R}_{abs}(i,j)}{max\mathcal{R}_{abs}} \right] \tag{11}$$

where m is the number of visitable cells. Finally, the effective average T_{mean}^{eff} is:

$$T_{mean}^{eff} = \frac{T_{mean}}{\kappa} \tag{12}$$

- **Effective homogeneity of coverage**: Defines the average deviation of T_{mean}^{eff} in the map. A low value means the coverage is homogeneous and, on average, the time between visits of every cell remains close. It is defined as:

$$H_{mean} = \sqrt{\frac{1}{m} \sum_{\forall(i,j)} \left[\mu_\tau^{(i,j)} - T_{mean} \frac{\mathcal{R}_{abs}(i,j)}{max\mathcal{R}_{abs}} \right]^2} \tag{13}$$

$$H_{mean}^{eff} = \frac{H_{mean}}{\kappa} \tag{14}$$

B. SIMULATION RESULTS FOR THE HOMOGENEOUS EXPLORATION

In this subsection, results for the DQL algorithm for the three different states are obtained. Once the best state representation is chosen, different reward parameters in Table 2 are evaluated for best performance. Finally, DQL and DDQL results are compared.

1) STATE REPRESENTATION EVALUATION

For this evaluation, the parameters used are summarized in Table 4. For this early tuning only the DQL algorithm has been used.

The results are shown in Fig. 9-11 for 20 episodes, with Deep Q Network trained in the best episode. It is clear that the best state representation is the RGB 2 image with relative

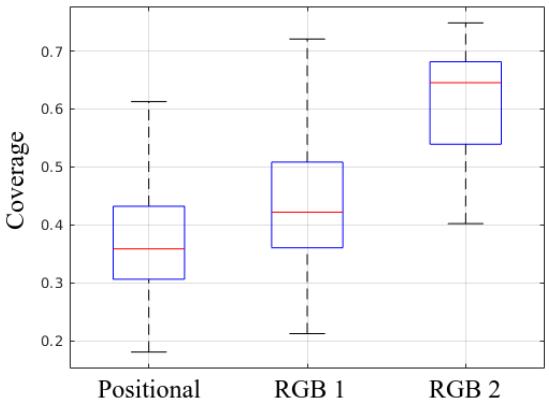

FIGURE 9. Coverage for state comparison.

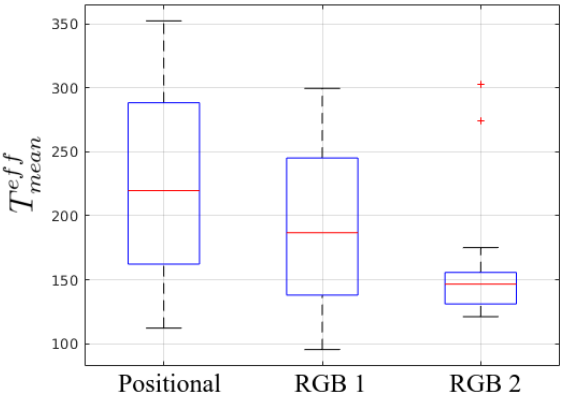

FIGURE 10. T_{mean}^{eff} for the state comparison.

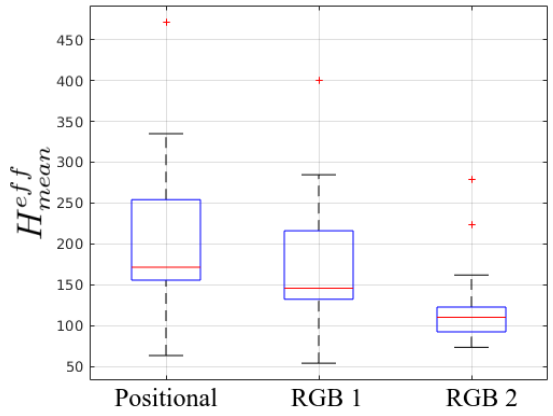

FIGURE 11. H_{mean}^{eff} for the state comparison.

importance matrix R values represented on it. The more information the Deep Q-Learning algorithm can get through the state, the better decisions is able to make. The positional state has the worst performance due to the lack of information generated (35% coverage rates on average). Thus, without any ability to remember the previously visited zones, it results in a path planning with bad coverage. The RGB representation improves significantly the performance; therefore in

TABLE 5. Reward parameters for calibration.

N	$C_{illegal}$	r_{min}	r_{max}
1	-5	-0.5	+1
2	-10	-0.5	+1
3	-5	-1.5	+1
4	-5	-0.5	+1.5
5	-10	-0.5	+5
6	-0.5	-0.5	+1

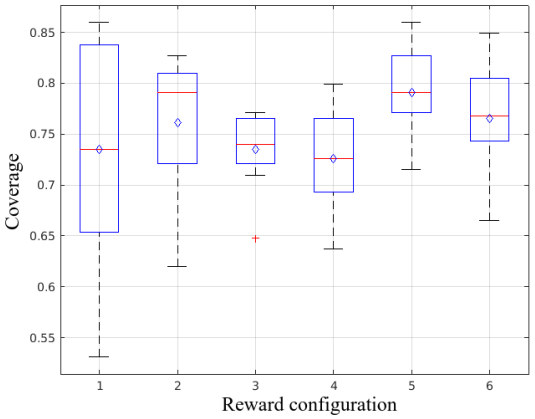

FIGURE 12. Coverage for reward calibration.

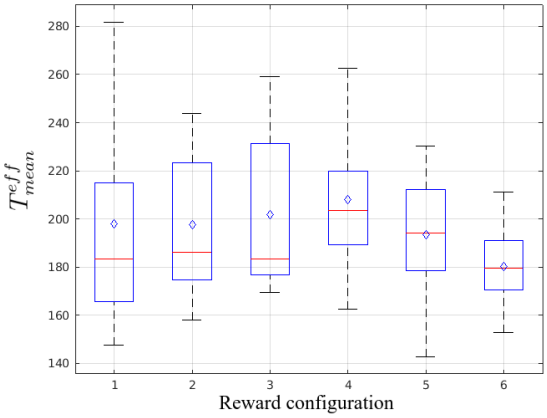

FIGURE 13. T^{eff}_{mean} for reward calibration.

the following evaluations, the complete RGB 2 representation will be used as it returns the best performance in every metric even with a significant lower dispersion.

2) REWARD PARAMETERS EVALUATION

Since the reward parameters affect directly to the agent behavior, a succinct search of the better reward triad is conducted. The different configurations of reward parameters ($C_{illegal}$, r_{min}, r_{max}) tested are included in Table 5.

The results are represented in Fig. 12-14 for 20 episodes, with Deep Q Network trained in the best episode. The calibration process shows that the configuration 5 and 6 have the best results and it has been observed that a high $C_{illegal}$ to r_{max} rate results in a bad performance.

Since there are infinite ways to tune the reward function, this brief selection will help to choose its parameters in a heuristic way. The best reward parameters selected for the following evaluations are the ones in the configuration 5: highest coverage (see Fig. 12), best effective homogeneity (see Fig. 14) and the second lowest effective mean time (see Fig. 13).

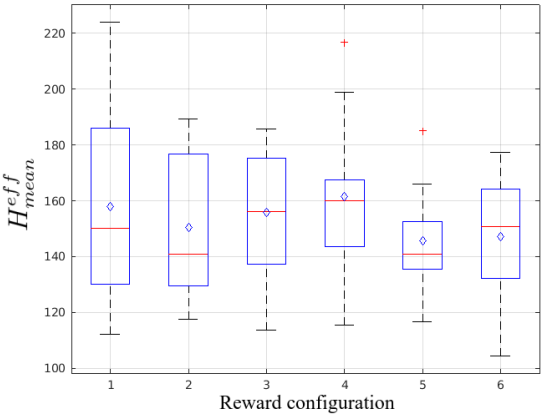

FIGURE 14. H^{eff}_{mean} for reward calibration.

TABLE 6. Nominal parameters for the homogeneous patrolling learning.

Parameter	Value
α	1E-3
Batch Size	250
Mem. Size	10E3
ϵ-decay	7E-4
γ	0.99
Steps	300
Epochs	1500

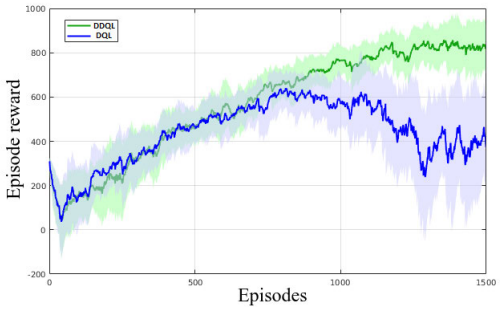

FIGURE 15. Reward through the learning process for DQL and DDQL in the homogeneous case.

C. HOMOGENEOUS PATROLLING RESULTS

Given the selected state representation and the reward parameters, the results of both DQL and DDQL algorithms are presented for the homogeneous patrolling case. In this case, the relative importance matrix R has a value of 255, therefore $R = R_{abs}$, and the objective is to achieve the best accumulative reward possible (pure coverage problem). The hyperparameters of training are summarized in Table 6 and have been selected based on previous results of different authors ([7] and [18]) with good results.

In Fig. 15 the difference between DQL (blue) and DDQL (green) learning is represented. With DDQL the learning

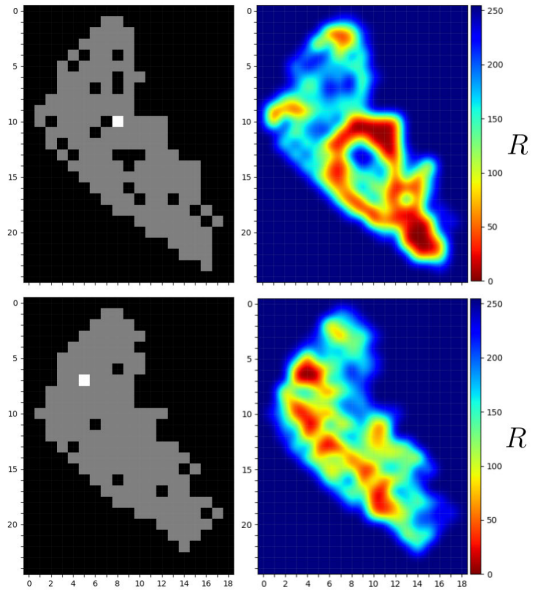

FIGURE 16. Visited cells and *R* matrix as a heat map for the best episode with DQL algorithm (up) and DDQL (down).

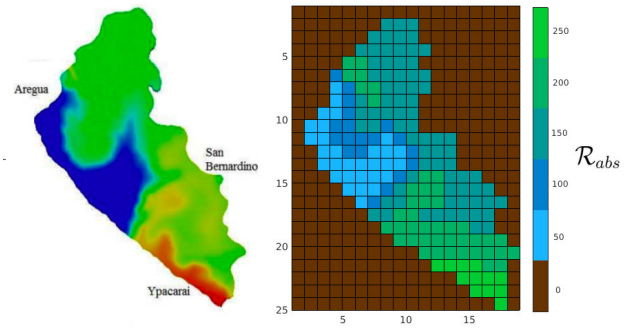

FIGURE 17. Non-homogeneous $\mathcal{R}_{abs}$ proposed based on [31].

process is much more stable than DQL. The catastrophic forgetting issue disappears in DDQL and much higher rewards are obtained at the end of the learning. The DDQL algorithm with this hyperparameter configuration achieves an accumulated reward per episode of 810, whereas DQL only obtains up to 600. In Fig. 16, the relative interest matrix *R* is presented as an interpolated heat map with the visited cell map. The improvement of homogeneity of the coverage between the two algorithms can be seen in Fig. 17. In DQL approach, some zones are excessively visited, while others remain poorly visited.

D. NON-HOMOGENEOUS PATROLLING RESULTS

For the non-homogeneous patrolling problem, an absolute importance matrix $\mathcal{R}_{abs}$ is proposed based on the research of [31] where a map of contamination-sensible zones is found. A discretization of the map has been made resulting in 5 different levels of importance. The non-homogeneous $\mathcal{R}_{abs}$ is shown in Fig. 17.

In Fig. 18, it is shown the learning process and accumulative reward in each epoch of 300 steps simulated using

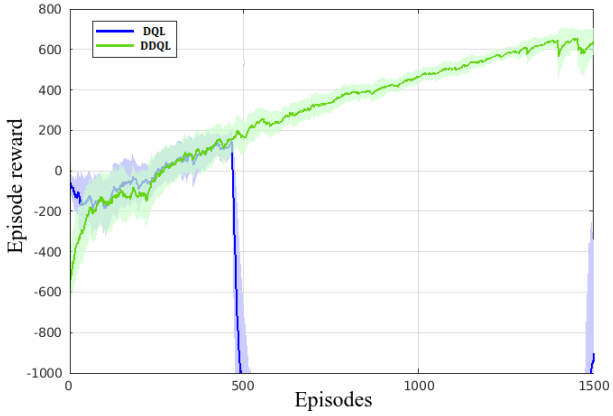

FIGURE 18. Reward through the learning process for DQL and DDQL in the non-homogeneous case.

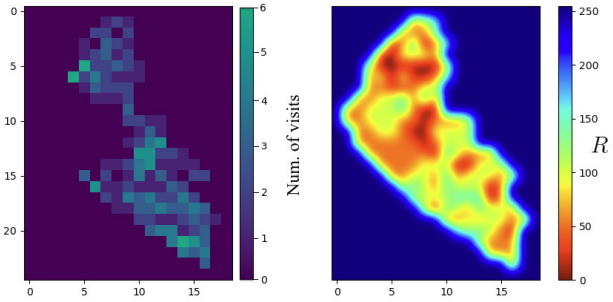

FIGURE 19. Number of visits in cells (left) and *R* matrix as a heat map (right) for an arbitrary episode with DDQL for the non-homogeneous case.

the same hyperparameters included in Table 6. It is clear that the DQL algorithm cannot stabilize the learning whereas DDQL can. This issue demonstrates the good performance of DDQL to train the agent through the episodes. The maximum achieved reward in the DDQL is 621, which is lower than the maximum reward in the preceding case due to the lower values of $\mathcal{R}_{abs}$. The paths generated by the algorithm in the latest training episodes achieve the task of exploring the most important cells multiple times, while the least important remains visited one time or unvisited since they are not important at all or have little importance compared to the others.

In Fig. 19, the results of the best episode with the trained DDQL can be seen. The more important cells, which are mostly located in the south, are visited more times than the least important ones. The agent visits some cells six times, returning an excessive of redundancy since $\mathcal{R}(i,j)_{abs}$ admits 5 visits until becoming zero. This shows some lack of optimality besides the acceptable performance of the algorithm. Since the algorithm provides good solutions but very perfectible ones, it is necessary to tune the hyperparameters for obtaining a better performance.

E. HYPERPARAMETRIZATION ANALYSIS

It is well-known that the value of training hyperparameters as well as the structure of the neural network influence in some degree of the final learning performance. To enhance

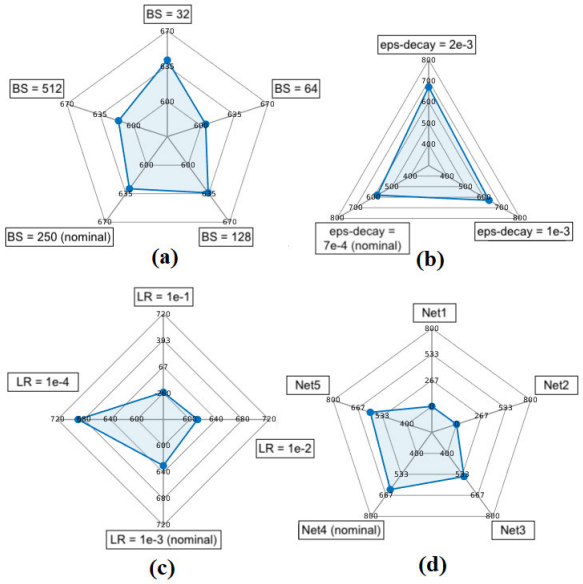

FIGURE 20. Average reward obtained in the last 50 episodes for each batch size (a), ϵ-decay (b), learning rate (c) and network size (d) value.

the obtained results for the ASV implementation, we study the effect of the most important hyperparameters in the non-homogeneous patrolling case.

The hyperparameters to vary are α, the batch size, the ϵ-decay rate and the neural network size. Different simulations are executed, changing one parameter value each time and by selecting the hyperparameters that achieve the best performance, a final tuned algorithm is obtained.

1) BATCH SIZE

We start from the nominal values from Table 6. Five different values for the batch size are tested: 32, 64, 128, 250 (nominal) and 512. The higher the batch size, the more previous experiences will be used to find the relation between a state-action pair (s, a) and its reward r. According to the current literature, it is not clear if a high batch size improves the learning. In [29], it is shown that higher batch sizes led to bad performance since the algorithm moves away from the off-policy paradigm. In Fig. 20(a) the result of the average reward obtained in 50 episodes with the trained Q-network is shown. The performance for each value of batch size can be seen in Fig. 21. According to the results, the best batch size is 32 with a maximum reward of 653.

2) ϵ-DECAY RATE

The ϵ-decay rates determines the level of greed of the behavioral policy according to the ϵ-greedy algorithm. Three different rates have been tested so that each makes ϵ its own minimum value (0.01) in a certain episode number. These three values are: 0.002 (reaches its minimum in episode 500), 0.001 (reaches its minimum in episode 1000) and 0.0007 (reaches its minimum in episode 1400). In Fig. 20(b) the result of the average reward obtained in 50 episodes with the trained Q-network is shown. The performance for each value of the learning rate used in simulation can be seen in Fig. 22. The best ϵ-decay rate will be 2E-3 with a maximum reward of 698.

TABLE 7. Different net sizes tested.

Net size	No of filters	Size of 1^{st} dense layer	Size of 2^{nd} dense layer
Net1 (x0.125)	1	128	128
Net2 (x0.25)	2	256	256
Net3 (x0.5)	4	512	512
Net4 (x1,nom.)	8	1024	1024
Net5 (x2)	16	2048	2048

3) LEARNING RATE

The learning rate affects on how quickly gradient of the loss function updates each parameter (weights and biases) of the neural network. If this parameter is very large, it will cause instabilities in training and if it is very small, the convergence will be excessively slow and a sub-optimal solution may be reached. We have tested 4 values of this parameter: 1E-1, 1E-2, 1E-3 (nominal) and 1E-4. In Fig. 20(c) the result of the average reward obtained in 50 episodes with the trained Q-network is shown. The performance with each value the learning rate can be seen in Fig. 23. The best learning rate is 1E-4 with a maximum reward of 720.

4) NEURAL NETWORK SIZE

We have changed the size of the Q convolutional network to study how the number of neurons and filters affects the performance of the agent. An excessively large network will have many of their neurons with weights that tend to 0 and a very small network lacks the plasticity to generalize the optimal Q function, so it must conform to an intermediate size. The different network architectures are listed in Table 7.

In Fig. 20(d) the result of the average reward obtained in 50 episodes with the trained Q-network is shown. The performance with each network size can be seen in Fig. 24. On the one hand, the simulations show that decreasing the size results in a bad performance as the neural function cannot estimate adequately the Q values. On the other hand, increasing the network size do not improve the performance and only contributes to a slower execution of the learning process. The selected size is the previously used value (Net4) since it is the smaller with the best performance.

5) SIMULATION RESULTS FOR THE BEST HYPERPARAMETERS

Selecting the best hyperparameters in the previous study, the agent achieves better results in both homogeneous and non-homogeneous patrolling cases. By training with the new parameters (listed in Table 8), the results show a much better learning performance in the first case (see Fig. 25) as well in the second case (see Fig. 26). The improvement is not only achieved in terms of rewards, with an improvement of the 50% and 13% for each case, but also in the standard deviation

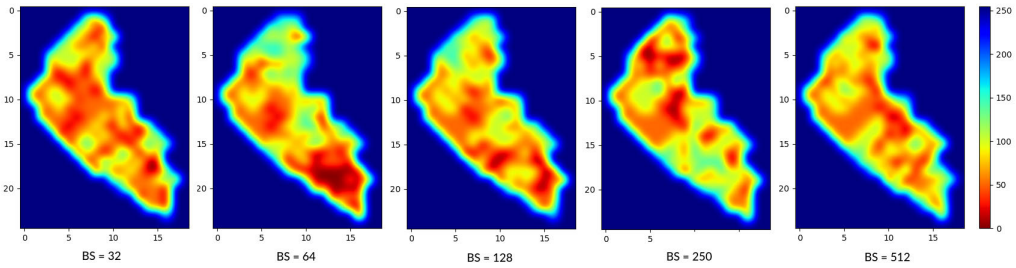

FIGURE 21. Heat map of the best episode for each batch size value.

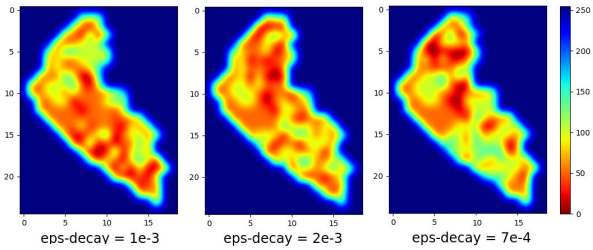

FIGURE 22. Heat map of the best episode for each ϵ-decay value.

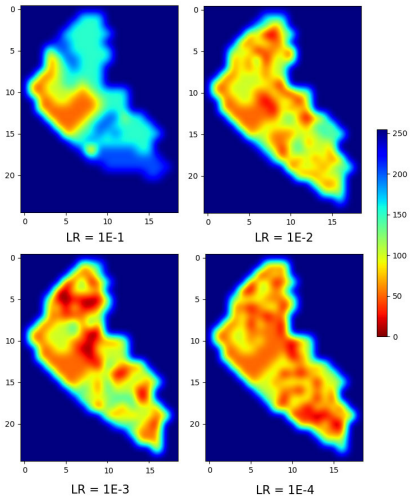

FIGURE 23. Heat map of the best episode for each learning rate value.

TABLE 8. Best parameters selected.

Parameter	Value
α	1E-4
Batch Size	128
Mem. Size	10E3
ϵ-decay	2E-3
γ	0.99
Steps	300
Epochs	1500

of the average reward in the final epochs. A low deviation of the reward implies a more homogeneous solution for every path generated i.e. a better ability to obtain a good solution for any initial conditions.

F. COMPARISON WITH OTHER APPROACHES

To illustrate the efficiency and the validity of the DDQL proposed approach, three algorithms for patrolling are used: a naive random exploration, a lawn mower complete coverage path and a Policy Gradient Reinforcement Learning (REINFORCE). The random exploration will avoid illegal actions and the lawn mower path will cover always the whole area in a cyclic path from a random position (see Fig. 27). Policy Gradient (PG) will work similarly to DDQL but taking descent gradient steps to optimize the behavioral policy $\pi(s;\theta)$ directly. In this RL approach, the same neural network is used for the policy and the equivalent hyperparameters (γ, Learning Rate, ...) remains the same in both DDQL and PG.

1) HOMOGENEOUS PATROLLING

For the homogeneous patrolling, 20 different epochs are simulated. The simulation results obtained are shown in Table 9. The simulations show the good performance of the proposed algorithm in the homogeneous case, with a 64% improvement in T^{eff}_{mean} respect to the lawn mower approach and for the randomized path planner also. For the Policy Gradient, it can be seen the improvement is insufficient due to a slower training. For 1500 epochs, DDQL shows a much better performance with a 25% improvement of the mean time between visits from the PG.

In Fig. 28 the heat map of the path generated in every algorithm is displayed. In this case, the proposed algorithm is able to revisit cells (mostly on the center of the lake due to the lower state values in the boarders) to maintain the average T^{eff}_{mean} lower than the lawn mower approach. In the lawn mower path, several zones remain forgotten since the agent completes a full cycle and DDQL approach seems to avoid this problem by covering from the center of the lake to its limits. Something similar happens to the PG approach: the algorithm is not able to compute a better solution in the given number of epochs and the resulting paths remains far from optimal with an excessive useless redundancy.

2) NON-HOMOGENEOUS PATROLLING

For the non-homogeneous patrolling, 20 different epochs are also simulated. The metrics obtained are shown in Table 10. The simulations show the good performance of the proposed algorithm in the non-homogeneous case, with a 30%

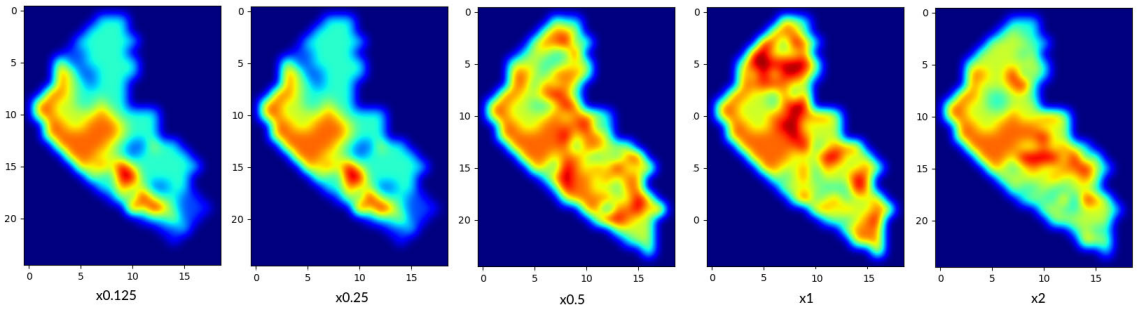

FIGURE 24. Heat map of the best episode for each learning rate value.

TABLE 9. Metrics for the homogeneous case.

	Lawn mower		Randomized		DDQN		PG	
	Mean	Std Dev.	Mean	Std Dev.	Mean	Std Dev.	Mean	Std Dev.
Coverage	1	-	0.50	0.10	0.93	0.02	0.64	0.15
T^{eff}_{mean}	178.5	-	178.7	57.26	115.35	4.47	144.31	11.67
H^{eff}_{mean}	25.41	-	148.7	58.76	59.91	2.29	78.9	7.33
$Reward$	991.1	-	423.5	128.9	1180.4	46.8	565.2	44.15

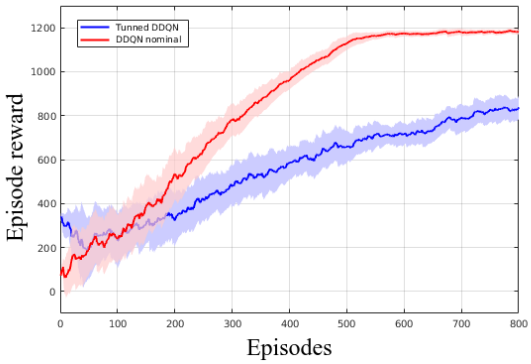

FIGURE 25. Episode reward in the learning process for tuned and nominal hyperparameters in homogeneous exploration.

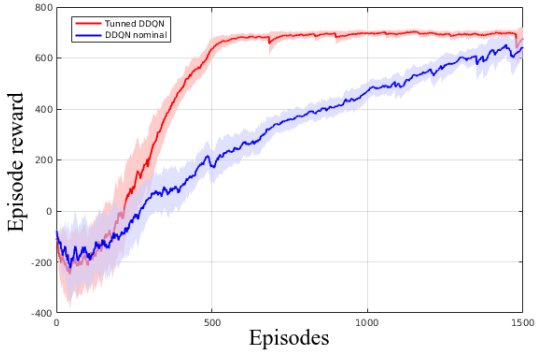

FIGURE 26. Episode reward in the learning process for tuned and nominal hyperparameters in non-homogeneous exploration.

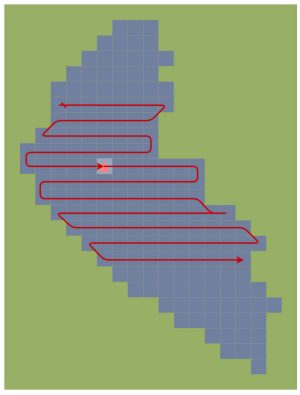

FIGURE 27. Lawn mower trajectory.

improvement in T^{eff}_{mean} respect to the lawn mower approach and a 21% improvement in the random path planner. The improvement is also noticeable in the effective homogeneity. For the PG algorithm, the improvement over lawn-mower and random-exploring is noticeable but, as happened in the homogeneous case, DDQL outperform in every metric. Once again, DDQL shows, for the Ypacaraí case, a better learning speed and a lower variance.

In Fig. 29 the heat map of the path generated in every algorithm is displayed. In this case, the proposed algorithm is able to choose the most interesting cells and avoid those cells of low interest. Such low interest could be due to those cells are already visited or because there are no algae blooms in it. The lawn mower approach is not able to avoid those zones and shows a total lack of intelligence. In our approach, the most important zones in the south are visited 4 to 5 times with an average visiting-time separation of 100 steps.

3) DISCUSSION OF THE RESULTS

Finally, we discuss the previous results with the most relevant aspects of the simulations.

- The selection of the state and the results have demonstrated that the more information the DQL/DDQL algorithm has, the better performance is achieved. As an off-policy algorithm, DQL approach needs to extract

TABLE 10. Metrics for the non-homogeneous case.

	Lawn mower		Randomized		DDQN		PG	
	Mean	Std Dev.	Mean	Std Dev.	Mean	Std Dev.	Mean	Std Dev.
Coverage	1	-	0.51	0.11	0.88	0.02	0.65	0.11
T^{eff}_{mean}	130.4	-	121.9	21.36	100.69	3.29	115.61	17.8
H^{eff}_{mean}	54	-	110.7	28.83	43.02	3.42	68.15	6.9
$Reward$	605.5	-	123.6	46.7	702.55	10.2	548.15	164.45

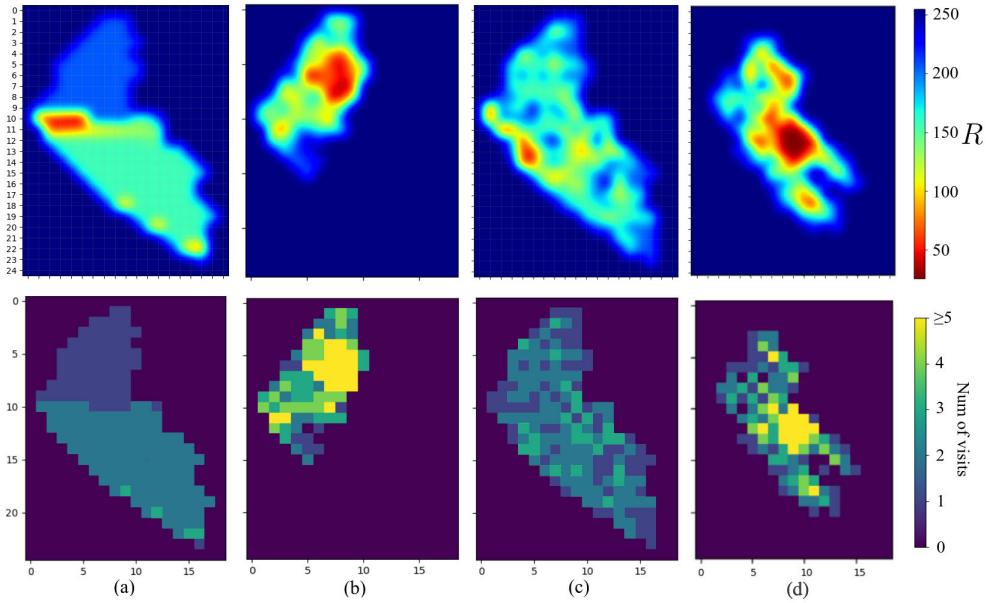

FIGURE 28. Best results for lawn mower path (a), random exploration (b), the proposed DDQL algorithm (c) and Policy Gradient (d) for the homogeneous coverage.

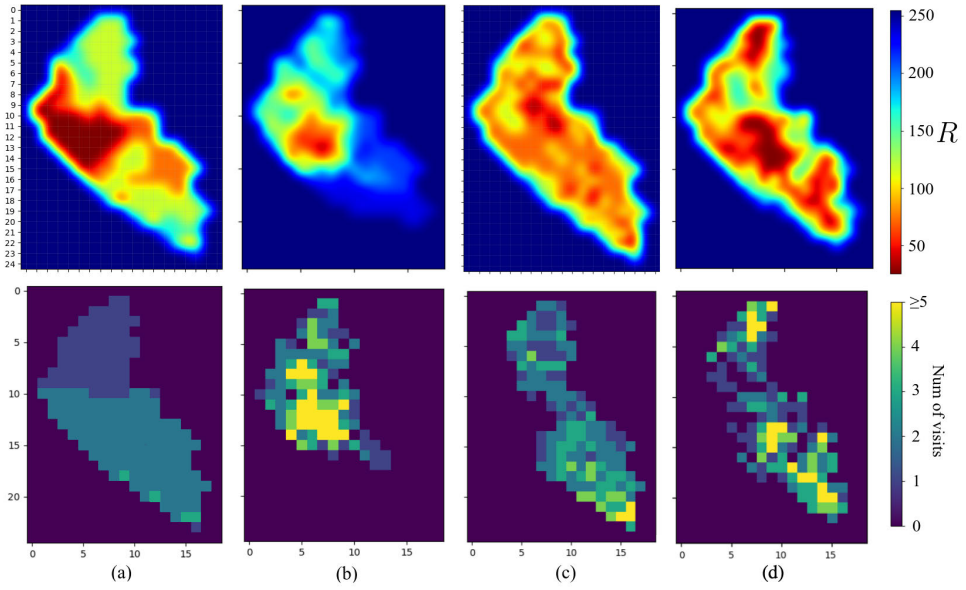

FIGURE 29. Best results for lawn mower path (a), random exploration (b), the proposed DDQL algorithm (c) and Policy Gradient (d) for the non-homogeneous coverage.

the correlation between the state-action pair (s, a) and its next state and reward (s', r). Without a proper state representation, the neural network will be inefficient, resulting in a very bad performance.

- The reward function, as a model of *what we want the agent to achieve*, plays an important role in the final performance. With a simple search of the reward parameters, the efficiency could be enhanced.

- In the DQL/DDQL comparison, the DQL algorithm shows worse results than DDQL as expected. The catastrophic forgetting phenomenon compromises the learning stability causing the agent to learn poorly and slower. The DDQL approach is proven to be a good RL algorithm to train in the two Ypacaraí patrolling cases.
- Besides the acceptable results of the DDQL approach in both exploration cases, a hyperparameter tuning is necessary to obtain a competitive performance respect to other approaches. The proposed tuning returns a much higher rewards and a better coverage of the lake.
- With the tuning, the DDQL is much better than the lawn mower path planner and, of course, the random exploration algorithm. The proposed algorithm is able to improve lawn mower path by 64% in terms of T_{mean}^{eff} for the homogeneous case and by 30% for the non-homogeneous problem, proving the good performance of the algorithm.
- Comparing PG and DDQL, it has been observed that Policy Iteration algorithms tends to be slower in learning with a worse sample efficiency. Another remarkable difference lies in the fact that DDQL has a much smaller variance in the expected return once trained. This characteristic of *performance homogeneity* is very desirable because it guarantees a better performance epoch after epoch regardless the initial point. Thus, the standard deviation of the reward results in an order of magnitud higher in PG.

VI. CONCLUSION

Reinforcement Learning approaches have been demonstrated to be flexible and efficient in the resolution of high-complexity problems with dynamic scenarios as the patrolling problem of the Ypacaraí Lake. Double Deep Q-Learning is a good approach to design a path planner for the homogeneous and non-homogeneous patrolling problem since: i) it does not need a model of the environment and ii) due to its off-policy behavior, it is not necessary any specific behavioral policy to achieve the optimality. Furthermore, when non-modeled obstacles are encountered or a change in the map is defined, it is possible with retraining to incorporate such new information. In addition, the trajectory generator and the collision-avoidance system could decide an action is impossible and some other action must be chosen, the Deep Q-Network could recalculate the better action for the new state or even select the the second-best action given by $Q(s, a; \theta)$ (this remarks the reactivity of the method) causing a non-optimal action but allowing the ASV to perform robustly in the real environment.

In the homogeneous patrolling case, the DDQL paths achieve virtually the complete area coverage task with an average 93% of the covered area with a remarkable diminution of the mean team between visits of every zone. In the non-homogeneous exploration case, it is also proven to be effective in the patrolling task with a good ability to travel to zones with higher importance. It is also confirmed the hypothesis explained in [6] for the DQL worse learning process. The duplication of the model function by using a Double Deep Q-Learning variation enhances the average rewards in every simulated case and the learning instabilities are avoided. Besides PG algorithm could be trained during more epochs in order to obtain higher rewards, DDQL has been proven sufficiently efficient for this application. Furthermore, the high variance of the rewards in PG results in a undesirable situation for the real application and since DDQL achieves a remarkable small deviation of its metric, it results in a more practical algorithm.

In order to achieve these acceptable results, it is required to tune the hyperparameters and the reward function as it was explained in the previous sections. It has been proven the Deep Q-Learning algorithms have high hyperparameter sensibility, and poorly-chosen parameters end up in a bad learning process even in learning instabilities.

For the next steps, the multi-agent exploration paradigm is planned to be achieved as the model-free behavior of the algorithm allows an *almost* immediate adaptation of the DDQL algorithm. To improve the performance of the path planner, some modifications are proposed like the use of Long Short-Term Memory (LSTM) neurons. LSTM networks are recurrent neural networks which have the capability to use the subsequent states to consider both short and long-term temporal dependencies into the learning. This recurrent networks are expected to return better results for a Partial Observable Markov Decision Process (POMDP), where the complete state is not fully observable. This will be very useful when adding sensor uncertainties to the scenario and the possibility of have moving obstacles into the lake.

REFERENCES

[1] M. E. L. Arzamendia, S. L. T. Marín, and D. G. Reina, "Reactive evolutionary Path planning for autonomous surface vehicles in lake environments," Ph.D. dissertation, Dept. Electron., Escuela Técnica Superior de Ingeniería de Sevilla, Seville, Spain, 2018.

[2] M. E. López Arzamendia, D. Gregor, D. G. Reina, and S. L. Toral, "An evolutionary approach to constrained Path planning of an autonomous surface vehicle for maximizing the covered area of Ypacarai lake," *Soft Comput. J.*, vol. 25, no. 5, pp. 1723–1734, 2019.

[3] J. C. Ho and A. M. Michalak, "Challenges in tracking harmful algal Blooms: A synthesis of evidence from lake erie," *J. Great Lakes Res.*, vol. 41, no. 2, pp. 317–325, Jun. 2015.

[4] J. SÃĺnchez-García, J. M. García-Campos, M. Arzamendia, D. G. Reina, S. L. Toral, and D. Gregor, "An evolutionary approach to constrained Path planning of an autonomous surface vehicle for maximizing the covered area of ypacarai lake," *Comput. Commun.*, vol. 119, pp. 43–65, 2018.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[6] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell., AAAI*, 2016, pp. 2094–2100.

[7] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," in *Proc. 13th Int. Conf. Distrib. Smart Cameras*, Sep. 2019, pp. 1–6.

[8] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm Evol. Comput.*, vol. 44, pp. 228–246, Feb. 2019.

[9] L. Huang, M. Zhou, K. Hao, and E. Hou, "A survey of multi-robot regular and adversarial patrolling," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 894–903, Jul. 2019.

[10] K. S. Kappel, T. Cabreira, J. L. Marins, L. B. de Brisolara, and P. R. Ferreira, "Strategies for patrolling missions with multiple UAVs," *J. Intell. Robotic Syst.*, vol. 99, pp. 499–515, Sep. 2019.

[11] J. Xiao, G. Wang, Y. Zhang, and L. Cheng, "A distributed multi-agent dynamic area coverage algorithm based on reinforcement learning," *IEEE Access*, vol. 8, pp. 33511–33521, 2020.

[12] A. A. Adepegba, S. Miah, and D. Spinello, "Multi-agent area coverage control using reinforcement learning," in *Proc. 29th Int. Florida Artif. Intell. Res. Soc. Conf., FLAIRS*, 2016, pp. 368–373.

[13] Y. Cheva, "Theoretical analysis of the multi-agent patrolling problem," in *Proc. IEEE/WIC/ACM Int. Conf. Intell. Agent Technol. (IAT)*, Sep. 2004, pp. 302–308.

[14] I. A. Hameed, "Coverage path planning software for autonomous robotic lawn mower using Dubins' curve," in *Proc. IEEE Int. Conf. Real-Time Comput. Robot. (RCAR)*, Jul. 2017, pp. 517–522.

[15] M. Arzamendia, I. Espartza, D. G. Reina, S. L. Toral, and D. Gregor, "Comparison of eulerian and Hamiltonian circuits for evolutionary-based path planning of an autonomous surface vehicle for monitoring ypacarai lake," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 4, pp. 1495–1507, Apr. 2019.

[16] M. Arzamendia, D. Gutierrez, S. Toral, D. Gregor, E. Asimakopoulou, and N. Bessis, "Intelligent online learning strategy for an autonomous surface vehicle in lake environments using evolutionary computation," *IEEE Intell. Transp. Syst. Mag.*, vol. 11, no. 4, pp. 110–125, Sep. 2019.

[17] B. Liu, Y. Zhang, S. Fu, and X. Liu, "Reduce UAV coverage energy consumption through actor-critic algorithm," in *Proc. 15th Int. Conf. Mobile Ad-Hoc Sensor Netw. (MSN)*, Dec. 2019, pp. 332–337.

[18] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "Uav coverage path planning under varying power constraints using deep reinforcement learning," 2020, *arXiv:2003.02609*. [Online]. Available: http://arxiv.org/abs/2003.02609

[19] L. Lv, S. Zhang, D. Ding, and Y. Wang, "Path planning via an improved DQN-based learning policy," *IEEE Access*, vol. 7, pp. 67319–67330, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8721655

[20] Z. Wang, N. D. Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proc. ICML*, New York City, NY, USA, vol. 4, 2016, pp. 2939–2947.

[21] R. Shah, Y. Jiang, J. Hart, and P. Stone, "Deep R-learning for continual area sweeping," Tech. Rep., 2020.

[22] A. K. Lakshmanan, R. E. Mohan, B. Ramalingam, A. Vu Le, P. Veerajagadeshwar, K. Tiwari, and M. Ilyas, "Complete coverage path planning using reinforcement learning for tetromino based cleaning and maintenance robot," *Autom. Construct.*, vol. 112, Apr. 2020, Art. no. 103078.

[23] R. S Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[24] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 610–617, Apr. 2019.

[25] P. Ruvolo, I. Fasel, and J. Movellan, "Optimization on a budget: A reinforcement learning approach," in *Proc. Adv. Neural Inf. Process. Syst. Conf.*, May2009, pp. 1385–1392.

[26] Q. Zhang, J. Lin, Q. Sha, B. He, and G. Li, "Deep interactive reinforcement learning for path following of autonomous underwater vehicle," *IEEE Access*, vol. 8, pp. 24258–24268, 2020.

[27] H. Eberhardt, V. Klumpp, and U. D. Hanebeck, "Density trees for efficient nonlinear state estimation," in *Proc. 13th Int. Conf. Inf. Fusion*, Jul. 2010, pp. 1–8.

[28] F. Peralta, M. Arzamendia, D. Gregor, D. G. Reina, and S. Toral, "A comparison of local path planning techniques of autonomous surface vehicles for monitoring applications: The ypacarai lake case-study," *Sensors*, vol. 20, no. 5, p. 1488, Mar. 2020.

[29] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 1–17.

[30] I. Khan, P. M. Roth, A. Bais, and H. Bischof, "Semi-supervised image classification with huberized Laplacian support vector machines," in *Proc. IEEE 9th Int. Conf. Emerg. Technol. (ICET)*, Dec. 2013, pp. 1–6.

[31] L. C. Oporto, A. Derlis Ramírez, J. D. Varela, and C. E. Schaerer, "Analysis of contaminant transport under wind conditions on the surface of a shallow lake," *Mecánica Computacional*, vol. 34, no. 31, pp. 2155–2163, 2016.

[32] A. Rodríguez, "Estudio de la contaminación del lago ypacaraí e introducción de un dron acuático para el monitoreo de la calidad del agua," M.S. thesis, Dept. Electron., Universidad de Sevilla, Seville, Spain, 2019.

[33] A. Cahill, "Catastrophic forgetting reinforcement-learning environments," Ph.D. dissertation, Dept. Comput. Sci., Univ. Otago, Dunedin, New Zealand, 2010.

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, New York City, NY, USA, vol. 4, 2016, pp. 2850–2869.

[35] Z. Wang, V. Mnih, V. Bapst, R. Munos, N. Heess, K. Kavukcuoglu, and N. Freitas, "Sample efficient actor-critic with experience replay," in *Proc. 5th Int. Conf. Learn. Represent., ICLR Track*, 2019, pp. 2–21.

[36] S. Ravichandiran, *Hands-on Reinforcement*. Birmingham, U.K.: Packt, 2018.

[37] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor—Critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

SAMUEL YANES LUIS was born in Tenerife, Spain, in 1997. He received the M.S. degree in electronics and robotics engineering from the University of Seville, Spain, in 2019, where he is currently pursuing the Ph.D. degree. His main research interests include reinforcement learning and machine learning for robotics applications, autonomous vehicles control, and artificial vision.

DANIEL GUTIÉRREZ REINA received the B.E. degree (Hons.) in electronic engineering, the M.S. degree in electronics and telecommunications, and the Ph.D. degree (Hons.) in electronic engineering from the University of Seville, Seville, Spain, in 2009, 2011, and 2015, respectively. He worked as an Assistant Professor with Loyola University, from October 2018 to April 2019. He is currently working as an Assistant Professor with the Electronic Engineering Department, University of Seville. He has been a Visitor Researcher with Liverpool John Moores University (U.K.), the Free University of Berlin (Germany), the Colorado School of Mines (USA), and Leeds Beckett University (U.K.). His current research interests include the application of meta-heuristic algorithms to solve wireless multi-hop network optimization problems, such as MANETs, VANETs, DTNs, and FANETs. He has published over 40 articles in JCR journals with impact factor. He is a part of the editorial board of several journals, such as *International Journal of Distributed Sensor Networks* (SAGE), *Electronics* (MDPI), *Future Internet* (MDPI), *Wireless Communications and Mobile Computing* (Hindawi), and organizing numerous SI for these journals.

SERGIO L. TORAL MARÍN was born in Rabat, Morocco, in 1972. He received the M.S. and Ph.D. degrees in electrical and electronic engineering from the University of Seville, Spain, in 1995 and 1999, respectively. He is currently a Full Professor with the Department of Electronic Engineering, University of Seville. He is an author or coauthor of 95 papers in major international peer-reviewed journals (with JCR impact factor) and of over 100 papers in well-established international conferences and workshops. His main research interests include ad hoc networks and their routing protocols, flying ad hoc networks, deployment of unmanned vehicles, and intelligent transportation systems.