Q-Learning for Autonomous Mobile Robot Obstacle Avoidance

Tiago Ribeiro
Industrial Electronics Dept.
University of Minho
Guimarães, Portugal
A71157@alunos.uminho.pt

Fernando Gonçalves
Mechanical Engineering Dept.
University of Minho
Guimarães, Portugal
A70569@alunos.uminho.pt

Inês Garcia
Industrial Electronics Dept.
University of Minho
Guimarães, Portugal
A70557@alunos.uminho.pt

Gil Lopes
Industrial Electronics Dept.
ALGORITMI Center
University of Minho
Guimarães, Portugal
gil@dei.uminho.pt

A.Fernando Ribeiro
Industrial Electronics Dept.
ALGORITMI Center
University of Minho
Guimarães, Portugal
fernando@dei.uminho.pt

***Abstract*— An approach to the problem of autonomous mobile robot obstacle avoidance using Reinforcement Learning, more precisely Q-Learning, is presented in this paper. Reinforcement Learning in Robotics has been a challenging topic for the past few years. The ability to equip a robot with a powerful enough tool to allow an autonomous discovery of an optimal behavior through trial-and-error interactions with its environment has been a reason for numerous deep research projects. In this paper, two different Q-Learning approaches are presented as well as an extensive hyperparameter study. These algorithms were developed for a simplistically simulated Bot'n Roll ONE A (Fig. 1). The simulated robot communicates with the control script via ROS. The robot must surpass three levels of iterative complexity mazes similar to the ones presented on RoboParty [1] educational event challenge. For both algorithms, an extensive hyperparameter search was taken into account by testing hundreds of simulations with different parameters. Both Q-Learning solutions develop different strategies trying to solve the three labyrinths, enhancing its learning ability as well as discovering different approaches to certain situations, and finishing the task in complex environments.**

Keywords*— *Reinforcement Learning, Q-Learning, Autonomous Mobile Robot, Obstacle Avoidance, Robotics, Simulated Robot, Bot'n Roll ONE A, RoboParty

I. INTRODUCTION

Research on autonomous mobile robots aims at building intelligent behavior systems. One of the main issues in autonomous mobile robot's navigation is obstacle avoidance capability. These must be able to adapt its movement or trajectory to react adequately in unknown, complex and dynamic environments. There are two considerably different types of obstacle avoidance: when a complete knowledge of the environment is assumed, and when said knowledge is partially or totally unknown. In the first case, classical global motion and path planning strategies as described in [1] can be applied. Still, the efficiency of such techniques rapidly decreases in more complex and unstructured environments since considerable modeling is needed. A suitable method for achieving the goal

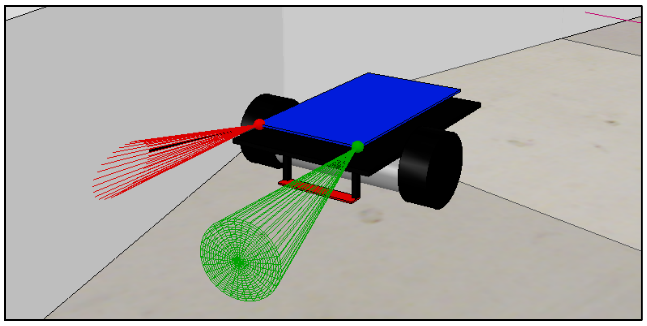
Fig. 1. Simulated environment screenshot. Note that the robot has its infra-red obstacle sensors as red and green whether or not an obstacle is being detected, respectively.

on such environments is Reinforcement learning, since a complete environment knowledge is not necessary, and the robot has an online learning capability.

Reinforcement Learning enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment. Instead of explicitly detailing the solution to a problem, in reinforcement learning, the designer of a control task provides feedback in terms of a scalar objective function that measures the one-step performance of the robot.

An agent's primary goal is to maximize the accumulated reward over its lifetime. In episodic tasks such as the one described in this paper, a task is restarted after every end of an episode. The objective is to maximize the total reward per episode. In this reinforcement learning problem, the agent and its environment are modeled being in a state $s \in S$ and can perform actions $a \in A$.

The agent in this case study is a simulated Bot'n Roll One A (developed by botnroll.com) robot as shown in (Fig. 1). The environments are three iterative complexity simulated mazes. To solve this episodic task, the robot must find the labyrinth's solution avoiding walls that come through the episode unrolling.

Q-Learning is the reinforcement learning technique chosen to solve this obstacle avoidance problem. The goal of

Q-Learning is to learn the optimal policy, which tells an agent the best action to take under a specific set of circumstances. This technique does not require a model of the environment (model-free) and can learn the optimal policy regardless of the behavior policy (off-policy). This paper compares two different Q-Learning approaches to the same problem. Their differences are described in section IV.H.

II. Related Work

The Open AI Team devoted great efforts into creating Open AI Gym [2]. Open AI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It has been a credited benchmarking system for a wide variety of reinforcement learning systems. It supports teaching agents such as robotics as described in [3]. However, the state-space and action-space of both environments do not have similarities with the one intended to be developed.

Different reinforcement algorithms have been applied to autonomous mobile robots that try to solve the obstacle avoidance problem. Since every robot is developed differently to better serve its purpose, there is no benchmarking for this kind of problem such as the ones available on Open AI Gym. These can be separated into three different input data categories: Low-level Obstacle Sensors (such as Ultrasonic or InfraRed), LIDARs and Cameras.

Robots using cameras for obstacle avoidance, such as [4] use deep reinforcement learning techniques merged with convolutional layers. This type of strategies allows the robot to learn image characteristics crucial for better performance. The usage of LIDARs as presented in [5] grants a 2D environment information. The beams spread over the agent's field of view granting detailed environment information. This input data category has been the most commonly described and used in robotics. For low-level obstacle sensors, the state-space is generally lower than the ones previously discussed. These are commonly used to introduce new algorithms or improvements to existing ones. The most similar agents, state-space and action-space to the case described in this paper can be analyzed in [6].

Classical control algorithms may also be used to detect and avoid obstacles as described in [7]. However, the incapability of learning from experience and adjusting to never seen situations led researchers to invest in reinforcement learning algorithms.

III. Problem Definition

The proposed task is part of RoboParty [8] challenges, named "RoboParty Obstacle Maze". RoboParty is an educational event where the participants are taught every step on how to build autonomous mobile robots. Every team receives an educational robotic kit called Bot'n Roll One A [9] Arduino based. Bot'n Roll One A (Fig. 2) has several inputs, due to its dual microprocessor and Arduino shields compatibility. As embedded inputs, the robot has two infra-red obstacle sensors, a line follower board, Wi-Fi communication, accelerometers, gyroscopes a compass and color sensors. As outputs, the robot is equipped with two DC motors (differential drive), pan and tilt servos, an LCD and a claw.

The obstacle challenge consists of a maze of approximately20 cm height walls, with an entrance point and an

Fig. 2. Bot n' Roll One A [9].

exit point. The teams must only use the robots infra-red obstacle sensors to avoid the maze walls. The robot must not have any modification to its original form to guarantee all teams compete under the same conditions.

Both the environment (RoboParty Obstacle Maze) and the agent (Bot'n Roll One A) were designed on V-Rep [10] graphical simulation environment. The agent was designed as a simplified model using only primitive shapes. It allowed a more simplistic design and an automatic inertial among other value calculations. The simulated agent has other sensors designed, however as the challenge rules state, the only sensors allowed are the infra-red obstacle sensors. As for the outputs, the control script calculates values for both the differential drive DC motors.

For the V-Rep simulator to communicate with the control software, ROS architecture was implemented. Two nodes were created, one for the simulator and other for the control script. Several topics were created, for data transition between the two programs and simulation control such as stop, pause, and start.

The control script is where reinforcement learning is implemented. It receives the sensor data (infra-red obstacle sensors values) from the simulator node. Afterward, it processes that information through the Q-learning method, resulting in the desired outputs for the sample state. The outputs are then sent to the simulator node. From the control script, the initial simulation parameters and the simulation control parameters are also sent.

The agent is presented with three different and iterative complexity mazes. To achieve success the agent must be able to reach the end point, thus solving the mazes. The first maze (approximately 1.2m long) consists of a straight narrow line. The second maze (approximately 5.0m long) has two curves to both sides, which force the robot to learn how to perform turns on non-straight walls and avoiding obstacles from both sides. The third maze (approximately 8.3m long) has curves on both sides, slim and narrow wall distances, sharp edges and deep turns. The distance between walls is on average 0.5m. In Fig. 3, the three mazes can be visualized as well as their starting points represented by the three robots, and their endpoints represented by the magenta lines.

IV. Methodologies

A. *Temporal Difference Learning*

Temporal Difference Learning (also known as TD Learning) is a central and novel idea to reinforcement learning. TD Learning is a combination of Monte Carlo and Dynamic Programming ideas. TD Learning, like Monte Carlo, can learn directly from raw experience without a model of the environment's dynamics. Like Dynamic Programming, it updates estimates partly based on the learned estimates, without

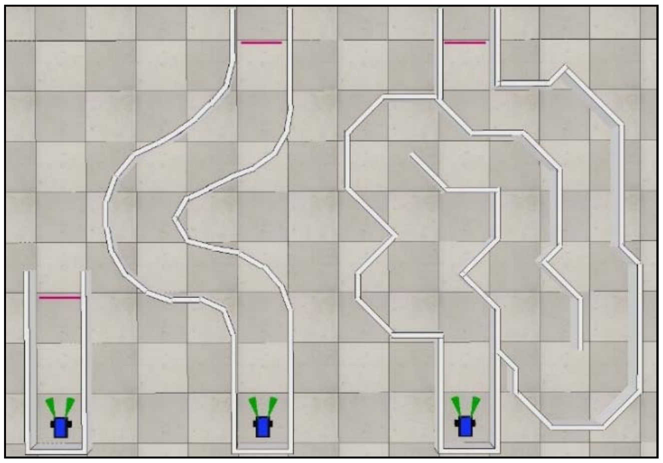
Fig. 3. Three mazes screenshot increasing in complexity (from left to right), the start point of each maze is represented by the robots, and the end point by the magenta lines.

having to wait for an outcome, by bootstrapping. One of the solutions to the TD control problem is precisely Q-Learning.

B. Q-Learning

A standard reinforcement learning setup consists of an agent interacting with an environment E in discrete timesteps. At each timestep t the agent receives an observation x_t, takes an action a_t and receives a scalar reward r_t. In the environment considered the actions are part of a predefined set of actions. Sometimes, environments may be partially observed, however in the case study, the environment is fully-observed so $s_t = x_t$. In Fig. 4 a visual schematic is presented with the different components used in this method.

An agent's behavior is defined by a policy, π, which maps states to a probability distribution over the actions: $\pi{:}S \rightarrow P(A)$. The environment, E, may also be stochastic. Thus, it is modelled as a Markov Decision Process (MDP), with a state space S, an action space A, an initial distribution $p(s_1)$, transition dynamics $P(s_{t+1}|s_t, a_t)$ and reward function $r(s_t, a_t)$.

The return from a state is defined as the sum of discounted future rewards $G_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r(s_\tau, a_\tau)$ with a discounting factor $\gamma \in [0, 1]$. As it can be seen, the return depends on the actions chosen, and therefore, on the policy π, and may be stochastic. The goal in reinforcement learning is to learn a policy which maximizes the expected return from the start distribution $J = E_\pi[G_t]$.

The action-value function is used in many reinforcement learning algorithms. It describes the expected return after taking an action a_t, in state s_t and therefore follow a policy π:

$$Q_\pi(s_t, a_t) = E_\pi[G_t|s_t, a_t] \tag{1}$$

The Bellman equation, allows a problem redesigning, as a value function recursive definition, which can be stated as:

$$Q_\pi(s_t, a_t) = E_\pi\left[r(s_t, a_t) + \gamma\, E\left[Q_\pi(s_{t+1}, a_{t+1})\right]\right] \tag{2}$$

The expectation depends only on the environment. Thus, it is possible to learn Q_μ off-policy. Q-Learning [11] is an off-policy algorithm that uses the greedy policy $\mu(s) = \arg\max_a Q(s,a)$.

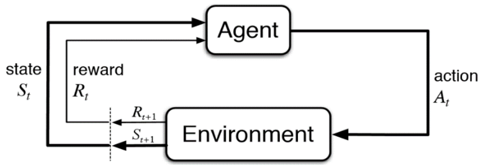

Fig. 4. Reinforcement Learning Components Schematic.

$$\begin{aligned} Q_\mu(s_t, a_t) &= Q(s_t, a_t) \\ &+ \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \end{aligned} \tag{3}$$

C. State-Space

The states are the sensory information resulting from the robot's space observation. Since, as previously stated, the robot must only use its infra-red sensors to detect the obstacles, it exclusively uses both left and right sensors to define the current state. The sensors on Bot'n Roll One A are discrete, resulting in two binary sensor information. Thus, the number of state-spaces in this case study is four. The robot's four possible states are represented in Fig. 5 (left side). The detection volume is conically shaped with a 20 cm range and a 20º angle parameters.

D. Action-Space

An action a is generated in the control script and sent to the simulation environment node, which in turn is applied to the agent. It represents a choice of a pre-determined set of possible actions. The complete pre-determined set of actions consists ofthree forward, three backward, three right turn, three left turn, all of which differently speeded. It has also two movements of self-rotation. Resulting in a total of fourteen possible actions. Fig. 5 (right side) describes all possible movements, where color represents motion intensity.

E. Policy

Q-Learning uses a state-action function, which is updated when a particular action a_t taken at time t, thus when all $Q(s_t, a_t)$ are maximized, the optimal policy is to choose actions with maximum $Q(s_t, a_t)$.

F. Reward

The reward function is denoted by R, which represents the reward obtained for each action a, on timestep t. The reward function is characterized by its following respective function, where i is a discrete variable from one to three and represents the maze level:

$$R(i) = \begin{cases} -1 \text{ ,for time step} \\ -10^3 \delta_u(i-1) \text{,for timeout} \\ -10^3\left(40^{i-1}\right)\text{, for collision} \\ 10^3\left(50^{i-1}\right)\text{, for success} \end{cases} \tag{4}$$

The low negative time step reward forces the agent to try to reach success as fast as possible to maximize the reward.

As for the timeout, δ_u stands for a unitary delta impulse function which can be represented by:

$$\delta_u(t-T) = \begin{cases} 1, & t-T=0 \\ 0, & t-T\neq 0 \end{cases} \tag{5}$$

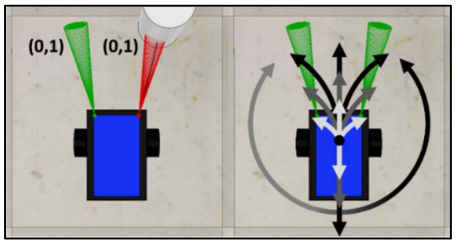

Fig. 5. State-Space (left) and Action-Space (right) visual demonstration with the simulated agent.

This results in a negative reward only on the first level. This reward prevents a robot from getting stuck in a state or set of states performing the same action never leaving the same spot or never being able to reach success or collision. It is only calculated for the first level since it is an essential part of the initial learning and not so for the latter levels. Both the success and collision functions were calculated for an optimized robot's performance. The exponentially increasing characteristic was an important feature for the iterative complexity solving the problem. Higher rewards for higher levels allow the agent to learn how to solve the respective maze coherently.

G. Hyperparameters

As with every Machine Learning implementation, a set of parameters whose value is set before the learning process begins must be tuned. As previously presented, the discount factor $\gamma \in [0, 1]$ represents the weight difference between present rewards and future rewards. It determines the importance of future rewards. A factor of zero will result in the agent only considering current rewards, while a factor closer to one makes the agent strive for long-term high rewards.

The learning rate $\alpha \in [0, 1]$ determines to what extent newly acquired information is more significant than older information. A factor of zero will make the agent learn nothing (only exploit previous knowledge), while a factor of one makes the agent consider only the most recent information. When the problem is stochastic, as in this case study, the algorithm converges under some technical conditions on the learning rate that require it to decrease to zero. Different technical conditions were tested such as a counter for each state-action pair and an iteration counter.

In order to create a source of randomness in the different episodes, random initial orientation is applied to the agent. The robot on each episode will unknowingly be rotated to avoid continuous episode repetition. θ is the absolute value of the max random turn in degrees the agent is placed on.

The number of possible actions is also a hyperparameter. One of the most common problems reinforcement learning implementations must deal with is the curse of dimensionality, as described in [12]. As the number of actions and states increases, the solutions must adapt to higher dimensionalities. Approximation methods such as artificial neural networks are a standard solution to this problem. The number of actions the agent must experiment is proportional to the time-costing and even results in the algorithm present.

Another hyperparameter is the strategy used for solving exploration/exploitation. In this case study, two different strategies are being used. Firstly, the strategy used is ϵ-greedy (epsilon greedy). With ϵ-greedy, the agent selects at each time step a random action with a fixed probability $0 \leq e \leq 1$, instead of selecting greedily one of the learned optimal actions with respect to the Q-Function:

$$\pi(s)=\begin{cases}\text{Random a from A(s), for E<e}\\ \quad\text{arg max}_a \text{ , for E}\geq\text{e}\end{cases} \tag{6}$$

Where $0 \leq E \leq 1$ is a uniform random number drawn at each iteration. Different decreasing ϵ-greedy functions were tested for result comparison. Secondly, Optimistic Initial Values (OIV) were used to provide a secondary source of randomness. OIV consists of initializing all variables with a considerably high value. As the episode runs, the collected data will go down one by one, resulting in an exploration of all variables. Thus, all variables will eventually settle into their true values.

H. Different Implementations

The different Q-Learning implementations were designed for an extensive proposed task comparison and analysis. The different behavior of both implementations will serve the purpose of reasoning the best method. The difference between the two lays on when $Q(s_t, a_t)$ calculated depending on the state transitions and action is executed. The implementations will be named Q-Learning method A and Q-Learning method B for a simpler explanation and an easier understanding.

On the first implementation (Q-Learning method A), the control script reads the sensor information, calculates the respective $Q(s_t, a_t)$ and sends an updated action to the simulator node. This occurs at a fixed Δt, meaning the robot will go through a constant number of iterations per second. For every Δt an action will be calculated and selected via the exploration/exploitation strategy, epsilon-greedy, and sent to the agent. This method allows the agent to experiment with different actions even though there is no state change.

The second implementation (Q-Learning method B) has the same Δt characteristic, meaning it will read the sensor value and calculate $Q(s_t, a_t)$. However, it differs from the previous method since it only chooses a new action when a state change occurs. The agent will keep doing the received action and calculating Q until a state transition occurs. Only when the sensor data differs, a new action is calculated, meaning that an action will go through the epsilon greedy strategy whenever the state differs. In its essence one can interpret this method as using options as introduced in [13].The main difference between the two happens when the agent remains in the same state for a significant amount of time. Method A can alter the action the agent is performing every iteration if so the $Q(s_t, a_t)$ and the greedy policy decide. Also, method B must remain with the chosen action until a state change occurs. Both methods present advantages and disadvantages in comparison to each other, and those are described in the next section (V).

V. Results

In this section, a comparison between both Q-Learning implementation methods is presented. The comparison equates the hyperparameters, the simulation results, the number of simulations necessary for a maze to be considered as learned, the Q values changes, the number of successes, collisions and timeouts. These simulations were performed in real time mode.

Both Q-Learning implementations, (method A and B) were intensively tested. For both the learning rate as well as the ϵ-greedy two distinctive functions were created for an optimized performance to better suit each parameter. For the learning rate, the equation is:

$$\alpha = \frac{\alpha_{ini}}{\text{update\_counts}(s_t, a_t)} \tag{7}$$

The *update_counts(s_t, a_t)* is initially set to one for each *(s_t, a_t)* pair and has an increase of a tenth of a thousand for each time a *(s_t, a_t)* is visited. As for the ϵ-greedy equation:

$$\epsilon = \epsilon_{ini}(0.99^{n\_ep}) \tag{8}$$

Where ϵ_{ini} stands for the initial ϵ-greedy and *n_ep* for the number of episodes the agent has went through. This function exponentially decreases until reaching an absolutely greedy policy. The hyperparameters optimization results from an heuristic based decision process by extensive mathematically substantiated trial-and-error parameter search

For method A, a total of a hundred episodes were performed for each maze level, except for level three. This value turned out to be insufficient for the agent to learn a successful optimal policy. In Fig. 6, a) and b), the results for this method are displayed. The graphics only show the first and second maze levels together, since this method could not successfully solve the second maze. The orange line represents the total reward for each episode, and the blue dashed line represents the moving average reward. As seen, the robot barely solves the first maze, assuming a much more basic strategy than the one displayed in method B (described further in this section). The average reward line demonstrates a sub-optimal policy compared to the one in method B. On TABLE I. hyperparameter transition table is presented, these values were adjusted for optimized learning.

The method A graphics are shown on Fig. 6, Fig. 7 and Fig. 8. In Fig. 6 are presented the total and average rewards for each episode and for the last fifty episodes. This represent the reward evolution, which allows an understanding of the robot's level of successfulness. In Fig. 7 the most prominent changes made on Q on each episode are presented. These show how the highest Q is adjusted for each episode, granting an understanding of how the intelligence system is evolving. Lastly, in Fig. 8, the number of collisions (red line), number of timeouts (blue dotted line) and number of successes (green dashed line) are displayed, granting a visual reference of how many collisions, timeouts and successes the agent faces along the learning process.

The full method B graphics are shown in Fig. 9, Fig. 10 and Fig. 11. These graphics labels and indexes are in all equal to the ones presented on the previous method. From the total and average reward in Fig. 9, to the biggest changes on Q in Fig. 10 , and the number of different occurrences in Fig. 11. For method, the same number of episodes per level were performed for all three mazes. This value revealed to be sufficient for the agent to learn the task at hand successfully.

In Fig. 9, the results for this method are displayed with the same labels and colors as in method A. As presented, the robot successfully learns the three mazes, however, when transitioning to the next level, the robot needs to adapt to the different environment by adjusting its Q values. In Fig. 9 the blue dashed line demonstrates that the agent is successfully learning a strategy in order to solve the maze. On TABLE II. the hyperparameter transition table is presented. These values had to be adjusted for optimized learning. The value evolution of each action is another critical parameter. Tracking this information allows an action-space as well as action transition proper understanding. Fig. 12 presents the different actions by color, and the different action value evolutions are presented throughout 100 episodes on the same state. This excerpt is method's B "Right Obstacle" state from the second level maze. Since the state is "Right Obstacle" a normal agent reaction would be a left turn, however in the situation above presented, not always the agent had that perception. Due to exploration resulting from the exploration/exploitation strategies above described, the agent managed to find a greater solution ("Front Medium" as the black dotted line greater than "Left Strong" as the red dashed line) around episode eighteen.

TABLE I. METHOD A MAZE LEVELS HYPERPARAMETERS

Hyperparameters	*Level 1*	*Level 2*	*Level 3*
α_{ini} (initial learning rate)	0.1	0.01	0.0001
r_c (collision reward)	-1000	-4000	-1600000
r_s (success reward)	+1000	+5000	+2500000
Timeout counter	1000	4000	6000
r_t (timeout reward)	-1000	0	0
ϵ_{ini} (initial ϵ-greedy)	0.05	0.01	0.01

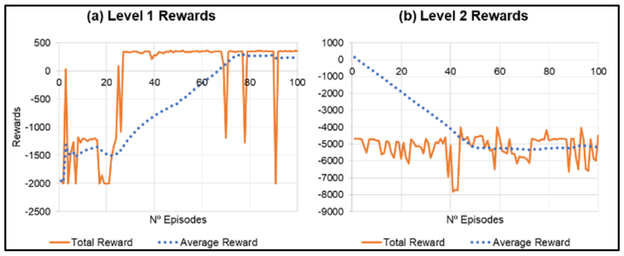

Fig. 6. Method A rewards graphic.

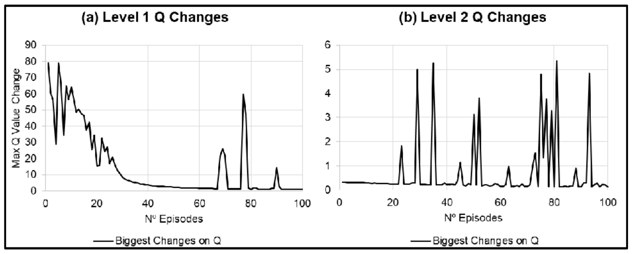

Fig. 7. Method A maximum Q value changes graphic.

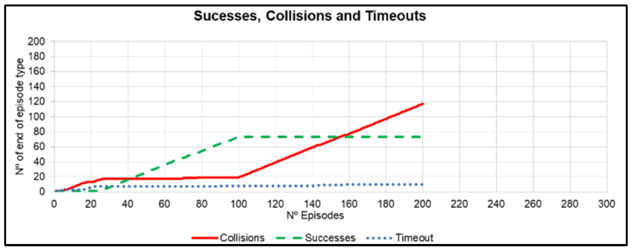

Fig. 8. Method A occurrences graphic.

TABLE II. METHOD B MAZE LEVELS HYPERPARAMETERS

Hyperparameters	*Level 1*	*Level 2*	*Level 3*
α_{ini} (initial learning rate)	0.008	0.001	0.0001
r_c (collision reward)	-1000	-40000	-1600000
r_s (success reward)	+1000	+50000	+2500000
Timeout counter	1000	4000	6000
r_t (timeout reward)	-1000	0	0
ϵ_{ini} (initial ϵ-greedy)	0.05	0.01	0.01

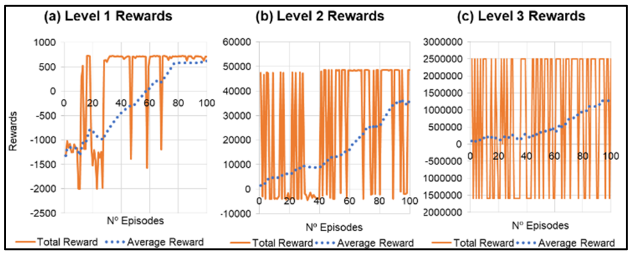

Fig. 9. Method B rewards graphic.

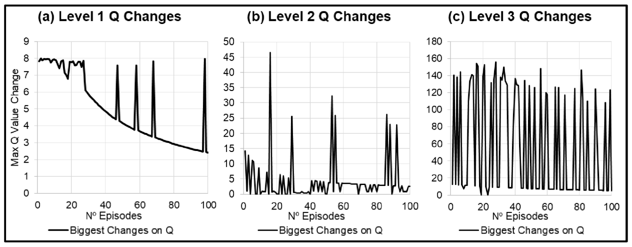

Fig. 10. Method B maximum Q value changes graphic.

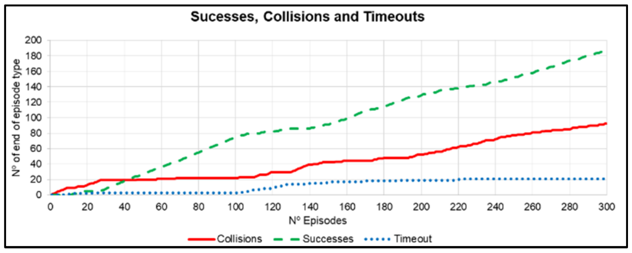

Fig. 11. Method B occurrences graphic.

This solution caused the Q value to alter its maximum action for this state. With time, the agent realized that said action which once was a great decision, was probably a lucky action, and is starting to decrease its trust on "Front Medium", eventually falling to a sub-optimal action. However due to another exploration, that action managed to surpass the now optimal action, which then slowly again eventually fades, and the agent manages to continuously increase its trust on "Left Strong" action. This trust continues to rise until the end of the training.

Fig. 13 presents a visual representation of the navigation evolution for method B. The colors: red, orange, yellow and green correspond to the episode's number for each maze as described in the figure's label. These navigation details showcase the robot evolution throughout the reinforcement learning process. As demonstrated, the robot manages to adjust its parameters for each maze to increase its success rate as well as the amount of maze completed. For the first maze, the robot manages to learn two different successful policies to reach the

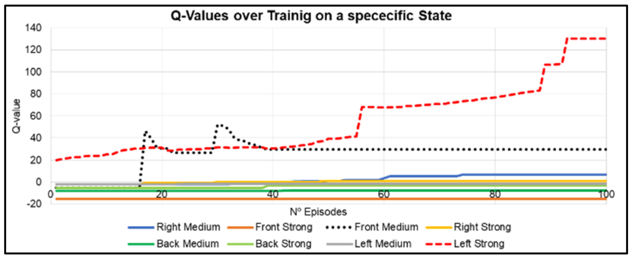

Fig. 12. Action Evolution on state "Right Obstacle" over one hundred episodes.

end point. As for the second and third maze the robot manages to successfully improve its performance by making minor adjustments to Q. On the overall, the robot manages to either go further in the maze or find new policies to solve the navigation and obstacle avoidance problem.

VI. DISCUSSION

Method A revealed to be a slow, safe and ineffective learning method for the case study even though the hundreds of different hyperparameters tested. As seen in Fig. 6, Fig. 7 and Fig. 8, it managed to solve the first maze with some long times. However, it never managed to solve the second level maze. Even with an extensive parameter search it was not possible to increase this method's learning quality. The rewards graphic in Fig. 6 demonstrates that the agent tends to stagnate its learning. The policy derived by the agent on the first level can solve the maze, but when put to the test on the second level, it tried to apply the same policy, yet, not very effectively. This is due to the considerably larger turns on level two maze, consistently colliding in turns or getting lost in the maze. In conclusion, this method was able to learn a sub-optimal policy that allows the agent to reach its goal, on the first level yet slower than method B. Its policy mainly consists of continuously turning when it sees no obstacles, in search of an obstacle so that the robot can toggle between moving against the object and moving away from the object. Since it chose two actions that when summed have a positive value for each motor, the agent manages to go through the first level maze very slowly and yet in a safer way. This turned out to be an impossible policy for the second level. This can be observed in Fig. 8, since the robot has a very high number of collisions and an almost null number of timeouts.

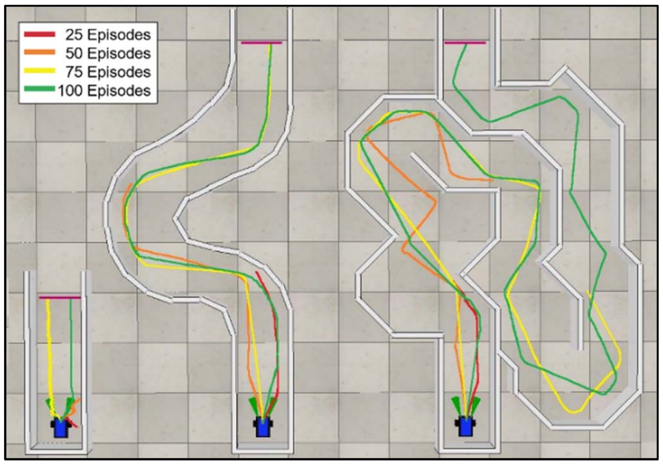

Fig. 13. Robot's navigation evolution for the three mazes.

Method B revealed to be a quick learning method. As seen in Fig. 9, Fig. 10 and Fig. 11, it managed to solve the three mazes on a fewer episodes than method A. The most significant changes on Q graphics in Fig. 10 reveal that the agent had a smooth learning on level one, on level two a more unstable learning, and on level three a completely unstable learning. However, to solve the three mazes, the reward equations had to grow exponentially, which demonstrates the good learning behavior the robot had. The reward graphics in Fig. 9 reveal that the agent with the course of the episodes managed to consistently increase the average reward, which results in successful learning through all episodes. The occurrences graphic in Fig. 11 shows a slight decrease on the slope on successes every time a level change occurs, resulting in a slope increase on the other two lines. In conclusion, this method was able to learn to avoid the walls from both sides and to go forward as fast as possible when there are no obstacles in sight.

In Fig. 12 the agent manages to explore and exploit efficiently. Initially, it starts by exploiting the "Left Strong" (red dashed line) action, and then exploring the "Front Medium" (black dotted line). This exploration revealed as a positive try, resulting in a new optimal action. By exploiting this new optimal action, the agent concluded that it was maybe a lucky shot, returning the "Left Strong" action being the optimal one. This happens once again, and even then, the agent is capable of overcoming this sub-optimal action.

The robot's navigation evolution, as demonstrated in Fig. 13 displays the different policies used along the training in the three mazes. For the first maze the robot starts out by experimenting different random policies. The navigation on episode 75 (yellow) showcases a non-optimal policy the robot figures out. This ends up solving the problem by turning around until it sees a wall and then toggling between left and right turns until it reaches the end point. However, with some more episodes it manages to figure out a new policy which increases its rewards, reaching the end point quicker. For the next two mazes the learning process consists on making minor adjustments to the policy so it can increasingly reach a further point in the maze.

Both methods can solve the first maze, method A with a slower and safer strategy, and method B with a quicker yet safe approach. Each agent managed to develop a different solution to the same problem; this mainly resides on the differences between methods and its exploration/exploitation strategies. Method A uses the Optimistic Initial Values much more significantly than ϵ-greedy, and method B the other way around. A different reward function structure could be implemented as a way to adapt to the method A strategy. Method B revealed to be a very promising solution to the problem at hand managing to solve all three mazes on a reduced number of iterations. Different solutions originated from different parameters, methods and strategies demonstrate the consistency and power of Q-Learning algorithms to solve obstacle avoidance problems.

VII. Conclusions and Future Work

Both methods were able to solve the first maze, method B more successful in terms of learning quickness and robustness. Method B was able to solve the first maze more efficiently than method A. The efficiency can be measured, reward, time, and speed wise. Method B also managed to solve the other three mazes with a reduced number of episodes. This implementation has a more reliable reward average meaning it had more efficient learning. By comparing hyperparameters, it is possible to see that method A needs higher values compared to method B, and yet it did not have the same performance efficiency.

Though both methods work, the state-space did not allow a higher complexity understanding of the environment. Some solutions are increase the number of states, either by increasing the number of sensors (which would stop physically matching the case study) or changing to a continuous sensor analysis (which would also not meet robot's physical characteristics).

For this case, method A is a considerably worse strategy than method B, since the state transition characteristics do not benefit method A. Method A would be a superior choice for continuous state-space and action-space, due to prompt response to different states and actions. Having a fixed number of actions is a great solution to simplify the curse of dimensionality problem and yet, it goes against the robot's philosophy, its analogic motors offer continuous action-space for future research.

The proposed obstacle avoidance method is shown to work well avoiding obstacles in simple mazes as the ones presented. However, the future of this project aims to adapt the implemented algorithms to a fully both state-space and action-space continuous solution, taking full advantages from the analog sensor and actuators present in modern robotics.

VIII. References

[1] M. G. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics and Autonomous Systems*. 2018.

[2] G. Brockman *et al.*, "OpenAI Gym." 2016.

[3] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric Actor Critic for Image-Based Robot Learning." 2017.

[4] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based Deep Reinforcement Learning for Obstacle Avoidance in UAV with Limited Environment Knowledge," 2018.

[5] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[6] B.-Q. Huang, G.-Y. Cao, and M. Guo, "Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance," *2005 Int. Conf. Mach. Learn. Cybern.*, 2005.

[7] T. Ribeiro, I. Garcia, D. Pereira, J. Ribeiro, G. Lopes, and A. F. Ribeiro, "Development of a prototype robot for transportation within industrial environments," *2017 IEEE Int. Conf. Auton. Robot Syst. Compet. ICARSC 2017*, pp. 192–197, 2017.

[8] A. F. Ribeiro, G. Lopes, N. Pereira, and J. Cruz, "Learning robotics for youngsters-the roboparty experience," *Adv. Intell. Syst. Comput.*, vol. 417, pp. 729–741, 2016.

[9] A. F. Ribeiro, G. Lopes, N. Pereira, J. Cruz, and M. F. M. Costa, "Bot'n roll robotic kit as a learning tool for youngsters," *9th Int. Conf. Hands Sci.*, no. 266647, pp. 192–192, 2012.

[10] E. Rohmer, S. P. N. Singh, and M. Freese, "v-repIros2013," pp. 0–5.

[11] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[12] T. P. Lillicrap *et al.*, "CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING." 2016.

[13] R. S. Sutton and S. Singh, "Summary for Policymakers," *Clim. Chang. 2013 - Phys. Sci. Basis*, vol. 1, no. 1999, pp. 1–30, 2015.