Mobile Robot Navigation in Dynamic Environments Using Reinforcement Learning

Fauzi Abdul Rohim
Electrical Engineering Department
Telkom University
Bandung, Indonesia
fauziabdulrohim@student.telkomuniversity.ac.id

Syamsul Rizal
Electrical Engineering Department
Telkom University
Bandung, Indonesia
syamsul@telkomuniversity.ac.id

Sony Sumaryo
Electrical Engineering Department
Telkom University
Bandung, Indonesia
sonysumaryo@telkomuniversity.ac.id

Eki Ahmad Zaki Hamidi
Electrical Engineering Department
UIN Sunan Gunung Djati Bandung
Bandung, Indonesia
ekiahmadzaki@uinsgd.ac.id

2022 8th International Conference on Wireless and Telematics (ICWT) | 978-1-6654-5122-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICWT55831.2022.9935359

***Abstract*—The navigation system is one of the most important and crucial concerns in the research of mobile robots. Perception, cognition, action, human-robot interaction, and control systems are among the difficulties that have been resolved. Each navigation system must handle the aforementioned common designs to ensure that all duties may be completed. The navigation system is built on learning techniques that provide the ability to reason in the face of environmental uncertainty. However, the design will be difficult to build due to a number of factors, including inherent uncertainties in the unorganized environment. A more expensive design cost, computational resources, and larger memory are all required in this case. Navigating an autonomous robot in an uncontrolled environment is difficult because it necessitates the cooperation of a number of subsystems. Mobile robots must be intelligent in order to adapt to navigation in unfamiliar environments, such as environmental cognition, behavioral decisions, and learning. The robot will then navigate around these obstacles without collapsing and arrive at a specific destination point. Combining two processes, such as environmental mapping and robot behaviors, can result in behavior-based navigation. Obstacle avoidance, wall following, corridor following, and target seeking are some examples. If only one of the two processes is used, the system should be used in two ways. When this approach is used, two major issues are bound to arise: I the combination of two simple behaviors to form a complex one, and (ii) the integration of more than two behaviors. Behavior induced by multiple concurrent goals can be smoothly blended into a dynamic sequence of control action. This study is concerned with the automatic navigation of a mobile robot from its starting point to its destination point. To solve a few sub-problems associated with automatic navigation in an uncontrolled environment. Monte Carlo simulation is used to evaluate the algorithm's performance and show under what conditions the algorithm performs better and worse. Obtaining position mapping to optimize action on mobile robots using a reinforcement learning framework. Reinforcement learning necessitates a large number of training samples, making it difficult to apply directly to real-world mobile robot navigation scenarios. To address this issue, the robot is trained in a Gazebo platform middleware Robot Operating System (ROS) simulation environment, followed by Q-Learning training on mobile robots.**

Keywords—navigation, mobile robot, dynamic environment

I. INTRODUCTION

Data from robot sensors can be mapped and used by robots for navigation and movement planning. In addition, data from sensors is also used to estimate the position of the robot needed when mapping the surrounding environment. All modules that represent one behavior work together [1]. The main attention is paid to two approaches to coordination mechanisms, namely competitive (arbiter) and cooperative (command) fusion. Meanwhile, cooperative coordination combines all existing behavior outputs and determines the performance of the robot‘s trajectory. Its main feature is the hybrid coordination of behavior, between competitive and cooperative approaches.

A mobile robot must have high navigation abilities before it can perform other jobs such as carrying items or performing mapping activities. This allows the robot to go from one location to another without colliding with obstacles. When dealing with areas containing static impediments, such as industrial locations like warehouses, current technology allows mobile robots to work very successfully. However, there are areas for further development, such as creating a navigation system capable of dealing with complex environments, such as those encountered by humans. [7].

It's not easy to create algorithms and program robots that can move in a static or dynamic environment. Many related research fields have been conducted, including the topic of navigation systems and several fragments of topics from related research fields, such as designing robotic paths with the A* algorithm, mapping and localization using mobile robots, and real-time collision avoidance on mobile robots using deep reinforcement. On the Turtlebot robot platform, learning, autonomous navigation using reinforcement learning, and transferring learning from simulation to robot automobile. The navigation system's use of non-learning algorithms allows the mobile robot to plan and follow a path to its destination, but the algorithm isn't good enough to provide local planning skills in an unfamiliar environment, where the robot is still stuck in the implementation area. minima in the immediate vicinity While the method avoids barriers that did not exist in the prior map, it also necessitates strong parameter assumptions and is extremely difficult to set manually.One of the most powerful methods for solving navigating issues in complicated and unknown surroundings is machine learning. Unfortunately, earlier research only tested at the simulation level, and only a handful were evaluated in real navigation settings, or only on robots that were particularly supported by ROS.

978-1-6654-5122-2/22/$31.00 ©2022 IEEE

One of the most suitable and widely used RL learning methods for autonomous robot applications is RL with the Q-Learning algorithm type. This algorithm uses a Q table to match discrete states and actions only. Meanwhile, in autonomous robots, state size and sensor data are continuous, so this becomes impractical. Therefore, to extend the QLearning algorithm related to continuous state and action, L. Jouffe in [2] combines Q-Learning. Reinforcement learning is the optimal control method, when the agent starts from an ineffective solution which gradually increases according to the knowledge gained to solve successively. decision problems [3]. To use reinforcement study, several approaches are possible. The first consists of manually discrete issues to obtain state and action space; which can be used directly by the algorithm using table Q [3].

However, it is necessary to pay attention to discretization options, so as to allow true learning by providing situations and actions that contain understandable rewards. The second method consists of working on a continuous state and action space using a value function [4]. Indeed, to use reinforcement learning, it is necessary to correctly estimate the value function. The results obtained show a substantial improvement of the robot's behavior and learning speed.

The goal of this research is to show how the Q-learning algorithm, which is a type of reinforcement learning, may be combined with the ROS stack navigation system to give robotic automatic navigation skills. Q-Learning is a representation of a machine learning algorithm based on a driven learning process, in which an agent learns and discovers via experience the pattern of action that will provide him with the best longterm reward. All of these navigation systems are made with ROS, which is modular and multithreaded, making it simple to create navigation functions and allowing numerous jobs to be accomplished at once.

II. METHOD

A. Behavior-based Robot (BBR)

The robot will be controlled directly by a low level controller (in the form of a P controller) which functions to control the movement of the robot by sending perceptual signals (through its sensors) to the high level controller. Then the signal becomes a stimulus input that will activate certain behaviors (which are coordinated by the behavior coordination section). The output of the behavior coordination is given to the low level controller for controlling the robot. The block diagram of the whole robot is shown in the following picture (see Figure 1).

B. Reinforcement Learning

Reinforcement learning is a kind of middle ground between supervised learning and unsu- pervised learning, where the algorithm is in a way told only how well it works, not knowing what the exact output of the system should be. Thus, an incentive learning system, pop- ularly called an agent, must explore and test different solutions and actions to find out how to come up with the right answers. The only information.

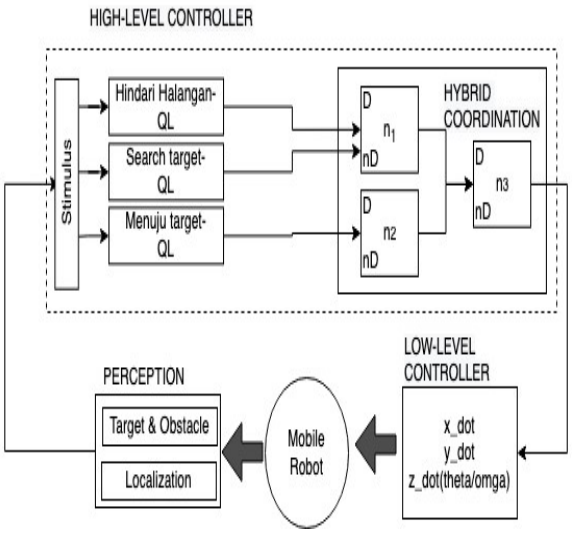

Fig. 1. Behavior based block diagram on mobile robot

The agent receives from the training algorithm is a reward if the action he has taken is favorable, or a penalty if it is not. A standardized schematic representation of an incentive learning system is shown in the following (Figure 2).

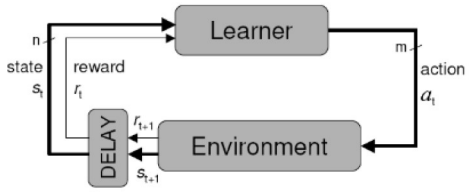

Fig. 2. Interaction diagram between learner and environment

This type of training is most often encountered in sequential decision-making and man- agement problems, where it is impossible to provide explicit supervision to the training algorithm. The branches in which stimulated learning is most often applied are robotics, game theory, autonomous driving, finance, speech processing, recognition, and many oth- ers. Since one of the algorithms of learning with encouragement was used to make this thesis, most of this chapter will be dedicated to this type of learning, while learning with supervision and learning without supervision will remain only briefly described. In order to understand the setting of the issue of learning with encouragement, it is first necessary to introduce the concept of Markov's decision-making process (MDP). (Figure 2).

1)*Q-Learning:* Q-Learning is competitive learning where there are agents with the ability to learn to act optimally by evaluating the consequences of their actions. States that Q-Learning is a Q function that tries to estimate the next discounted reinforcement signal to take actions from the given states. Q-learning is one of the free-learning models in RL Engineering. Use of Q-learning to find the optimal value of Q-value (action value function). During the learning process the Q value will continue to be updated, from the old Q value to the new Q value. Any change in the value of Q depends on the selection of an action on the service.

Q-learning has a way of working by evaluating each episode, the process in one episode is said to end if the agent has reached the goal state point, and every action will affect the Q value. The Q value is used as a ”brain” by the agent during the learning process. For more details, here is a picture of the Q-Learning algorithm (Figure 3):

Algorithm 1: Q-Learning

Result: Off-policy control for estimating $\pi \simeq \pi_*$
Parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialize $Q(s,a)\ \forall s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ arbitrarily.
Set $Q(terminal,\cdot) = 0$
for *each episode* **do**
  Initialize S;
  for $step = 0$ **to** T **do**
    Choose A from S using policy derived from Q (e.g., ϵ-greedy);
    Take action A, observe R, S';
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$;
    $S \leftarrow S'$;
  end
end

Fig. 3. Q Learning Algorithm on Robot

C. Design of Q-Learning Algorithm

In order to improve the control system based on the closed loop controller, the artificial intelligence algorithm described, the Q learning algorithm, will also be designed. In this way, the robot, with the help of LIDAR sensors, will have information about obstacles in its path and will learn to avoid them. The first step in the process of designing a Q-learning algorithm is to discrete the state and action space and define the reward function that the agent will receive for the action taken. Only when these things are defined is it possible to start the training process, which will eventually lead to the convergence of the algorithms and the realization of the desired agent performance.

1)Discretization of state space and action: The agent state space is based on measurements from the laser distance sensor. Since the laser sensor used measures distances in the range 12cm to 3.5m, 356^0 around the robot, it is clear that this will result in a very large state space that needs to be discretized. in some way so that the algorithm can converge.

2)First, the distance measurement is limited to a maximum of 1m and it is chosen that, the LIDAR measurement is in the range $[-75^0,75^0]$ in relation to the movement direction of the robot which is included in the state discretization process, which is a valid assumption because obstacles are greater than 1m distance from or behind robots do not pose a hazard and should not be considered.

State space consists of 4 state variables (x_1,x_2,x_3,x_4) which are determined based on the obstacle distance from the robot and its position. The variables x_1 and x_2 are determined based on the distance of the closest obstacle to the robot. The distance marker is denoted by d, the state variable declares x_1 and x_2 is defined as follows:

$$x_i = \begin{cases} 0, & 12cm \leq d \leq 40cm \\ 1, & 40cm \leq d \leq 70cm \\ 2, & 70cm \leq d \leq 100cm \end{cases} \quad i = 1,2 \tag{1}$$

The distance d is calculated separately for the left and right sides of the robot, where x_1 corresponds to the left and x_2 to the right. An illustration of the state of the sub-space variables x_1, and x_2.

The two remaining state variables x_3 and x_4 are determined based on the position of the obstacle relative to the robot. Assume that the obstacle to the robot is denoted by p, and represents the angle at which the obstacle is detected by the

LIDAR, then let the segments $s_1 : 0^\circ \leq p < \frac{h}{3}$ and s_2 : $dfrach3 \leq p < \frac{2h}{3}$, where h indicates the width of the LIDAR range, which is 75°. Then the calculation of the variables of the state x_3 and x_4, as follows:

$$x_i = \begin{cases} 0, & obstacledetectedins_1 \\ 1, & obstacledetectedins_2 \\ 2, & obstaclescovertheentirefrontarea \\ 3, & obstaclesoutofreach \end{cases} \quad i = 3,4 \tag{2}$$

The position p is calculated separately for the left and especially for the right side of the robot, where x_3 corresponds to the left and x_4 to the right. It should be noted that the values of state variables are assigned with decreasing priority, which means that if the condition for x_i = 0 is met, it will be higher priority than the condition for x_i = 1, which in that case will not be checked. This makes sense because the highest priority will be the obstacles that are directly in front of the robot. An illustration of this division of the state subspace of variables x_3 and x_4.The next step is to discrete the action space that the agent can take. To make the algorithm as simple as possible, we define 3 actions that the agent can take: move forward, turn left, and turn right. Each of these actions is determined by the linear and angular velocity of the robots v_x and ω_z, as follows:

$$v_x = \begin{cases} 0.9m/s, & forward \\ 0.6m/s, & turnleft \\ 0.6m/s, & turnright \end{cases} \tag{3}$$

$$\omega_z = \begin{cases} 0rad/s, & Forward \\ 0.7rad/s, & Turnleft \\ 0.7rad/s, & turnright \end{cases} \tag{4}$$

2) Defining the Reward Function: The next step in designing the algorithm is to define the reward function needed to determine the Q-value and fill in the given Q-table. The reward function is defined as a combination of three different reward functions:

$$r_1 = \begin{cases} +0.2, & a_t = straight \\ -0.2, & a_t = right/left \end{cases} \tag{5}$$

The reward function r_1 is defined to provide a positive reward if the agent moves in a straight line, and a small negative reward if he turns. The purpose of a small negative reward when the agent turns is to give priority to moving forward so that the agent's movement is directed to the desired position.

$$r_2 = \begin{cases} +0.2, & W_{r2} \Delta d < 0 \\ -0.2, & W_{r2} \Delta d \geq 0 \end{cases} \quad (6)$$

The reward r2 is positive if the weighted cumulative distance from the obstacle decreases. The cumulative distance is denoted by is the robot's distance from the obstacle at time t and dt-1 is the robot's distance from the obstacle at the time before t . The weight vector W is defined so that it has the highest value r2 at a point in the robot direction and decreases linearly as the LIDAR measurement angle increases symmetrically on both sides. The notation W *deltad* represents the scalar product *eWidi* which returns a single number. r1 cannot be greater than the negative reward on the function r2, so the agent has no tendency to move towards the hitch. Also, the positive reward r2 must outweigh the negative payoff of r1 to give the agent priority to avoid hindrances over other actions.

$$r_3 = \begin{cases} -0.8, a_t = left\,right \cap a_{t-1} = right\,left \\ 0, otherwise \end{cases} \quad (7)$$

The reward function r3 aims to prevent sudden changes in the turning direction of the agent, that is, to make his movements as smooth as possible. To assign this function as the highest priority, the value of the negative reward must be greater than the positive reward of the r2 function.

The total reward rt will be equal to the sum of the values of the previous three reward functions if there is no collision, while the very large negative reward is -100 if there is a collision. Collision is defined as:

$$min\{d_w\} < d_{collision}, d_w^i = w_t^i d_t^i \quad (8)$$

Where Wt is the weight vector which has the smallest value at a point in the direction of the robot's movement and increases linearly as the LIDAR measurement angle increases symmetrically on both sides, and dt is the distance vector representing the LIDAR measurement. Such an arrangement gives higher priority to obstacles directly in front of the robot, while the distance from adjacent obstacles tends to increase to reduce their impact. The distance of 14 cm is selected for the limit value *dcollision.*

3) Policy Determination: To select the best action, the QLearning algorithm requires a search strategy to determine the actions taken by the agent as a function of the agent's state and environment, which is called Policy (π)). The two strategic models that will be used in this study will provide the tradeoff compromise that has been mentioned in the explanation in Chapter II, namely the greedy - search and the algorithm based on the Boltzmann distribution, which is popularly called softmax. Greedy - search is based on selecting the best action using probability 1 -, whereas with probability a random action is chosen, for example:

$$a = \begin{cases} a^*, & with probability 1 - \epsilon \\ a_r, & with probability \epsilon \end{cases} \quad (9)$$

The symbol a^* represents the optimal action, while represents random action, and the eps parameter is selected in the range 0 to 1. Changing the eps parameter will change how the training process takes place, if a random value of r is less than the exploration rating (in the case of this is 1 ϵ)the agent will choose the action with the highest Q-value in the Q-table, while if the random value of r is greater than or equal to the exploration rating (), the agent will choose the action at random. In this way, the agent in the early stages of training will choose many actions at random, which is called the exploration phase. Then over time and as the agent's knowledge increases, the agent will adopt greedy behavior in the exploitation phase, i.e. the agent will only take action with the highest Q-value in the Q-table [5]. The problem with the greedy - search strategy is that all actions will be randomly selected with a uniform probability distribution at the start, meaning that the probability of finding a good action and a bad action is the same, so the search strategy for the best action is not optimal due to the high value action. .To overcome this, a Boltzmann distribution strategy such as softmax is used.

Softmax's search strategy is based on selecting an action based on probability considering the Q-value of the actionstate pair. The probability of selecting an action is proportional to , which means an agent in the state s will choose an action a with a large probability [6]:

$$P_a(s,T) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_a e^{\frac{Q(s,a)}{T}}} \quad (10)$$

The T parameter determines how random the action will be. If the value of T is large, the set of actions will be randomly selected with proportional probability. whereas at lower values T the action set is selected with a higher value of Q. If T is zero, the action with the highest Q value will continue to be selected. In the early stages of training the value of T is chosen higher, which will eventually drop to zero as the agent gains knowledge [6].

TABLE I. MAPPING OF ACTION SPACE AND STATE ON Q-TABLE

Q-table	$a_1 = f(v,\omega)$	$a_2 = f(v,\omega)$	$a_3 = f(v,\omega)$
$S_1 = f(x_1,x_2,x_3,x_4)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	$Q(s_1,a_3)$
$S_2 = f(x_1,x_2,x_3,x_4)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	$Q(s_2,a_3)$
$S_3 = f(x_1,x_2,x_3,x_4)$	$Q(s_3,a_1)$	$Q(s_3,a_2)$	$Q(s_3,a_3)$
...	...	...	...
$S_{143} = f(x_1,x_2,x_3,x_4)$	$Q(s_{143},a_1)$	$Q(s_{143},a_2)$	$Q(s_{143},a_3)$
$S_{144} = f(x_1,x_2,x_3,x_4)$	$Q(s_{144},a_1)$	$Q(s_{144},a_2)$	$Q(s_{144},a_3)$

III. RESULT AND DISCUSSION

A. Algorithm of Q-Learning

Define abbreviations and acronyms the first time they are uFor the learning algorithm parameter Q itself, the values α=0.3 and γ= 0.9 were chosen to ensure slower agent training, as well as to account for the long-term effects of the current action. The training process is carried out in 200 epochs, where the maximum number of agent actions during

one epoch is 300. The simulation results that aim to compare search strategies are shown in the following graph:

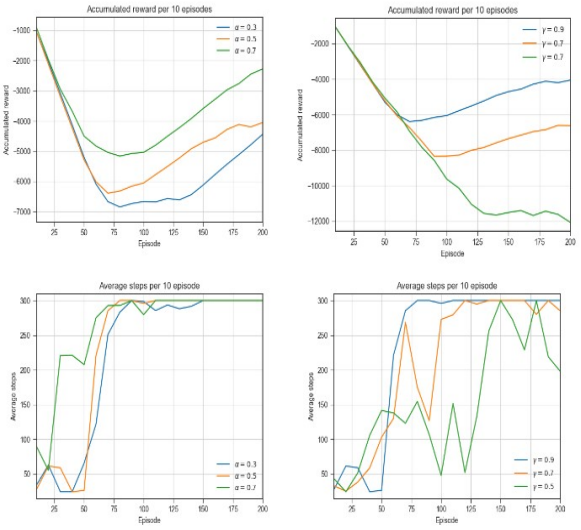

Fig. 4. Average number of agent steps through epochs for a parameter ,Accumulated agent reward through epochs for a parameter, Accumulated agent reward through epochs for a parameter,Average number of agent steps through epochs for a parameter

The accumulated rewards and the average number of agents steps are averaged over 10 epochs to create the smoothest possible graph, emphasizing their trend in relation to the value at each less important point. Based on the attached graph, it can be clearly concluded that the search for softmax is a better choice in certain problems because it successfully converges towards the maximum number of steps per epoch, which means that the agent successfully learns to avoid obstacles. This is precisely because of the theoretical basis of the strategies used, which talks about the probability distribution of choosing stocks. The same conclusion can be drawn by looking at the graph of the accumulation of rewards through the epoch, which clearly shows that the agent using softmax search already in the first 100 epochs recognized the adequate pattern of action that would give him the greatest reward, which is not the case with greedy search. Softmax search defined in this way will be used in the following simulations as an adequate search strategy. The following simulation is dedicated to determining the optimal value of the training parameters.

From the attached graph, it can be clearly seen the impact of the parameters on the training process, where for higher parameter values, updating the values in the Q table is faster, i.e. the impact of the knowledge gained is increased compared to the existing ones. This conclusion confirms the influence of parameter to update the Q - value.

Although higher parameter values provide faster algorithm convergence, too large values can result in the instability of the training process. Since the final training process will involve a larger number of epochs and a greater number of steps per epoch, the optimal value of the parameters will be chosen = 0.5 to ensure safer algorithm convergence in the long run. The last parameter discussed is the parameter . Again, 3 simulations with different values were performed, in which other parameters and hyperparameters of the algorithm were adjusted based on the previous conclusions. The simulation results are shown in the following graph:

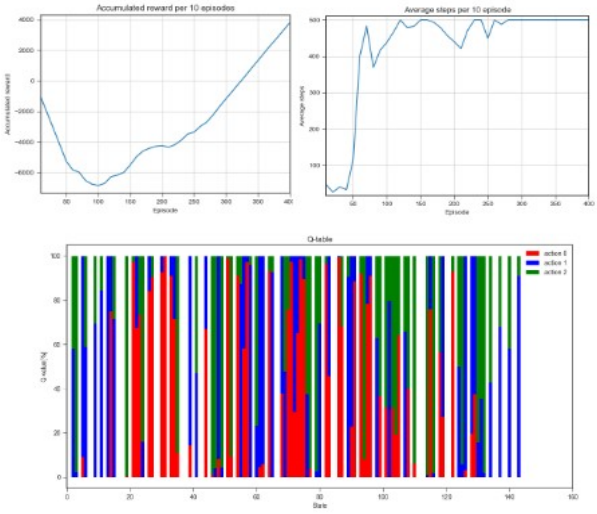

Fig. 5. Accumulated agent reward through epochs, final training, Average number of agent steps through epochs, final training, Q-table at the end of the training algorithm

From the attached graph it can be concluded that only a higher parameter value will ensure the convergence of the algorithm which means that the agent must take into account the long-term consequences of his current actions, not relying solely on the current rewards they carry. Influence parameters in the described way can also be seen from the formula for updating the Q - table . Based on the experiments conducted, it is concluded that the optimal parameter value is 0.9, so it will be used in the final agent training.

It should be noted that the training parameters are not very independent and cannot be observed completely separately. However, the simulations carried out show that some conclusions can still be drawn and optimal values can be determined. Regarding the presented simulation results, it should be noted that some simulations have to be run more than once, due to the stochastic nature of the algorithm, convergence is uncertain. Finally, the final process of training agents in a simulated environment is carried out with the parameters and hyperparameters selected based on the previous analysis, but this time the training is extended to a larger number of epochs and a greater number of steps per epoch. An overview of the parameters and hyperparameters of the final training process is given in the following table:

TABLE II. PARAMETERS AND HYPERPARAMETERS OF THE FINAL TRAINING PROCESS

search strategy	softmax
T_0	25
∇_T	0.95
α	0.5
γ	0.9

The results of the final training process are shown in the following graphs:

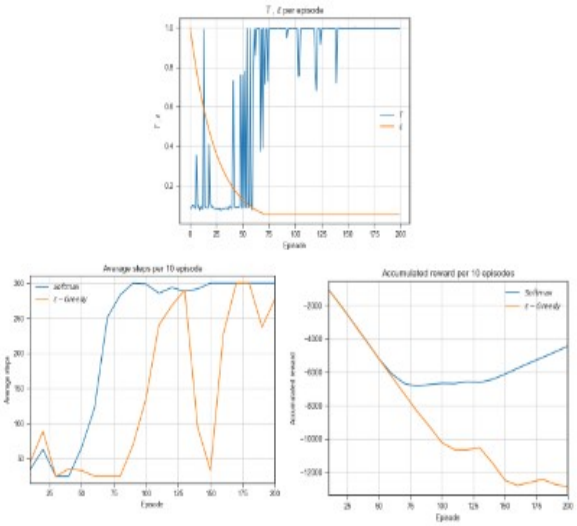

Fig. 6. Parameters of epoch search strategies, Accumulated agent reward through epochs for different search strategies, The average number of agent steps through epochs for different search strategies.

Based on the attached graph, it can be concluded that the training process was successful and convergence was achieved. It should be noted that several successive simulations are required to achieve convergence, since the training process itself is stochastic and convergence is not guaranteed, which was discussed earlier. The end product of the training process is actually a Q-table on which the agent will make a decision on which action to take under appropriate circumstances. The Q-table results obtained from the training process are illustrated in the following graph:

From the attached Q - table it can be seen that the agent has clearly distinguished which action represents the optimal solution under certain circumstances, because in most of the circumstances, one dominant Q - value can be observed which corresponds to the best course of action. It should also be noted that certain conditions remain unexplored. However, this is not dangerous because it is a harmless situation for the robot or a very dangerous situation but the robot has learned to avoid it through the training process. This completes the process of designing and training the Q learning algorithm and it remains only to implement and test it in combination with the closedloop control algorithm, which will be the subject of the next chapter.

IV. CONCLUSION

This paper discusses the application of artificial intelligence and machine learning algorithms in the field of mobile robotics and autonomous driving. One of the most popular incentive learning algorithms has been implemented, the Q-learning algorithm, which is based on agent learning from experience. This thesis first projects a conventional control algorithm based on a closed loop controller which ensures that the robot reaches a certain position and orientation. The controller is implemented and tested in a simulated environment, where its parameters are adjusted. The big disadvantage of this algorithm is the inability to avoid obstacles that the robot encounters on its way to its destination. To solve this problem, the Q-learning algorithm has been designed to provide this functionality. The agent training process was conducted in a simulation environment, different values of algorithm parameters as well as different search functions were considered. When the optimal values of the parameters and the optimal search strategy were found, the final training of the agent was performed. After several simulations, the training resulted in convergence and the final Q-table that will be used in the implementation of the algorithm.

REFERENCES

[1] B. K. Oleiwi and H. Roth, “Application of Fuzzy Logic for Collision Avoidance of Mobile Robots in Dynamic-Indoor Environments E 1 oooooo /,” pp. 131–136, 2021.

[2] L. Jouffe, ”Fuzzy inference system learning by reinforcement methods,” in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 28, no. 3, pp. 338-355, Aug. 1998, doi: 10.1109/5326.704563.

[3] J. Bai, S. Lian, Z. Liu, K. Wang, and D. Liu, “Deep Learning Based Robot for Automatically Picking Up Garbage on the Grass,” IEEE Transactions on Consumer Electronics, vol. 64, no. 3, pp. 382–389, 2018.

[4] Houxiang Zhang, Jianwei Zhang, Guanghua Zong, Wei Wang, and Rong Liu, “Sky Cleaner 3: a real pneumatic climbing robot for glass-wall cleaning,” IEEE Robotics & Automation Magazine, vol. 13, no. 1, pp. 32–41, 2006.

[5] Efroni, Yonathan, Gal Dalal, Bruno Scherrer, and Shie Mannor. ”Beyond the one step greedy approach in reinforcement learning.” arXiv preprint arXiv:1802.03654 (2018).

[6] Tijsma, Arryon D., Madalina M. Drugan, and Marco A. Wiering. ”Comparing exploration strategies for q-learning in random stochastic mazes.” In 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1-8. IEEE, 2016.

[7] Dobrevski, Matej, and Danijel Skocaj. ”Map-less goal-driven navigation based on reinforcement learning.” 23rd Computer Vision Winter Workshop. 2018.