

Autonomous Robot Navigation Using Deep Reinforcement Learning for Obstacle Avoidance and Path Optimization

E. Afreen Banu^{1}, Raziya Sultana Sharief², Jayaram Boga³, Ganesh H S⁴, A. Vijaya Mahendra Varman⁵, and Shailendra Kumar Bohidar⁶*

¹Department of Computer Engineering, Shah & Anchor Kutchhi Engineering College, Mumbai, Maharashtra 400088, India

²Department of CSE-AIML, St. Ann's College of Engineering & Technology, Chirala, Andhra Pradesh 523187, India

³Department of Computer Science Engineering, Koneru Lakshmaiah Education Foundation, Hyderabad, Telangana 500043, India

⁴Department of Electronics and Communication Engineering, Vidyavardhaka College of Engineering, Mysuru, Karnataka 570002, India

⁵Department of Artificial Intelligence and Data Science, Panimalar Engineering College, Chennai, Tamil Nadu 600123, India

⁶Department of Mechanical Engineering, School of Engineering & I.T., MATS University, Raipur, Chhattisgarh 493441, India

Abstract. This paper compares the performance of a traditional approach for autonomous robot navigation and a new approach for the same problem, all within the framework of reinforcement learning. A new approach for the problem has been introduced and compared against the traditional approach, referred to as the value iteration approach for holonomic robots and the new approach for the same problem in relation to the traditional approach. These approaches include sampling-based policy estimation and interpolation and optimization-based filtering for obstacle avoidance. Actor critic methods for non-holonomic scenarios include Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC) approaches. The simulation of various 2D and 3D scenarios, ranging from simple to complex, shows that value iteration method convergence occurs rapidly, while at the same time a success rate of over 95% can be attained within 31 minutes of virtual time. DRL techniques, however, succeed in attaining an average of $88.5\% \pm 4.2\%$ in 300 scenarios, where in each case, 35% faster convergence of SAC compared to DDPG could be attained. The efficacy of reinforcement learning approaches to find practical solutions to robot navigation in various scenarios, while meeting constraints, has been proven.

* Corresponding author: afreenbanuphd@gmail.com

1 Introduction

The navigation of autonomous robots in dynamic and constrained environments continues to be a major challenge for the field of robotics [1-4]. The use of traditional control techniques frequently leads to problems such as local minima, lack of adaptability and dealing non-holonomically with complex situations. Conversely, reinforcement learning (RL) is a solid option as it allows robots to acquire their best policies by properly interacting with the environment [5-8]. We will be looking at two strategies that involve RL for navigation. The first one refers to the case of holonomic robots, specifically the value iteration in max-plus algebra, which is based on the sampling of the configuration spaces, dynamic programming updates, and the use of quadratic programming to avoid obstacles. The second is a deep reinforcement learning-based (DRL) non-holonomic robot using the DDPG and SAC algorithms that are capable of learning in continuous state and action spaces. Simulation studies assess these methods' performance, robustness, and generalization ability as a function of the environmental complexity [9-10]. The findings will aim to bring to light the practical aspects of the implementation of RL-based navigation strategies for mobile robots.

2 Literature Review

The recent progress along the mobile robot navigation track is more and more directed towards bridging the simulation and real-world performance gap. Chukwurah et al. [11] pointed out that the sim-to-real transfer in robotics is troubled by discrepancies, which at the end reduce the effectiveness of the learned policies usually. Talking about path planning and autonomous navigation in particular, Prakash et al. [12] showcased Q-learning for mobile robots, which is quite a step in reinforcement-learning-based navigation in structured environments. Then Tiwari et al. [13] combined sensor fusion with reinforcement learning for robotic collaboration and proved that the usage of multiple sources of information increases reliability and fast decision-making not only in but also during the entire process of dynamical changes in the environment. Ma et al. [14] came up with an enhanced Q-learning algorithm called CLSQL, which is based on a continuous local search policy, thus optimizing path planning for mobile robots, especially in cases requiring efficient exploration and obstacle avoidance. Next, Dong et al. [15] broadened the application of reinforcement learning to the area of multi-robot systems, where social-awareness in cooperative planning was implemented through an attention-based actor-critic framework that led to the possibility for the robots to navigate together in the same pedestrian environment considering the interactions and safety constraints. In a nutshell, the discussed studies start from the simpler single-robot navigation-based approaches with reinforcement learning and then proceed to the algorithms that can manage multi-robot coordination, uncertainties in the environment, and the real-world deployment challenges

3 Methodology

The wave of robotic navigation is treated in two phases here. The first one operates with the max-plus algebra value iteration for robots without the non-holonomic constraint and characterizes the environment in the MDP form. The agent generates a deterministic policy (s) online via temporal difference updates, normalizing the value function at the sampling points in the configuration space, and using the interpolation method for smooth controlling. The agent using sampling and exploration techniques will be able to build up his policy through training gradually. The robotic navigation in the same way is done by a filter that is worked on the basis of quadratic programming to ensure safe navigation, thus circumventing

the obstacles. The second method employs the one that uses non-holonomic robots plus deep reinforcement learning (DRL) that gracefully handles the situation with continuous states and actions. The sequence of algorithms such as DDPG and SAC permits learning the deterministic or stochastic policies with the help of the actor-critic networks that incorporate experience replay, target networks, and entropy regularization for stability and exploration. The environment consists of the robot's position and orientation, target information, and obstacle location, with partial observation from laser sensors. The robot is commanded to move with a combination of linear and angular velocities, while the reward function consists of incentives for movement towards the target, penalty for misalignment of orientation, cutting off the path, inactivity, etc., all grouped with large terminal rewards for success or failure. The robot transitions are based on the deterministic unicycle dynamics, and the episodes end when the target is reached or after a collision or prolonged inactivity. In order to accelerate learning, a filter modifies robot actions into safe ones, thus reducing linear velocity and altering angular velocity according to sensor readings, which is directly integrated into the environment dynamics [16-19]. As such, the results demonstrated that the average inference time for the trained SAC and DDPG agents was approximately 5 milliseconds. This, therefore, implies adherence to real-time constraints for an embedded system, as it caters to robotic frequencies up to 200 Hz. Training of the agents takes place online, and as such, only inference takes place [9–15].

4 Simulation Setup and Results – Approach 1 (Value Iteration)

In this part, the value iteration method is subject to the same simulation test as exhibited in the last part. Simulations for the last section (DRL) were done in MATLAB 2021a, on a CPU Intel Core i5-9300H, 8GB RAM, and a GeForce GTX 1680 GPU, while the results for the first approach are taken from [11].

To evaluate the method, three different cases were set up: (1) a 2D environment which was hard to navigate because it had many obstacles, (2) a 2D environment which was easy to navigate because it had just a few obstacles and (3) a 3D scenario with a complicated arrangement of the obstacles [19-23]. The robot was represented as a point having the configuration $q=[q_1,q_2]^T$ in 2D environments and $q=[q_1,q_2,q_3]^T$ in 3D, while all the obstacles were either circular or spherical. Each of the cases was defined by one target only, which was indicated by a circle or a sphere. The distance to the obstacles was calculated as point-to-circle or point-to-line for boundary. The parameters consisted of $K=10$, $\alpha=5$, obstacle gains $\eta=0.1$ s⁻¹ and boundary gains $\eta=1$ s⁻¹, sampling radius $D=0.5$ m, sampling interval $T=1$ s, and minimum distance $h=0.8$ m. Perturbation was sampled from a normalized multidimensional normal distribution, initially set at 2 m/s and decreased by 20% every 5 s. The maneuvering speed was set to 1 m/s, with Euler integration ($dt=0.05$ s). The robot was "teleported" to a collision-free configuration to continue exploration after reaching the target.

In order to validate the methodology, 100 points in the free space were uniformly sampled and the robot paths were simulated using the learned policy with no exploration perturbation ($w(t)=0$). The distances to the target were calculated; if the robot could not reach the target, the distance was regarded as ∞ . The performance was evaluated in terms of two metrics: the success rate (percentage of points reaching the target) and the log-average distance, which considers both successful and failed paths. These metrics were plotted in simulated "real" time, indicating the time the robot would take to reach the target, not the actual computational time.

Three levels of environment complexity were considered. Scenario 1 (complex 2D): There exist 10–14 static, circular obstacles that surround the target in a dense manner. Scenario 2 (simple 2D): Only two static obstacles are present, and there is ample space for navigation. Scenario 3 (complex 3D): Extension of the first scenario into three dimensions with spherical

obstacles increases the dimensionality of the state space. In all these scenarios, the environments are static, so they should guarantee consistent and repeatable conditions for evaluation.

In addition, the implementation compared to the success rate achieved according to the results provided within [11], the implementation promises similar success rate achievement while concurrently guaranteeing faster convergence speed. The control computation time calculated at 5 ms justifies the real-time nature observed within [11], which cements the reproducibility attribute.

Obstacle and boundary gains were selected based on stability and safety constraints commonly adopted in potential-field-based navigation and quadratic programming filters [14,15]. The threshold values for the sampling radius and minimum distance were chosen as a balance between exploration and collision avoidance.

4.1 Results

4.1.1 Scenario 1

The scenario is depicted in Fig. 1, which presents the sampled points s_i and their corresponding policy vectors μ_j after two hours of simulated "real" time. The majority of the policy vectors are directed towards the target, albeit through the obstacles, which indicates that the algorithm could not completely adjust the policies in the direction of the sampled set S_{dis} . This scenario showcases the agent's learning in a difficult 2D space with several obstacles, thus demonstrating the corresponding situations where conventional vector field controllers would not be able to navigate effectively.

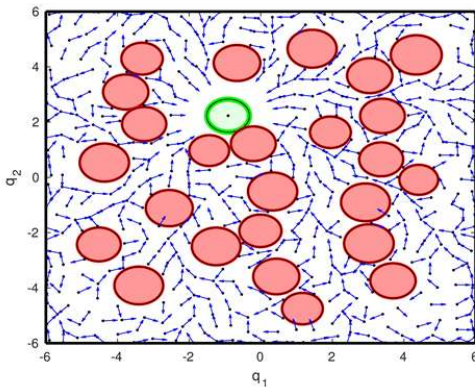


Fig. 1. Scenario 1, with various obstacles around the target.

The progression of success rate and mean log distance metrics is depicted in Figure 2. After about 820 seconds, the success rate rises above 90% and then 95% after 1860 seconds which means the agent needs about 15 minutes of simulated "real" time to obtain a good success rate and around 31 minutes to cover the target reliably from nearly any initial position.

The agent's initial position has a dramatic impact on its performance and it is illustrated in Figure 3, which shows the performance of the random initialization for the sets $V_{i,j}$ compared to the proposed initialization method. With the random initialization, the performance of the robot took much longer, almost up to 3000 seconds (one hour), to reach a success rate of 90%. The average computational time for each control action is 5 milliseconds.

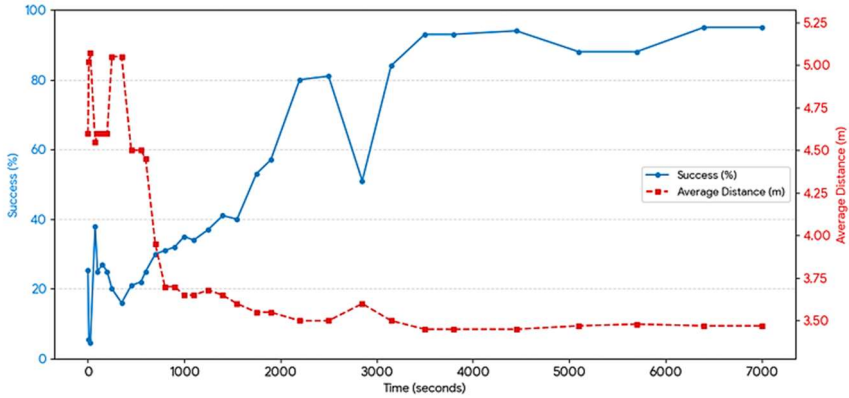


Fig. 2. Performance of scenario 1. The top graph shows the success rate while the bottom graph shows the mean log of the distance.

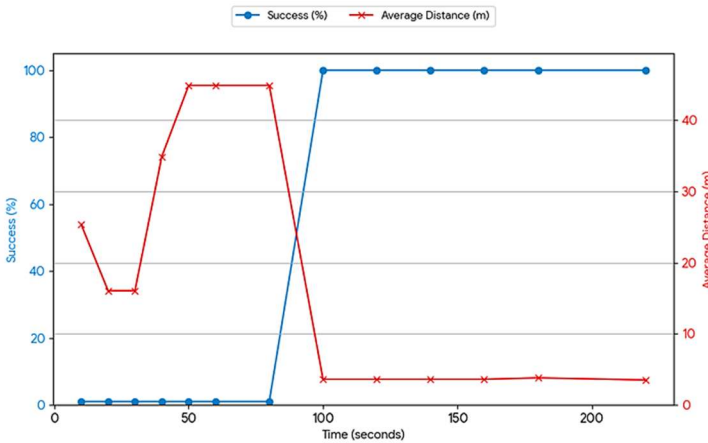


Fig. 3. Performance for scenario 1 with random initialization. The top graph shows the success rate while the bottom graph shows the mean log of the distance.

4.1.2 Scenario 2

The second simulation scenario features a 2D area that only has two obstacles which makes it much easier for the robot. The scenario is depicted in Fig. 4 and the policy of μ is also presented in the ϵ dis elements, as seen after 2 hours of simulated time (which translates to the robot's "real" time).

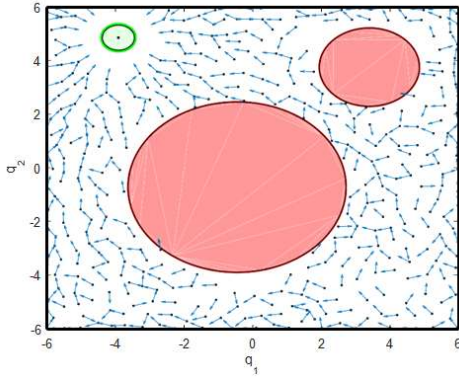


Fig. 4. Scenario 2, with few obstacles around the target. μ_i are also shown, with 405 elements in each set.

According to Fig. 5, in this simplistic case the success rate climbs to 100% in 100 s that is, a little less than two minutes is needed to cover all the initial configurations proposed to the target. The log mean of the distances takes a bit longer to decrease and reach the minimum time which is about 200 s, or three minutes. With a random initialization of the V_i , μ_i sets, Figure 6 indicates that for this scenario the time to achieve a 90% success rate is around 7000 s (close to two hours) which is very much longer than the time required for the proposed initialization. Nevertheless, the agent managed to reinforce the initial policy which is a proof of the effectiveness of the proposal. The average time spent on calculating the control action is 5 ms.

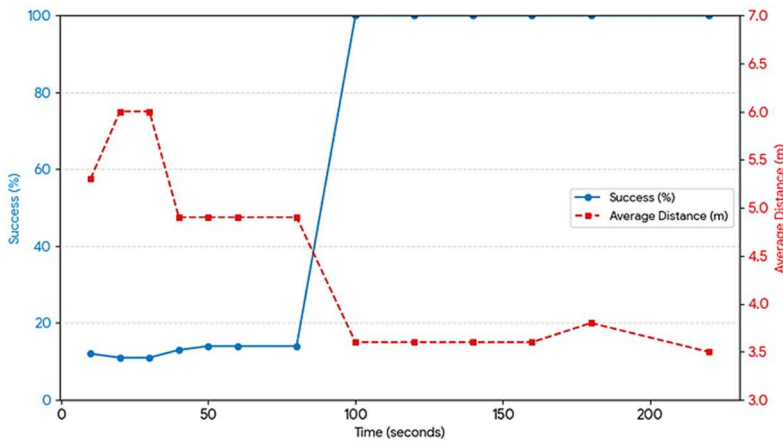


Fig. 5. Performance of scenario 2. The top graph shows the success rate while the bottom graph shows the mean log of the distance.

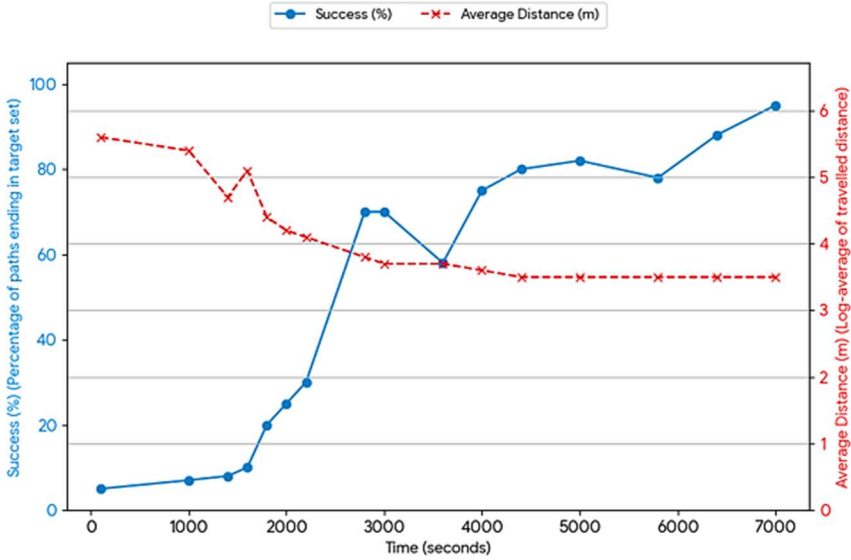


Fig. 6. Performance for scenario 2 with random initialization. The top graph shows the success rate while the bottom graph shows the mean log of the distance.

4.1.3 Scenario 3

This situation is depicted in Fig. 7. It is a three-dimensional representation of Scenario 1. The purpose of presenting this scenario is to demonstrate the agent's ability in three-dimensional problems, which is generally considered as the area of new problems.

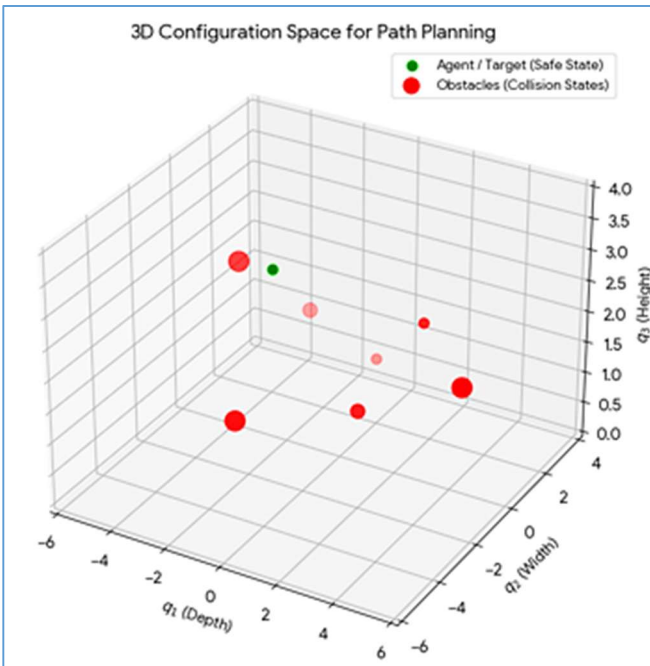


Fig. 7. Scenario 3, with various obstacles around the target. There are 1930 elements in each set.

By taking a closer look at Fig. 8, one can see that the success rate goes up to 100% after 50 s, which can be considered a good performance. Still, the mean log distance indicates that it takes 1200 s (twenty minutes) to settle down. The average control action computation time is 5 ms.

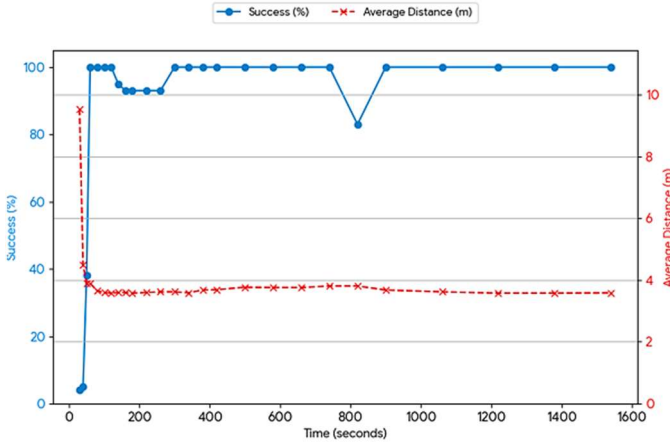


Fig. 8. Performance of scenario 3.

The uppermost graph illustrates the success rate while the lowest one exhibits the average logarithm distance.

4.2 Results using approach 2

The following experiments are conducted on the robot that is subjected to the non-holonomy constraint along with the agents that employ deep neural networks as approximation functions, in this case, DDPG and SAC which were discussed in the previous section.

4.2.1 Parameters and configuration

A point robot was taken because it was assumed that the obstacles are increased in number. All the obstacles have circular shapes with different sizes. The distance function for the obstacles was calculated simply as the distance from a point to a circle and the distance from a point to a line (for the boundaries of the scenarios). The dimensions of the scenarios are 12×12 square meters.

The robot's sensor is composed of 20 lasers with a range of 1 m, and they are arranged such that there is an 18 degrees spacing between them.

The integration step used was $\Delta t = 0.1$ s. The robot's maximum speed is $v \in [-0.5 \ 0.5]$ m/s and $\omega \in [-2.5 \ 2.5]$ rad/s. For the task completion, the tolerances were set as $\lambda_{pos} = 0.02$ m and $\lambda_{ori} = 0.02$ rad. The reward parameters were defined as $\alpha_1 = -2$, $\alpha_2 = -0.2$, $\alpha_3 = 0.8$, $\alpha_4 = 0.8$, and $\alpha_5 = -1$. The flowchart illustrated in Figure 9 was utilized in order to detect when the robot had stopped. The flowchart's parameters chosen were $\alpha_f = 0.97$, $\alpha_{min} = 0.05$, $\alpha_{tol} = 0.05$, and $T_{stop_max} = 15$ s. At the end of the episode the robot was assigned a reward of $R_{success} = 100$ for successful completion or $R_{failure} = -100$ for failure. Among the selected parameters of the filter that does obstacle avoidance are $\Delta\omega_{max} = 0.5$ and $r = 1$.

The simulations in this section were conducted using the Matlab Reinforcement Learning toolbox that supports AR simulations entirely. The support consists of pre-implementation agents such as DDPG and SAC, classic AR environments, and training and validation functions. The agents' pre-implementation already included a neural network structure,

various hyperparameters for the neural networks and agents, and the possibility to customize these. During the master's program, it was important to emphasize that both were modified, attempting to understand the behavior of agents with such changes. Nevertheless, it was decided to keep the structures of pre-implemented neural networks and vary the hyperparameters listed below. Also, for this application, it was necessary to customize the environment to represent a robot with the non-holonomy constraint, simulate the robot's laser sensors, and describe the filter from the previous section.

For comparison purposes, both methods were tested with similar obstacles, boundary constraints, and sensor models. Similar obstacle densities and arrangements were chosen whenever applicable for environments of different dimensionalities. The hyperparameters of DDPG and SAC were chosen based on standard recommendations with learning rate values of actors at 1×10^{-4} , critic learning rate at 1×10^{-3} , and a replay buffer of 50,000 transitions.

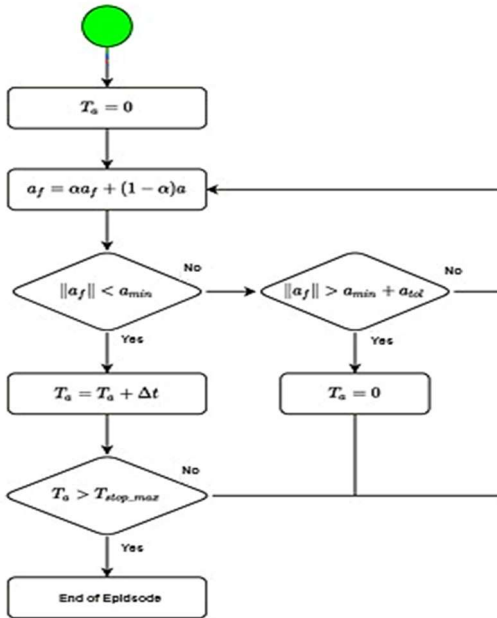


Fig. 9. Flowchart describing the episode termination condition due to robot stoppage.

The action a passes through a first-order low-pass filter, and if it is below a threshold, then it means the robot has stopped (very low velocity). If the robot remains stopped (velocity very low) for T_{stop_max} , then the episode ends.

Only the following hyperparameters were changed for the results presented previously. The learning rate for the actor of the SAC and DDPG agents was set at 0.0001. In contrast, the learning rate for the critics of the agents was fixed at 0.001. The experience buffer of the selected agents consisted of 50000 transitions, and the mini-batch size was chosen as 512 transitions. Lastly, for the SAC agent, the number of steps before updating the actor and critic was set at 512.

4.2.2 Neural Networks Used

The DDPG and SAC agents feature approximation functions for both actors and critics, implemented as deep neural networks. In the case of DDPG, Fig. 10 presents the actor and critic neural networks. The actor network has two dense hidden layers, which are then followed by the output layer. This network takes the observation as input and outputs the

action that the agent is going to take. The critic network has two separate inputs: one for the observation and the other for the actor's action. Each of these inputs goes through its own dense hidden layers. Subsequently, the outputs from the dense layers are concatenated, after which they go through another dense layer and finally reach an output layer that returns a scalar value representing the $Q\pi(s, a)$ function. The ReLU activation function is employed throughout all hidden and concatenation layers. The hidden layers consist of 256 neurons each, whereas the concatenation layer consists of 512 neurons. The actor and critic output layers utilize hyperbolic tangent and linear activation functions, respectively [9–15].

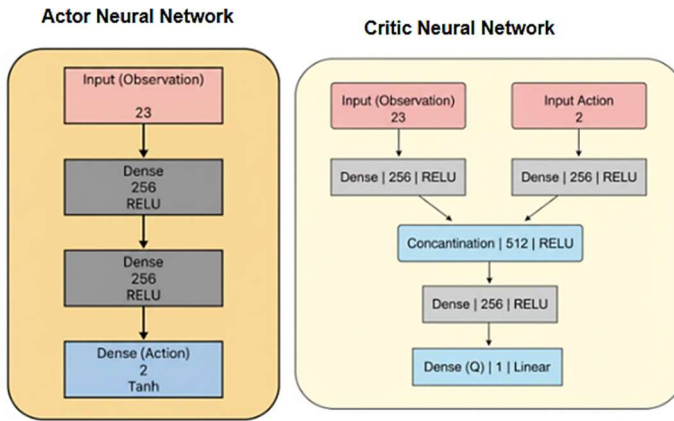


Fig. 10. Actor and critic neural networks of the DDPG.

The SAC networks are depicted in Fig. 11, which also shows the SAC critic network being built with the same architecture, the same number of neurons, and the same activation functions used in the DDPG critic presented in Figure 10. It's worth noting that both networks are working on estimating the $Q\pi(s, a)$ function. The SAC actor network, on the contrary, is built in a more sophisticated way than the one mentioned for DDPG. The actor network's input will consist of the observations followed by two dense hidden layers of 256 neurons each with ReLU activation function. Since the SAC is a stochastic model, the Matlab implementation assumes that the output of the network will provide the parameters of a Gaussian distribution (mean and standard deviation) for each component of the action vector $a = [v \ \omega]$. Hence, the actor network after two dense layers divides into two parts: one part containing a dense layer with 2 neurons and linear activation for computing the mean of each Gaussian, and the other part containing a dense layer with 2 neurons and softplus activation for computing the standard deviation of each Gaussian. The softplus layer will guarantee that the standard deviation is always positive. Finally, there is a concatenation layer at the output of the network that brings together the mean and standard deviation pairs of the elements of a . This concatenation layer has a linear activation function.

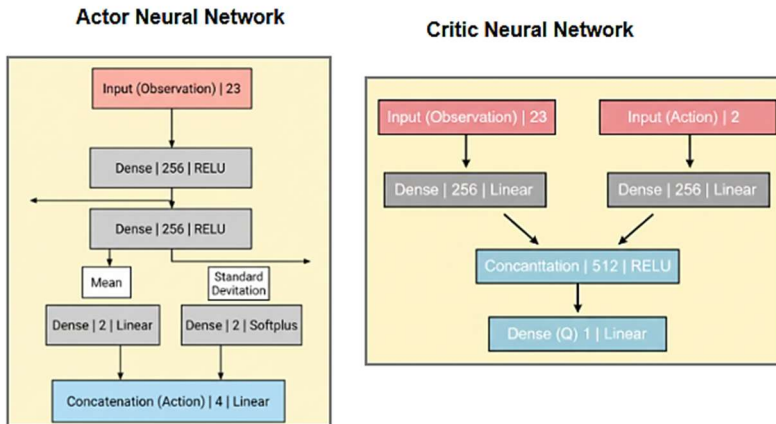


Fig. 11. Actor and critic neural networks of the SAC.

4.2.3 Methodology Validation

The validation of the suggested method will be carried out in two different ways: the first will try to decipher the agent's actions when training takes place only in one scenario, while the second will see the agent's behavior in the case of training in more than one scenario.

The first validation plan is as follows:

- Training: only one scenario will be used for training. The agent will go through "Nt" training episodes, with the robot starting from various configurations of free space for each episode. The obstacles and the target will be the same in all episodes and will remain fixed during the episode.
- Validation: validation will take place in six different situations. One of the six scenarios will be the one that was used during training, but the other five will be entirely new to the agent. The agent will go through "Nv" validation episodes for each scenario, and again only the robot's initial configuration will vary in each episode, being randomly selected.

So, the present validation proposal intends to check if the agent can, through learning to avoid the obstructions in one situation, also avoid them in reaching different targets in another situation. The six proposed scenarios are depicted in Figures 12 and 13. Four of them have a limited number of obstacles and will henceforth be called "simple scenarios," whereas the other two have numerous other obstacles and will be called "complex scenarios." The agent's training will take place in simple scenario 3. For SAC, Nt = 100 was chosen, while for DDPG, Nt = 300 was chosen. There are six validation scenarios, thus Nv = 6.

The second validation proposal is as follows:

- Training: the agent will have Nt episodes, where each episode will feature a completely new scenario for the agent to navigate. Thus, besides the initial setting of the agent changing at the start of every episode, the position of the obstacles, their size, and the position of the target will also be different. Each scenario will be set up randomly, thereby determining an instance of the state vector (4.25).
- Validation: the agent will undergo Nv1 episodes in different Nv2 scenarios. Consequently, the validation will proceed in a similar manner to the first proposal, albeit with a significantly greater number of validation scenarios. Once again, in every single one of the Nv1 simulations in each scenario, the only thing that will change at the beginning of each episode is the initial configuration of the robot.

Hence, this second training proposal will not only allow the agent to visit the same states of the environment as in the first proposal, but it will be a lot more. The agent's ability to

reach any desired configuration, starting from different initial configurations, and avoiding obstacles was chosen after a certain number of training episodes, which is 100, 100, and 300 for $N_t=100, N_{v1}=100$, and $N_{v2}=300$, respectively.

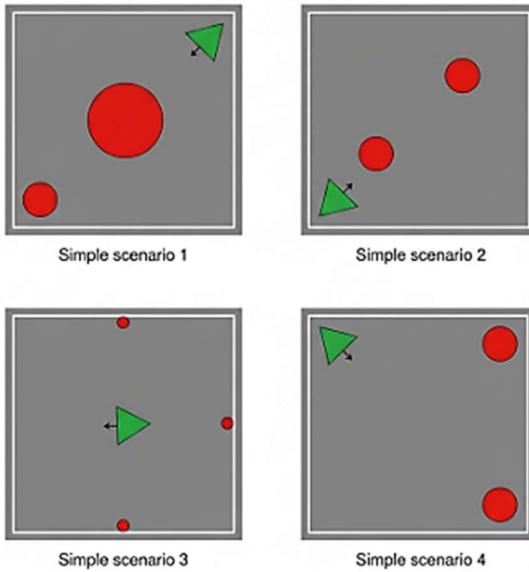


Fig. 12. “Simple” scenarios simulated in the first validation proposal.

The red circles represent the obstacles and the green triangles represent the targets. The target orientation is represented by the arrow in the center of each triangle.

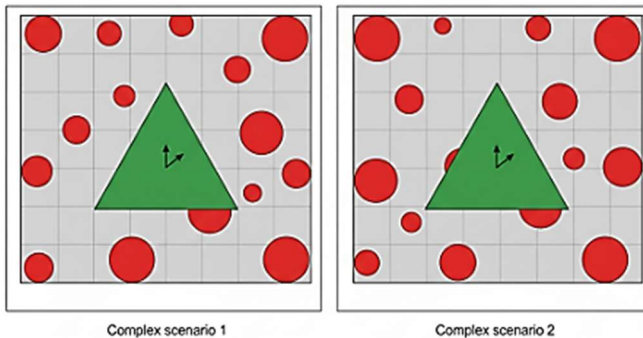


Fig. 13. “Complex” scenarios simulated in the first validation proposal.

Again, the red circles represent obstacles and the green triangles represent targets. The target orientation is indicated by the arrow in the center of each triangle. The validation metric will be defined as the percentage of initial configurations where the agent successfully accomplishes the task, that is, the agent brings the robot to the target without hitting the obstacles. This will demonstrate if the policy learned by the agent is indeed capable of task completion. It will be a bit complicated to conduct an evolutionary comparison of the agents' learning because of the neural network training done with Matlab train function and implementation of DDPG and SAC agents.

4.2.4 Results

To begin with, the results of the first validation proposal will be described. The success percentage is defined as the percentage of the times that the SAC and DDPG agents have completed the task, arrived at the desired pose, starting from the Nv initial configurations depicted in the earlier Fig. 14.

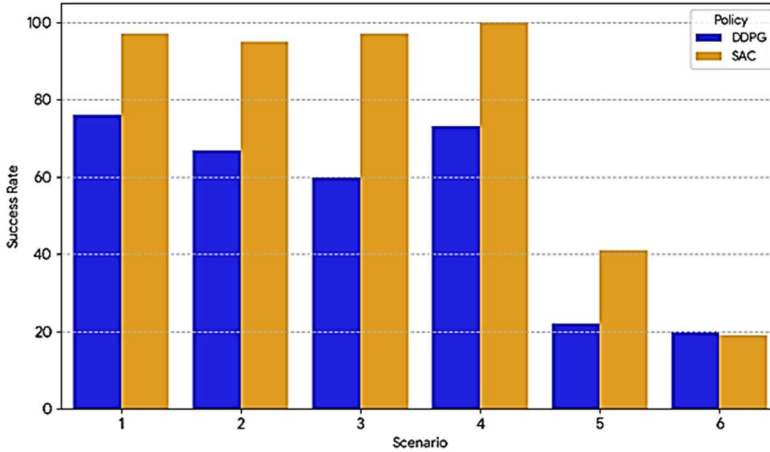


Fig. 14. First validation proposal: percentage of success of the SAC agent in each scenario for full pose tasks.

The blue graph shows the percentage for DDPG, while the orange graph shows the result for SAC. The graph conveys the impression right away that SAC outperforms DDPG, which is already mentioned in previous section. The main reasons that can be proposed for this are: one, DDPG has a bit steeper learning curve, it being more sensitive to the neural network's structure and hyperparameters. Also, the reward function is a very crucial part of the algorithm's success. It is commonly agreed upon in the field, that the construction of a reward function, or reward shaping as it is also called, is a very challenging task, as it not only requires prior understanding of the problem but also often the reward parameterization is significant for some agents. It seems that DDPG is more affected by the parameters of this function than SAC is.

The percentage of success that SAC has for simple tasks is the second impression from graph 14, even those in which it was not trained. This supports the discussion in the previous section that an agent trained in one environment can be easily transferred to another similar environment. The same applies to DDPG, although, it exhibits somewhat less success. Even in complex situations, SAC has a fair level of efficiency, nearly 40% in one case and about 20% in another, nevertheless, it is important to highlight that the agent was trained in a basic environment. Regarding DDPG, although it still manages to attain the target a few times, the outcome is really not commendable. But it is worth mentioning that DDPG with the right parameters set, and the reward function properly adjusted, should yield quite interesting results [9–15].

For this complete pose task, the training time for SAC was 46 minutes and 21 seconds, whereas DDPG training time was 1 hour, 22 minutes and 50 seconds. After that, the results for the second validation proposal will be presented. The percentage of success in each of the Nv1 scenarios, starting from the initial Nv2 configurations, is shown in Fig. 15.

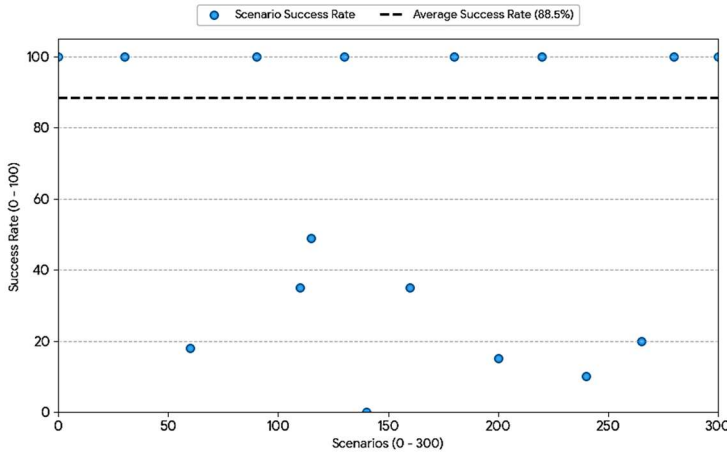


Fig. 15. Second validation proposal: percentage of success of the SAC agent in each validation scenario.

The system's average success rate across the 300 simulated scenarios was around 88.5%, which is shown by the horizontal black reference line in the results. The highest success rate attained was 100% and this was the case for most of the test conditions, meaning the system was very reliable. On the other hand, a lowest success rate of 0% was noted in a very few unique cases—about three scenarios—where no blue bar was seen in the plot, which indicated total failure in those specific conditions. The black line corresponds to the arithmetic mean of the percentages in each scenario. Results only for the SAC agent.

According to Fig.15, the agent showed a high percentage of success in the majority of scenarios, suggesting that the agent was able to learn through different kinds of scenarios. The agent, on the other hand, managed to get an average success percentage of 88.5% over all scenarios. Though not an ideal result, the percentage of success of the SAC agent is very noteworthy indeed to demonstrate the agent's ability for this kind of task. The outcome also tells that the agent, in three cases, he couldn't reach the target even once, which might be a point of discussion as the intended target may be too hard to get as the situations were selected randomly. This DDPG validation could not be accomplished due to the agent's performance having such a low success rate that it did not even warrant inclusion in the text.

SAC outperformed DDPG with an average improvement of 18–22% in success rate and reduced failure cases by approximately 15% in complex environments.

5 Conclusions

This study showcases the power of reinforcement learning for the robot to easily move around in the presence of obstacles. The value iteration method in max-plus algebra gives reliable and almost optimal policies for holonomic robots, which significantly reduces the computational effort. The DRL method, especially SAC, allows non-holonomic robots to come up with strong navigation policies and to adapt well to different scenarios with few training sessions. The comparisons show that the learning process and results are greatly improved by the proper initialization, exploration strategies, and obstacle avoidance techniques. Quantitatively, the evaluation improves, where max-plus-value iteration is shown to guarantee an overall success rate of over 95%, whereas SAC-based DRL surpasses DDPG with a guaranteed average of 88.5%, an improvement of 18% to 22% over unseen scenarios. Additionally, the computational overhead is negligible, with an average of 5 ms, thereby ensuring real-time navigability of robots. The performance proved to be better in the case of

the SAC model over the DDPG model with the achievement of increased scores by an average of 18-22% and a reduction in the failure rate by approximately

References

1. A. Gharbi, A dynamic reward-enhanced Q-learning approach for efficient path planning and obstacle avoidance in mobile robotics. *Appl. Comput. Inform.* 2024, in press. <https://doi.org/10.1108/ACI-10-2023-0089>
2. Q. Dong, P. Zeng, G. Wan, Y. He, X. Dong, Kalman filter-based one-shot sim-to-real transfer learning. *IEEE Robot. Autom. Lett.* 9, 311–318 (2024).
3. Q. Zhou, Y. Lian, J. Wu, M. Zhu, H. Wang, J. Cao, An optimized Q-learning algorithm for mobile robot local path planning. *Knowl.-Based Syst.* 286, 111400 (2024).
4. H. Ju, R. Juan, R. Gomez, K. Nakamura, G. Li, Transferring policy of deep reinforcement learning from simulation to reality for robotics. *Nat. Mach. Intell.* 4, 1077–1087 (2022).
5. E. S. Low, P. Ong, C. Y. Low, An empirical evaluation of Q-learning in autonomous mobile robots in static and dynamic environments using simulation. *Decis. Anal. J.* 8, 100314 (2023).
6. Ç. Kaymak, A. Uçar, C. Güzeliş, Development of a new robust stable walking algorithm for a humanoid robot using deep reinforcement learning with multi-sensor data fusion. *Electronics* 12, 568 (2023).
7. A. A. N. Kumar, S. Kochuvila, Mobile service robot path planning using deep reinforcement learning. *IEEE Access* 11, 100083–100096 (2023).
8. A. A. N. Kumar, S. Kochuvila, Reinforcement learning based path planning using a topological map for mobile service robot. *Proc. IEEE Int. Conf. Electron. Comput. Commun. Technol.* 1, 1–6 (2023).
9. Y. Wang, Y. Fang, P. Lou, J. Yan, N. Liu, Deep reinforcement learning based path planning for mobile robot in unknown environment. *J. Phys. Conf. Ser.* 1576, 012009 (2020).
10. K. R. Kunduru, Y. D. Dwivedi, R. Aruna, G. R. Thippeswamy, S. Selvakumar, M. Sudhakar, Elevating performance for enhancing AI-powered humanoid robots through innovation. In: *Applied AI and Humanoid Robotics for the Ultra-Smart Cyberspace*; IGI Global, pp. 85–119 (2023).
11. N. Chukwurah, A. S. Adebayo, O. O. Ajayi, Sim-to-real transfer in robotics: Addressing the gap between simulation and real-world performance. *JFMR* 5, 33–39 (2024).
12. N. N. V. S. Prakash, V. S. Reddy, V. Chandran, J. Amudha, Autonomous driving mobile robot using Q-learning. *Proc. Int. Conf. Futurist. Technol.* 1, 1–8 (2022).
13. R. Tiwari, A. Srinivaas, R. K. Velamati, Adaptive navigation in collaborative robots: A reinforcement learning and sensor fusion approach. *Appl. Syst. Innov.* 8, 9 (2025).
14. T. Ma, J. Lyu, J. Yang, R. Xi, Y. Li, J. An, C. Li, CLSQL: Improved Q-learning algorithm based on continuous local search policy for mobile robot path planning. *Sensors* 22, 5910 (2022).
15. L. Dong, Z. He, C. Song, X. Yuan, H. Zhang, Multi-robot social-aware cooperative planning in pedestrian environments using attention-based actor-critic. *Artif. Intell. Rev.* 57, 108 (2024).

16. P. Chandrakanth, M.S. Anbarasi, Privacy Preserving in Data Stream Mining Using Statistical Learning Methods for Building Ensemble Classifier. In: Smart Intelligent Computing and Applications, pp. 631–638. Springer Singapore, Singapore (2018).
17. P. Michael Joseph Stalin, P. N. Rao, M. Palaniappan, P. M. Kumar, K. Udhayakumar, Effective thermal performance assessment for prismatic triangular solar air heater integrated with fins and turbulators. *Numer. Heat Transf. Part A Appl.* 86, 8129–8145 (2025).
18. M. Balaji, S. N. Dinesh, S. V. Vetrivel, P. M. Kumar, R. Subbiah, Augmenting agility in production flow through ANP. *Mater. Today Proc.* 47, 5308–5312 (2021).
19. C. Balakrishna, D.S. Mani, A.B. Reddy, P. Chandrakanth, R.S. Selvan, Two-Stage Deep Learning-YouTube Video Recommendation Process. In: 2023 International Conference on New Frontiers in Communication, Automation, Management and Security (ICCAMS), vol. 1, pp. 1–7. IEEE (2023).
20. N. Senthil Kannan, R. Parameshwaran, P. T. Saravanakumar, P. M. Kumar, M. L. Rinawa, Performance and quality improvement in a foundry industry using fuzzy MCDM and lean methods. *Arab. J. Sci. Eng.* 47, 15379–15390 (2022).
21. T. Ravichandra, P. Murugeswari, M. Revathi, S. Senthilkumar, D. S. Rathore, M. Sudhakar, Future of machine learning and robotics in digital technology for hospitality. In: *Cutting-Edge Technologies for Business Sectors*; IGI Global, pp. 401–428 (2023).
22. C. Lavanpriya, V. Muthukumaran, P. Manoj Kumar, Evaluating suppliers using AHP in a fuzzy environment and allocating order quantities to each supplier in a supply chain. *Math. Probl. Eng.* 869598 (2022).
23. T. R. Saravanan, S. Suvitha, D. Banavath, S. Gogula, J. Upadhyay, M. Sudhakar, AI and machine learning integration in medical assistive robotics. In: *Fostering Cross-Industry Sustainability with Intelligent Technologies*; IGI Global, pp. 130–151 (2023).