Indian Institute of Technology Palakkad
भारतीय प्रौद्योगिकी संस्थान पालक्काड
Nurturing Minds For a Better World

EE4120 - W - 2024 : PROJECT 2

CONTROL OF A MOBILE ROBOT USING REINFORCEMENT LEARNING

BTP FINAL REPORT

Krishan Vinod Nair - 122001023
Dheeraj S. Warrier - 122001015

Faculty in Charge - Dr. Shaikshavali Chitraganti
Electrical Engineering
Indian Institute of Technology Palakkad

PHASE 2 : CONTROL OF A MOBILE ROBOT USING REINFORCEMENT LEARNING

Krishan Vinod Nair

Electrical Engineering
Indian Institute Of Technology Palakkad
122001023@smail.iitpkd.ac.in

Dheeraj S Warrier

Electrical Engineering
Indian Institute Of Technology Palakkad
122001015@smail.iitpkd.ac.in

***Abstract*—This study investigates the application of reinforcement learning (RL) algorithms, specifically Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), for controlling a TurtleBot3[2] robot in simulated environments. Through experiments conducted in both obstacle-free and obstacle-laden scenarios, we evaluate the performance of these algorithms in learning navigation policies for the robot. Our results demonstrate the effectiveness of both DDPG and TD3 in enabling the robot to navigate through environments, with TD3 showing slightly better adaptability in environments with obstacles. Then we have investigated the optimization of reinforcement learning parameters for robotic navigation tasks, focusing on the TurtleBot3 platform. Through meticulous parameter tuning and simulation-based training, an optimal policy is developed to guide the bot's behavior in a controlled environment. Real-world deployment of the trained model on the TurtleBot3 Waffle Pi reveals challenges and insights into the translation of simulation results to practical applications. Despite obstacles encountered during real-world testing, the study demonstrates the potential of AI-driven navigation in diverse environments. The findings underscore the importance of iterative refinement and adaptation in the development of autonomous robotic systems. Through perseverance and innovation, researchers are poised to unlock the full capabilities of intelligent and autonomous robots in various industries and applications.**

***Index Terms*—Gazebo, TurtleBot3, DDPG, TD3**

I. INTRODUCTION

In recent years, advancements in robotics and artificial intelligence have led to the development of sophisticated algorithms for autonomous navigation and control. Reinforcement learning (RL) algorithms, such as Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), have gained prominence due to their ability to learn complex behaviors directly from raw sensor inputs.

In Phase 1, we had focused on learning the major algorithms and implementing it on a small scale project to learn the effectiveness of each algorithm and how it works exactly. Now we shift to apply our learnings on the actual TurtleBot3 we want to work with, so before we get on the hands on work we test our algorithms using simulations in the Gazebo environment.

In this report, we present our exploration and implementation of DDPG and TD3 algorithms in the context of robotic control, specifically applied to a TurtleBot3 Waffle robot simulated in the Gazebo environment. The objective of our project was to investigate the performance of these RL algorithms in various simulated environments and evaluate their effectiveness in learning navigation policies for the robot.

The report is structured as follows: firstly, an introduction to Gazebo, the simulation environment utilized for our experiments, and discusses the rationale behind its selection. Subsequently, we describe our experimental setup, including the simulation environment in Gazebo and the configuration of the TurtleBot3 Waffle robot. We then discuss the methodology employed for training the RL agents using both algorithms and the environments used for evaluation.

Our experiments encompassed several simulated environments with varying complexities, including open spaces and obstacle courses. For each environment, we conducted extensive training sessions to allow the RL agents to learn optimal navigation policies, as well as for us to understand how the algorithms reacts to different factors and get a grasp on its working. Throughout the training process, we monitored key performance metrics such as episode rewards and convergence behavior.

Furthermore, we conducted a comparative analysis of the performance of DDPG and TD3 algorithms across the different environments. By examining metrics such as training time, final performance, and robustness to environmental changes, we aimed to gain insights into the strengths and weaknesses of each algorithm in the context of robotic control tasks.

Finally, we present our findings and discuss the implications of our results, including potential avenues for future research and practical applications of RL algorithms in real-world robotic systems. Overall, this report serves as a comprehensive exploration of the application of DDPG and TD3 algorithms in simulating robotic navigation tasks using the TurtleBot3 Waffle Pi Bot in Gazebo.

II. GAZEBO ENVIRONMENT

Gazebo serves as the virtual playground where the TurtleBot3 can learn, explore, and develop its capabilities in a simulated environment before deployment in the real world.

The Python code acts as the controller, orchestrating the bot's actions and learning process within this simulated environment.The Python code interacts with Gazebo by sending commands to the TurtleBot3, receiving sensor data, and controlling its behavior. This allows us to test and refine the bot's algorithms and behaviors in a simulated environment before deploying it in the real world.

The reasons why Gazebo has been chosen as the simulation environment for this project are as follows:

- Realistic Simulation: Gazebo provides a highly realistic simulation environment, including accurate physics engines and sensor models. This realism allows for more reliable testing and evaluation of robotic algorithms before deploying them on physical hardware.
- Customizable Environments: Gazebo offers the flexibility to create custom environments tailored to specific experimental needs. This capability enables researchers to simulate a wide range of scenarios and test algorithms under various conditions, such as different terrains, lighting conditions, and obstacles.
- Support for Robotics Platforms: Gazebo supports numerous robotic platforms and sensors, making it suitable for simulating a diverse range of robots and hardware configurations. In this project, the compatibility with the TurtleBot3 Waffle platform likely influenced the choice of Gazebo as the simulation environment.
- Open-Source and Extensible: Gazebo is an open-source project with an active community of developers and users. Its open nature allows for easy customization and integration with other software tools and frameworks, including reinforcement learning libraries like OpenAI Gym.
- Cost-Effectiveness: Using Gazebo for simulation reduces the need for expensive physical hardware, saving both time and resources. It also enables researchers to iterate quickly and experiment with different algorithms without the constraints of physical equipment.
- Validation and Verification: Simulation in Gazebo allows researchers to validate and verify algorithms in a controlled environment before deploying them in real-world scenarios. This process helps identify and address potential issues early in the development cycle, improving the overall robustness and reliability of the algorithms.

Algorithm 1 Algorithm for TurtleBot Navigation in Gazebo

1: Initialization:
Initialize ROS nodes and required services.
Initialize the TurtleBot environment with the necessary parameters and setup.
Initialize the respawn mechanism for the goal position.
2: Environment Setup:
Define the Env class, which encapsulates the TurtleBot environment.
Implement methods for obtaining robot odometry, receiving laser scan data, setting/resetting goal positions, calculating state representations, calculating rewards, and executing actions.
3: ROS Communication:
Establish communication with ROS topics and services to interact with the Gazebo simulation environment.
Subscribe to the laser scan topic to receive sensor data.
Publish velocity commands to control the TurtleBot's movements.
4: TurtleBot Control:
Implement methods to control the TurtleBot's movements based on velocity commands.
Handle actions received from the DDPG algorithm.
5: Respawn Mechanism:
Implement a respawn mechanism for the goal position to provide a continuous environment for training.
Define methods to respawn the goal position within the environment in specified places.
6: Applying the Algorithm to the TurtleBot:
7: Collision and Goal Handling:
Handle collision events and adjust rewards accordingly.
Detect goal arrival and trigger goal-related actions (e.g., respawning the goal, updating episode status).
8: Visualization and Logging:
Visualize training progress by plotting learning curves.
Log important events, such as collisions, goal arrivals, and training statistics.
9: Model Saving:
Save trained actor and critic models for future use or evaluation.
10: Conclusion:
Terminate the ROS nodes and cleanup resources.
Provide summary and concluding remarks.

A. Algorithm

The algorithm can be seen in Algorithm 1.

B. Reward Function

The reward function[3] for the algorithm is as follows:

$$r(S_t, a_t) = \begin{cases} r_{arrive} & \text{if } d_i < C_d \\ r_{collision} & \text{if } d_i < C_o \\ \alpha * (d_i - d_{i-1}) + \beta * (\theta_i - \theta_{i-1}) & \text{otherwise} \end{cases}$$

Here d_i is the distance of the bot from the target, C_d is the distance from the target at which we can declare that the bot has reached the target. C_o is the minimum distance from the obstacle less than which will cause a collision.θ_i is the angle that the bot is facing from the line connecting the bot and the goal. α and β are just the multiplicative constant factors for angle reward and distance reward. The values we took for this project are r_{arrive} as 1000, $r_{collision}$ as -1000, α as 500, and β as 3.

C. Reset Conditions

The environment is programmed to reset under several conditions:

1) Collision with Walls: If the robot collides with any of the walls or obstacles within the environment, the simulation

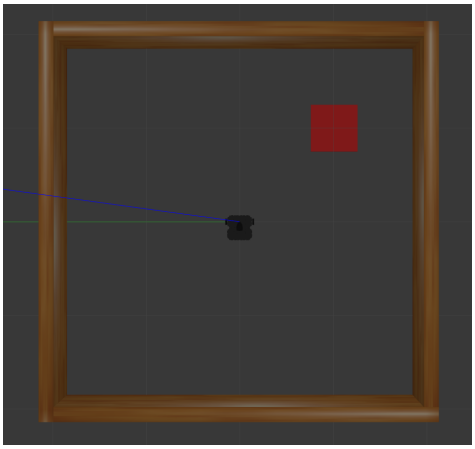

Fig. 1. No Obstacles Environment showing TurtleBot3 Waffle Pi at the center/origin and the red square is the target location

is reset, and the robot is respawned back at the center of the square.

2) Staying in the Same Position: If the robot remains in the same position for more than 20 consecutive steps without making progress towards the goal, indicating a potential deadlock or inability to navigate, the simulation is reset.
3) Time Limit: If the robot fails to reach the goal within 25 seconds from the last time it reached a goal, the environment is reset. This time limit encourages efficient navigation and prevents the robot from getting stuck indefinitely in circles.

These reset conditions ensure that the simulation remains dynamic and challenging for the robot, promoting robust navigation behavior.

III. DDPG SIMULATION IN AN OPEN ENVIRONMENT

We started the simulations of our TurtleBot3 with the DDPG algorithm[5] in an open environment, which is a no-obstacle course.

The Gazebo environment utilized in this project is based on the ”turtlebot3_square” environment provided in the company’s GitHub page[3]. This environment is specifically designed for simulating navigation tasks for the TurtleBot3 Waffle robot. The environment resembles a typical indoor environment. In this setup, a goal is randomly spawned at one of four locations: (1,1), (-1,1), (1,-1), or (-1,-1). These goal positions represent target locations that the TurtleBot3 Waffle robot must navigate to within a specified time frame. If it reaches the target, a new target is spawned at one of the other 3 locations and the bot has to navigate to the new target from this new position that it is in.

The total time taken for training the model was 36557.71 seconds, which is about 10.15 hours. Below are the training results:

- Fig 2. shows that in each episode, the TurtleBot3 travels a different distance. It appears to start at around 0.5 units and then slowly increases over the course of the episodes. By episode 700, the farthest distance traveled is around 4 units. What we understand from the plot is that, in the initial episodes, the agent might be exploring more, leading to potentially shorter distances due to unsuccessful actions or collisions. As it learns, it might prioritize exploitation, leading to a more consistent and potentially higher distance travelled, but with a temporary drop as the agent transitions between exploration and exploitation strategies.

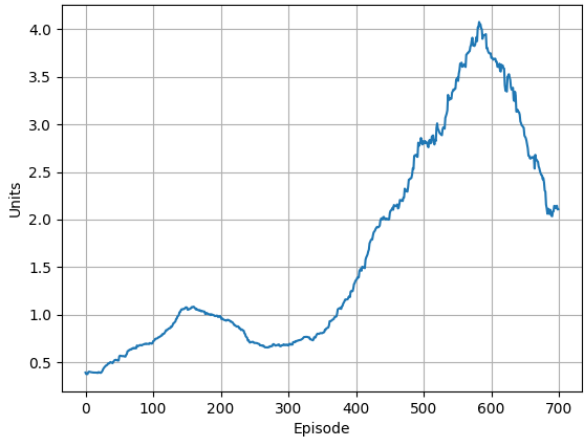

Fig. 2. Distance travelled in each episode for DDPG with no obstacles

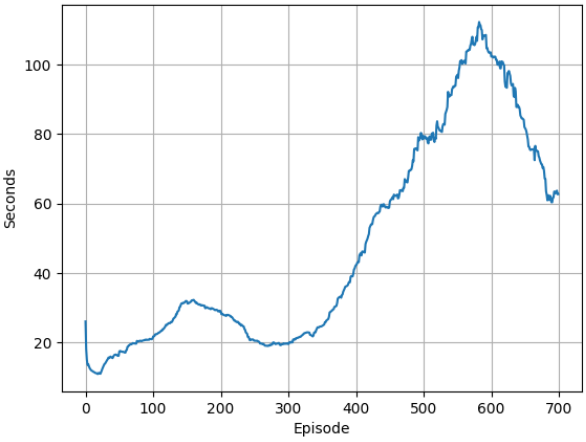

Fig. 3. Timespan of each episode for DDPG with no obstacles

- In Fig 3. what we can note is that it is very similar to the distance plot, that means the bot is constantly moving and isn’t stopping anywhere until collision in an episode, else the time for an episode would have shot up while the distance traveled would have remained zero (such a scenario can be observed in further cases)

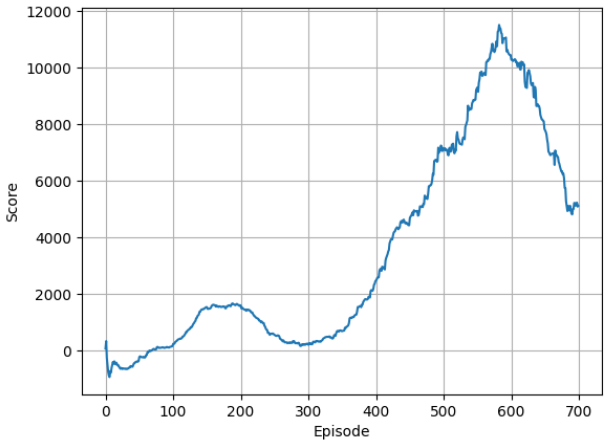

Fig. 4. Q-Value Plot for DDPG with no obstacles

- In Fig 4. what we incur is that a steeper increase at the beginning is possible as the agent rapidly explores and encounters new states. As it learns from these experiences, it updates its Q-values to reflect the potential for higher rewards associated with certain actions in those states. The ideal scenario is an increasing trend in the average Q-value over episodes. This signifies that the agent is learning better policies (action choices) to achieve higher rewards in the simulation environment.The increase might slow down or plateau after a considerable amount of episodes, the agent might have discovered the most rewarding actions in most states, and further exploration might not yield significantly better Q-values. It is also possible that the DDPG algorithm might have converged to a local optimum, meaning it has found a good policy but there might be even better policies to explore.

It can be seen that the three plots are following the same trend and are very similar to each other. One of the possible reason is because of the reward function design. Since the reward function in the simulation is designed to provide higher rewards for travelling further distances, then the improvement in distance travelled (Fig 1) would naturally lead to an increase in the average Q-value (Fig 3) as the agent learns to take actions that maximize those rewards. Also a consistently high reset time (Fig 2) across episodes indicates the TurtleBot3 might be very good at avoiding collisions with the boundaries. This would allow it to focus on covering more distance within the central area of the environment, contributing to the increasing distance traveled (Fig 1)

IV. DDPG SIMULATION IN A MULTIPLE OBSTACLE ENVIRONMENTS

In this phase of training, we are revisiting the training process from scratch, focusing on the navigation algorithm's performance in a modified environment with four obstacles. Each obstacle at each corner of an imaginary square centered at origin as shown in Fig.5 , introducing new challenges for navigation. By retraining the model in this updated environment, we aim to further enhance its adaptability and robustness in navigating through cluttered spaces. This approach allows us to evaluate the algorithm's ability to handle multiple obstacles simultaneously and develop effective obstacle avoidance strategies. Through iterative training in diverse environments, we strive to optimize the navigation behavior of the robot for real-world deployment. The total time taken to train this model was 14400 seconds which is about 4 hours.

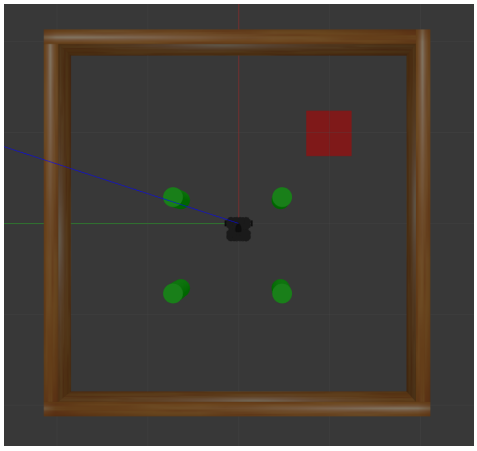

Fig. 5. 4 Obstacles Environment showing TurtleBot3 Waffle Pi at the center/origin and the green cylinders are the obstacles while the red square is the target location

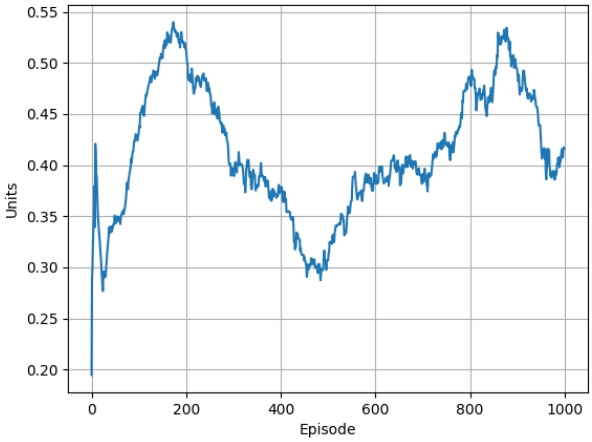

Fig. 6. Distance travelled in each episode for DDPG with 4 obstacles

Below are the training results:

- In Fig 6 the general trend in the plot appears to be upward, which suggests that the TurtleBot3 is learning to

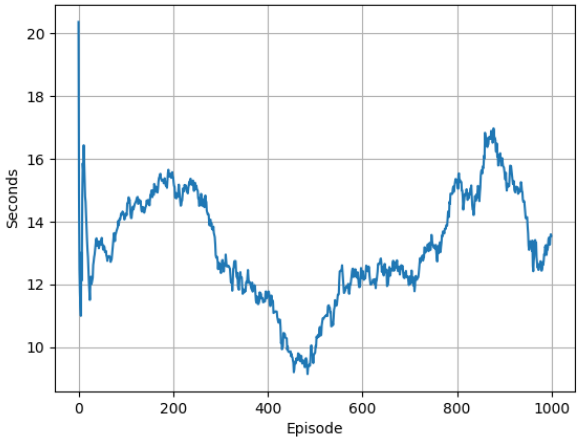

Fig. 7. Timespan of each episode for DDPG with 4 obstacles

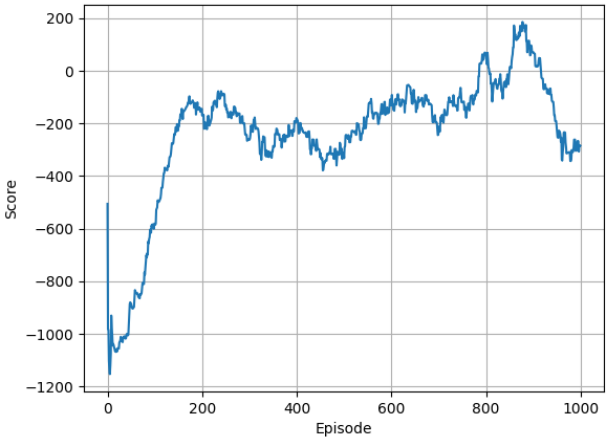

Fig. 8. Q-Value Plot for DDPG with 4 obstacles

navigate the environment with obstacles more effectively as the number of episodes increases. This is a positive sign, indicating that the DDPG algorithm is enabling the agent to improve its performance over time. The fluctuations seen can possibly be because of the exploration-exploitation trade-off. But in this case its also possible that the distance travelled can be affected by how the TurtleBot3 interacts with the obstacles. Collisions with obstacles can lead to shorter total distances in some episodes.

- In Fig 7, it is seen that the reset times throughout the episodes seem to be consistently low, but the fluctuations can be mainly because of the near misses. Even if the TurtleBot3 avoids collisions entirely, close encounters with obstacles might trigger the reset mechanism due to safety thresholds being reached momentarily.

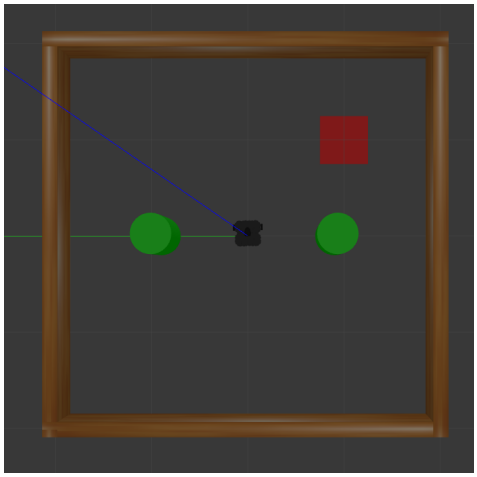

Fig. 9. 2 Obstacles Environment showing the TurtleBot3 Waffle Pi at the center/origin and the green cylinders are the obstacles while the red square is the target location.

- Fig 8. shows an increasing trend in the average Q-value over episodes. This signifies that the agent (TurtleBot3) is discovering actions that lead to higher expected future rewards in the simulation environment with obstacles. The graph plateaus as it might have converged to a local optimum, meaning it has found a good policy but there might be even better policies to explore as it can be seen that the score is still about -200 and this requires us to train the model for considerable number of episodes for better results.

After observing persistent collisions with the obstacles during training in the environment with four obstacles, we have identified narrow pathways as a significant contributing factor in the learning process. To facilitate more effective learning and navigation, we have decided to simplify the environment by reducing the number of obstacles to two obstacles, positioned on either side of the origin as shown in Fig.9 .This adjustment aims to provide the robot with wider pathways to maneuver through, reducing the complexity of the environment and enabling the algorithm to focus more on learning efficient obstacle avoidance strategies. By repeating the training process in this modified environment, we anticipate improving the robot's ability to navigate safely and smoothly, laying the foundation for robust performance in real-world scenarios. The training time for this model was 32462.54 seconds which is about 9 hours.

Below are the training results:

- Again in Fig 10. the increasing trend shows that the TurtleBot3 is learning to navigate the environment with obstacles more effectively as the number of episodes progresses.Sharp drops in distance travelled could indicate the agent encountering a new obstacle configuration or difficulty it has not learned to handle effectively yet.
- Fig 11. supports the previous plot and is similar to

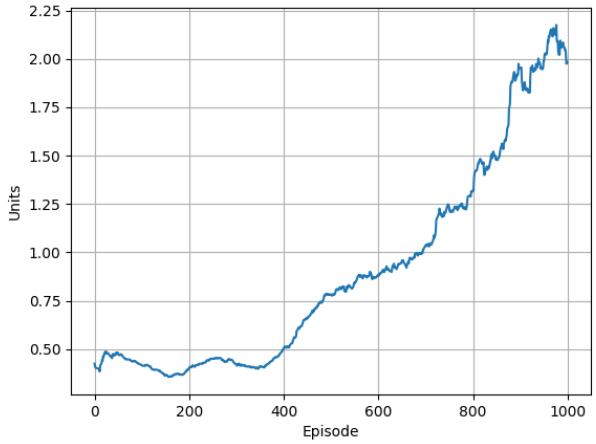

Fig. 10. Distance travelled in each episode for DDPG with 2 obstacles

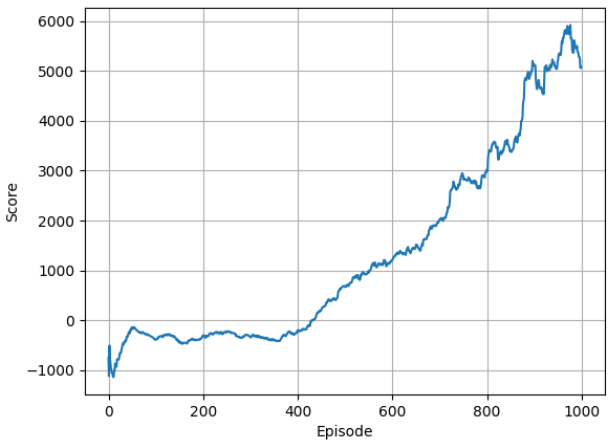

Fig. 12. Q-Value Plot for DDPG with 2 obstacles

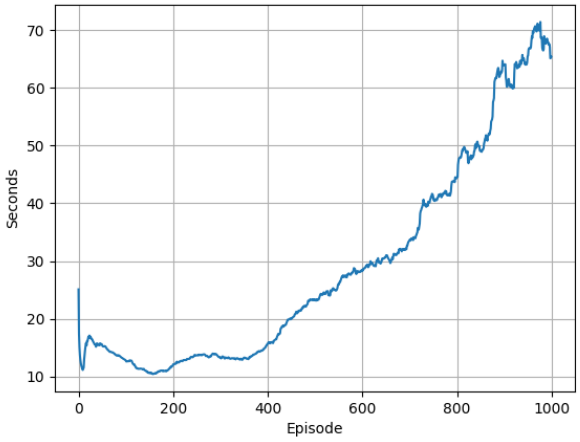

Fig. 11. Timespan of each episode for DDPG with 2 obstacles

it.The TurtleBot3 seems to be learning well, traveling further (Fig 8) with consistently low reset times (Fig 9). This suggests efficient obstacle avoidance, but some fluctuations might be due to exploration or near misses.

- Fig 12. Q value plot is also is closely similar to the previous plots indicating that it is possibly reaching a good policy as score is increasing with distance travelled in each episode.

V. TD3 SIMULATION IN A MULTIPLE OBSTACLE ENVIRONMENT

Our next step involves comparing two different ways of helping our TurtleBot3 Waffle robot navigate through tricky environments. We're going to test out Twin Delayed Deep Deterministic Policy Gradient (TD3)[1]. We'll run these tests in the same simulated environment where there are four obstacles placed strategically. By doing this comparison, we aim to understand which method helps the robot handle obstacles better and move around more effectively. This information will be crucial for us in deciding which method to use to make our robot's navigation abilities stronger and more reliable for real-world situations. The training time for this model was 15398.80 seconds which is about 4.27 hours.

Below are the training results:

- Fig 13. shows a general upward trend, which suggests that the TurtleBot3 is learning to travel farther distances as the number of episodes increases. There is also a small peak in the middle of the plot. The small peak could represent an episode where the TurtleBot3 explore a part of the environment that it had not visited before. This could have led to the TurtleBot3 traveling a longer distance than usual.The small peak could also represent an episode where the TurtleBot3 was able to take a more efficient path to its goal. This could have allowed it to travel a longer distance than usual.
- In Fig 14. the initial peak at the beginning (around episode 0) is likely due to the TurtleBot3 having no prior knowledge of the environment. So, in the first few episodes, it might be exploring its surroundings randomly, which can take a long time or just moving at a very slow speed. As the training progresses, the algorithm tries to learns a policy for navigating the environment. The small peak in the middle of the plot (around episode 500) could be because the TurtleBot3 encountered a new situation during this episode that it had not experienced before. This could have caused it to take longer to explore.
- In Fig 15 the plot shows a generally increasing trend, which suggests that the Q-value, and therefore the expected future reward, is increasing as the number of episodes increases. The small peak could also be due to

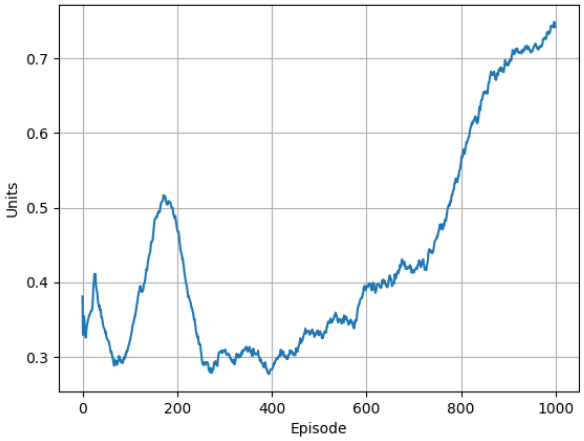

Fig. 13. Distance travelled in each episode for TD3 with 4 obstacles

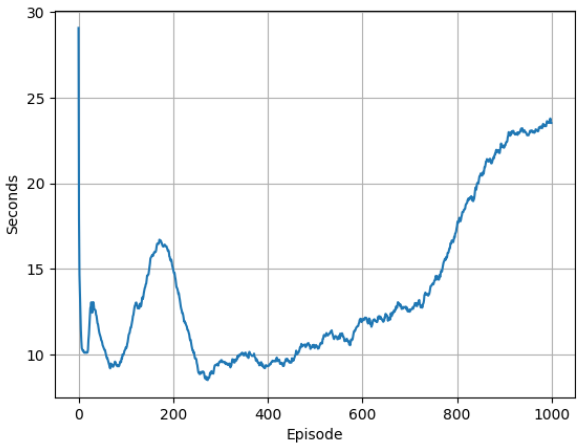

Fig. 14. Timespan of each episode for TD3 with 4 obstacles

the exploration strategy of the TD3 algorithm. During training, the algorithm might explore different actions to gather more information about the environment. This exploration might have led to a higher reward in this particular episode.

It can be noted that the plot hasn't plateaud yet and the score at the end is quite low which indicates that an optimal policy is yet to be learnt, which requires us to train it for further more number of episodes. The environment with four obstacles might be challenging for the TurtleBot3 to navigate efficiently. More training episodes might be needed for the algorithm to learn a policy that consistently achieves high rewards.

Similar to our adjustment with the DDPG algorithm, we'll now decrease the obstacles to two for the TD3 algorithm.

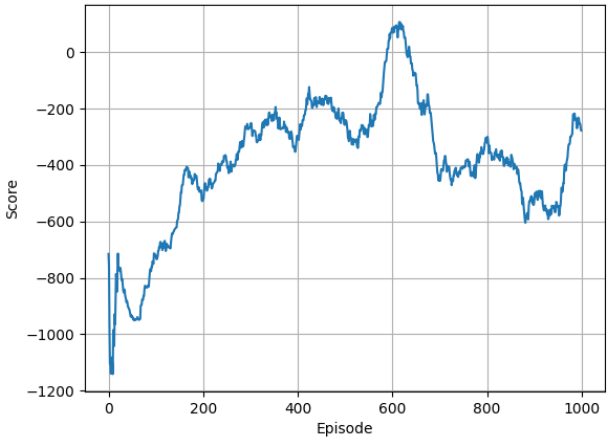

Fig. 15. Q-Value Plot for TD3 with 4 obstacles

This change simplifies the environment, making it easier for the algorithm to learn effective navigation strategies. By reducing complexity, we aim to evaluate how TD3 performs in a simpler setting compared to its performance with four obstacles. This step helps us understand the algorithm's adaptability and guides our decision-making for the robot's navigation capabilities. The training time for this was 23294.6 seconds, which is about 6.47 hours. .

Below are the training results:

- In Fig 16. even though the general trend is increasing, as the TurtleBot3 learns, it initially make mistakes that result in lower distances traveled. Over time, the performance should improve, but there can be fluctuations as the algorithm is still acquiring knowledge about the environment.
- In Fig 17. it follows the same trend and is very similar to the case of other scenarios where the plots look similar, the reasons remaining the same.
- In Fig 18. again a plateau isn't reached yet but it has acquired a decent score at the end of training. In the case of TD3 it is seen that letting the model train for more episodes will help reach steady score, but it still follows a steady learning.

VI. COMPARISON BETWEEN RESULTS

A. 4 Obstacles vs 2 Obstacles

Reducing the number of obstacles from four to two reveals a notable trend in the performance of both DDPG and TD3 algorithms, as indicated by the plotted scores over the same number of episodes in Fig.19 and Fig.20 . The scores, representing the effectiveness of navigation, demonstrate a significant increase across both algorithms in the simplified environment. Specifically, the scores achieved by both DDPG and TD3 are substantially higher compared to their performance in the more complex four-obstacle setting. This improvement suggests that

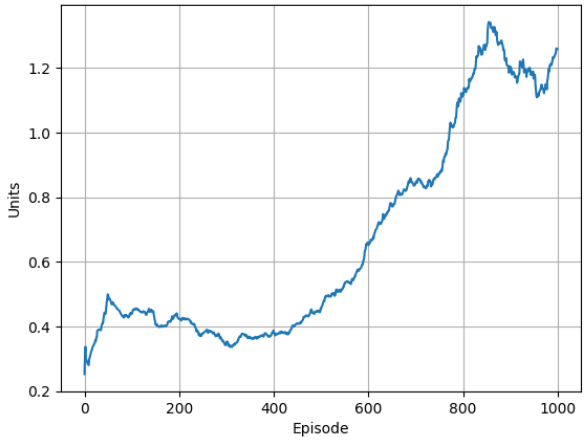

Fig. 16. Distance travelled in each episode for TD3 with 2 obstacles

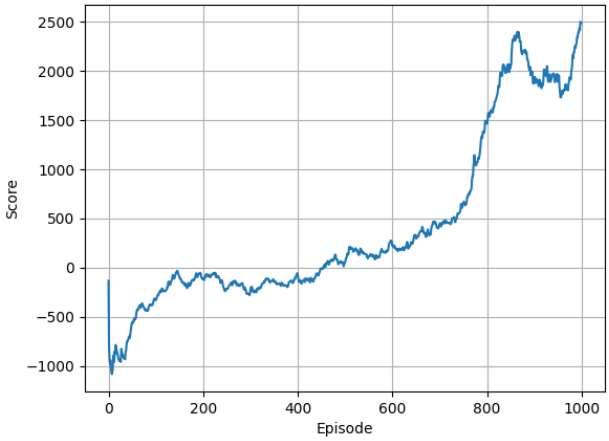

Fig. 18. Q-Value Plot for TD3 with 2 obstacles

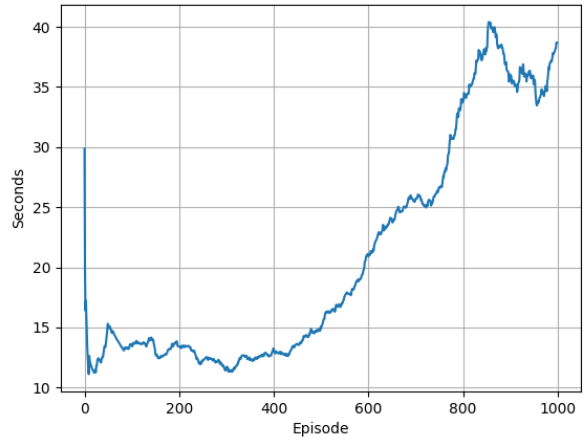

Fig. 17. Timespan of each episode for TD3 with 2 obstacles

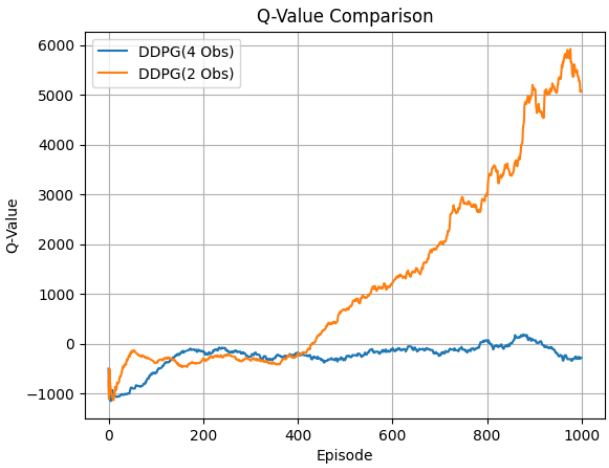

Fig. 19. Q-Value Comparison between DDPG in a 4 Obstacles Environment and in a 2 Obstacles Environment

with fewer obstacles to navigate around, the robots trained using DDPG and TD3 exhibit enhanced proficiency in traversing the environment. Such findings underscore the importance of environmental complexity in influencing the learning process of reinforcement learning algorithms.

B. DDPG vs TD3 for 4 Obstacle Environment

As the environment becomes more complex the performance of TD3 is slightly higher at the end of the 1000 episode mark. This can mean that TD3 can perform better in complex environments than DDPG. It can be because TD3 reduces overestimation and hence Q-value doesn't vary much or suddenly change.

C. DDPG vs TD3 for 2 Obstacle Environment

As we can see that for the 2 Obstacle Environment the DDPG algorithm score is much higher than for TD3. Some reasons for this could be:

- Reduced exploration needed: With only two obstacles, the environment might be simpler to navigate, requiring less exploration compared to the environment with four obstacles. DDPG, which might prioritize exploration more than TD3 in early stages, could potentially exploit this simpler environment quicker, leading to higher initial Q-values.
- TD3 Avoids Overestimation: By taking the minimum between both critics the Q-Value from the TD3 algorithm is always more pessimistic and that might lead to slow increase in Q-Value.

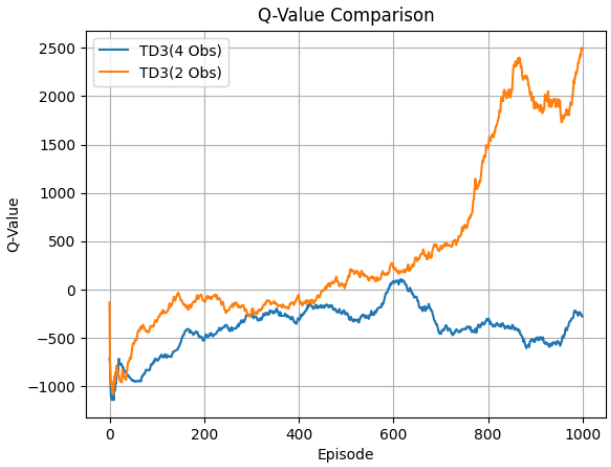

Fig. 20. Q-Value Comparison between TD3 in a 4 Obstacles Environment and in a 2 Obstacles Environment

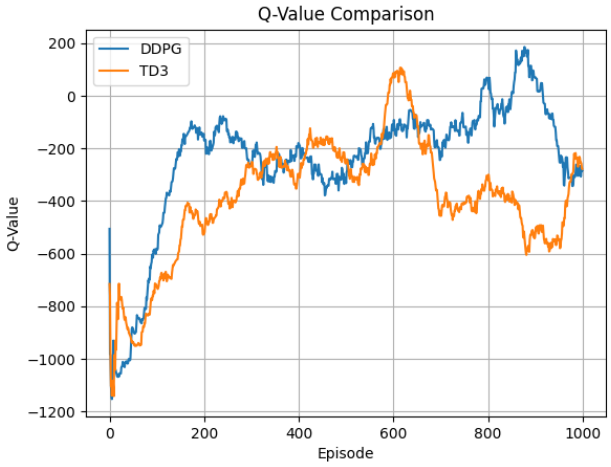

Fig. 21. Q-Value Comparison between DDPG and TD3 for 4 Obstacles Environment

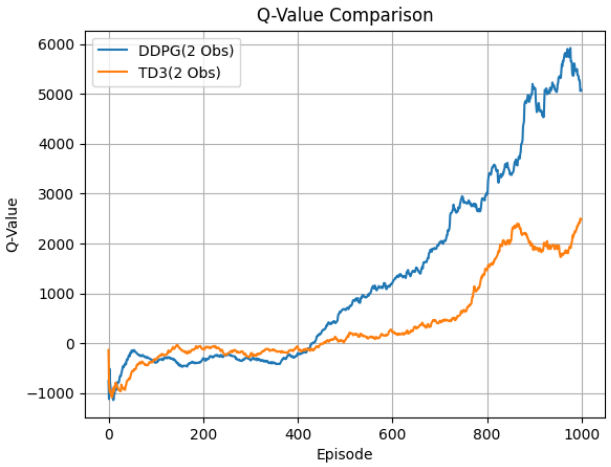

Fig. 22. Q-Value Comparison between DDPG and TD3 for 2 Obstacles Environment

Fig. 23. Training Time Comparison

VII. CONCLUSION OF TD3 AND DDPG COMPARISION

In this study, we explored the application of reinforcement learning (RL) algorithms, specifically Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), for controlling a TurtleBot3 robot in simulated environments. We trained and evaluated both algorithms in various scenarios, including open spaces and environments with multiple obstacles.

Our results indicate that both DDPG and TD3 showed promise in learning effective navigation policies for the TurtleBot3. In environments with no obstacles, the RL algorithm demonstrated a steady increase in distance travelled over episodes, suggesting successful learning. However, when introduced to environments with obstacles, both DDPG and TD3 algorithms faced challenges, with varying degrees of success in navigating through cluttered spaces.

Comparing the results between DDPG and TD3, we observed that DDPG showed slightly better performance in terms of adapting to environments with obstacles which might mainly be because TD3 hadn't been trained enough. While both algorithms showed improvements over training, TD3 exhibited a more stable learning curve and achieved higher rewards in some cases. The link to all of the code can be found here.

Overall, our study highlights the potential of RL algorithms in enabling autonomous navigation for robotic systems. Future work could focus on further optimizing these algorithms, exploring different reward functions, and enhancing their robustness in real-world scenarios.

VIII. OPTIMUM POLICY

To improve the effectiveness of the reinforcement learning approach, a systematic examination of parameter setups using the Deep Deterministic Policy Gradient (DDPG) framework was conducted. Extensive tweaking of critical parameters, such as learning rate, every component of the reward function, and exploration noise, was required for the the study. Every change in a parameter was a purposeful experiment designed to determine the best setup for training the TurtleBot3. Extensive experimentation and thorough evaluation were carried out to determine how parameter changes affected model performance after learning. This thorough investigation not only produced insightful information about the algorithm's sensitivity to various environments, but it also made it easier to create a finely calibrated reinforcement learning framework that was customized to the particular dynamics of the TurtleBot3 environment.

A. Initial Model Configuration

Reward Function:

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ & \text{if } dist\_rate > 0 \\ -10 & \text{if } dist\_rate < 0 \end{cases}$$

Noise Factor: σ = 0.15

Time limit to travel from one goal to another = 25 seconds

1) Result: The results can be seen in Fig 24,25 and 26.

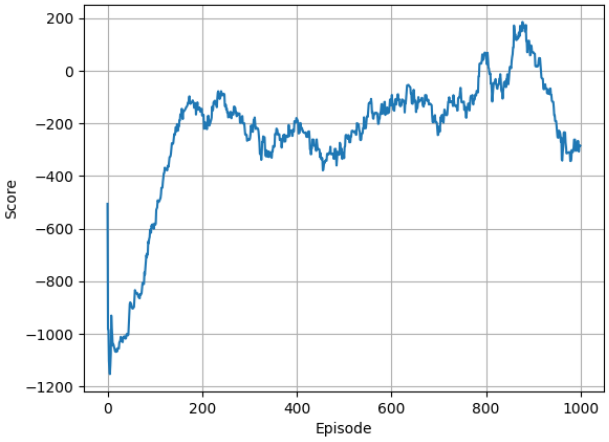

Fig. 24. Q-Value Plot for the Initial Configuration

2) Conclusions: As we saw that the result for training the bot in this original configuration for 1000 episodes in the 4 obstacle environment had a negative score and was still unable to make it to even one goal on average. To try to teach the robot to avoid obstacles we will introduce a new category of reward called avoidance reward. This will contain the penalty that the bot will receive if it gets too close to an obstacle and its rate is decided by the $avoid\_rate$ which is also the set distance around the obstacle that if the bot comes closer than, then it would start receiving the penalty and $dist\_obs$ is the distance of the obstacle from the bot after it is closer than the set distance.

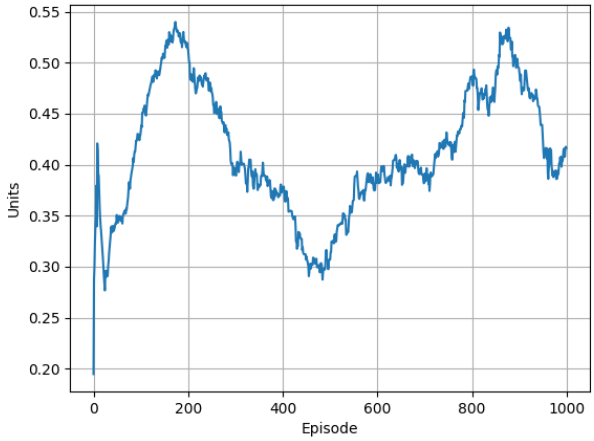

Fig. 25. Distance plot for the Initial Configuration

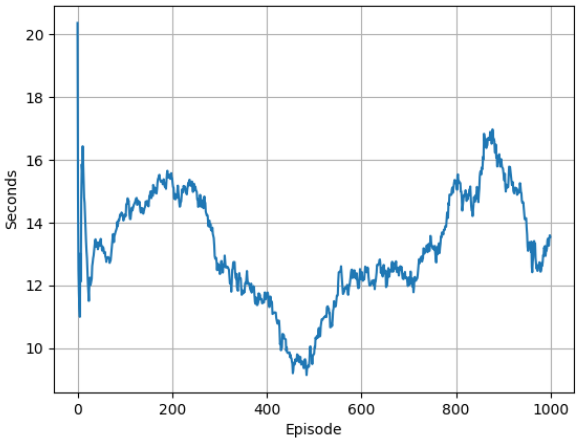

Fig. 26. Time plot for the Initial Configuration

B. Addition of Avoidance Penalty

Reward Function:

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -100 * avoid\_rate & \text{if } dist\_rate > 0 \\ -10 & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.29-$dist\_obs$

Noise Factor: σ = 0.15
Time limit to travel from one goal to another = 25 seconds

1) Result: The results can be seen in Fig 27,28 and 29.

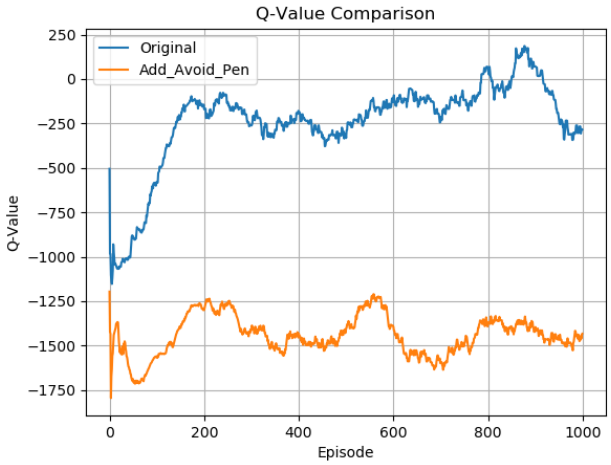

Fig. 27. Q-Value Plot

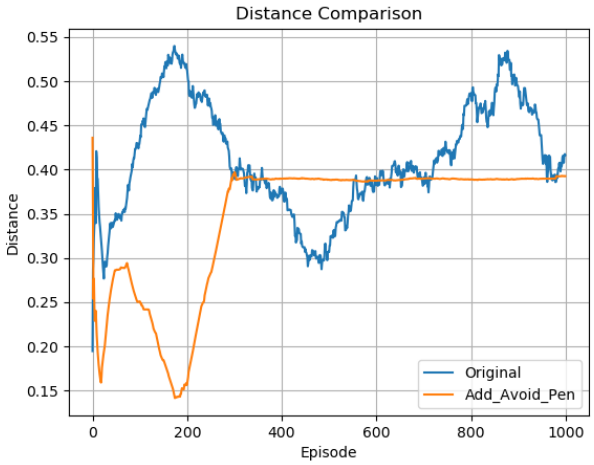

Fig. 28. Distance plot

2) Conclusions: The avoidance rate was set as 0.29 which meant that the bot would start receiving penalty as soon as it comes closer than 0.29 meters from an obstacle which was a very big boundary around the obstacle that the bot couldn't enter without getting penalties so it was reduced to 0.14 and then finally to 0.09 as the zone which the bot should avoid. Adding this avoidance penalty was proving to be a good decision even though the scores are much more negative as the bot had started avoiding going very close to the obstacles at even small number of episodes teaching it to avoid the obstacles in a better way than just penalizing it for finally colliding. The scores are much more negative as there is a high negative penalty for going close to the obstacle. The bot still does move away from the goal even after training for a 1000 episodes and therefore changing the penalty for a negative distance rate as a linearly varying penalty according to its speed moving away from the target to see if that will make the bot learn to move towards the target. To be able to see difference avoidance rate can make we simulated it in the 2 obstacle environment so that we can see which factor can quickly adapt to the obstacles while also moving towards the target.

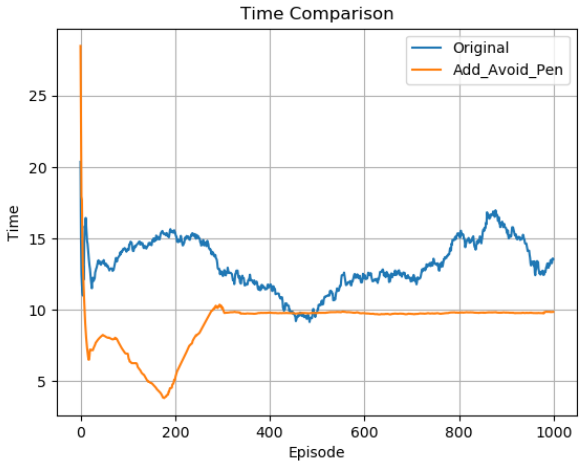

Fig. 29. Time plot

C. Modification for dist<= 0

Reward Function: The results can be seen in Fig 30,31 and 32.

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -100 * avoid\_rate & \text{if } dist\_rate > 0 \\ 500 * dist\_rate - 100 * avoid\_rate & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.09-$dist\_obs$
Noise Factor: σ = 0.15
Time limit to travel from one goal to another = 25 seconds

1) Result:

2) Conclusion: It became evident that implementing a linearly varying penalty for negative distance rates did not yield favorable results compared to the fixed negative penalty. As a result, the decision was made to revert to the fixed negative penalties. Additionally, the angular range for penalty imposition was adjusted from [0, 2π] to [-3π/2,3π/2], enabling the bot to incur penalties for deviations in any direction from the goal. Now that the angle reward is negative for the bot facing away from the goal therefore we can now add it to the negative penalty given for moving away from the target as well. The avoidance factor was also changed from 100 to 1000 to make the penalty for avoidance comparable to the reward for moving towards the goal. Also the time limit for traversing

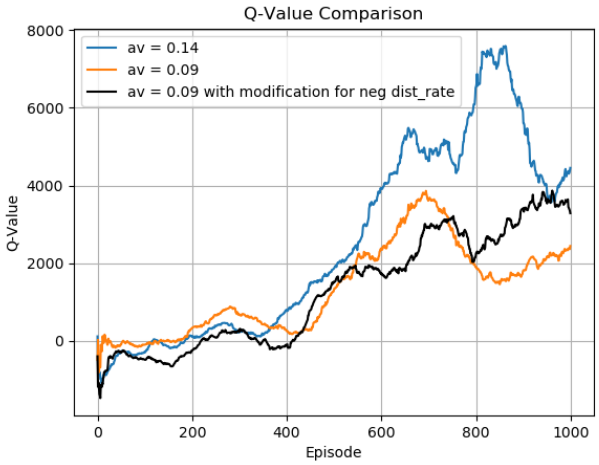

Fig. 30. Q-Value Plot

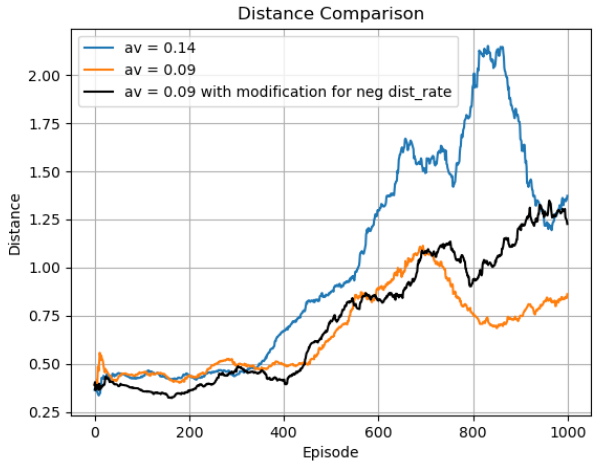

Fig. 31. Distance plot

from one goal to another was extended to 40 seconds to allow for more comprehensive learning iterations. These simulations were done in the 4 obstacle environment to see how the overall reward function acts in an obstacle filled environment.

D. Adjustment of Negative Rewards

Reward Function:

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.09-$dist\_obs$

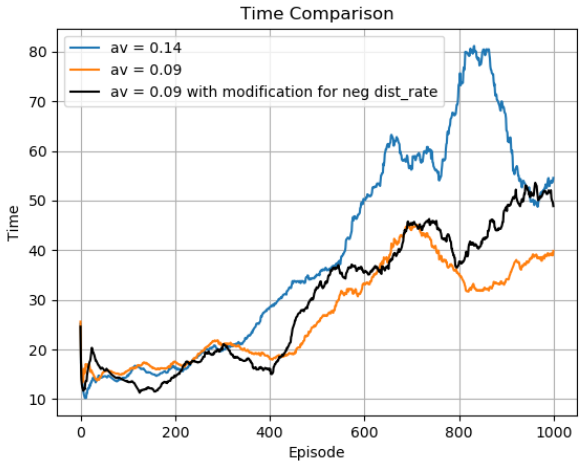

Fig. 32. Time plot

Noise Factor: σ = 0.15

Time limit to travel from one goal to another = 40 seconds

1) Result: The results can be seen in Fig 33,34 and 35.

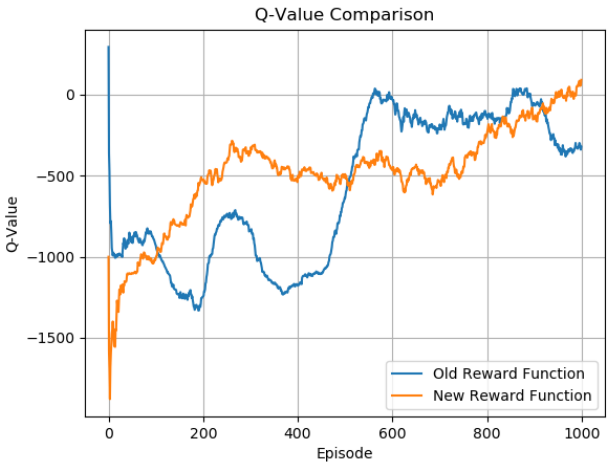

Fig. 33. Q-Value Plot

2) Conclusion: Making the penalty for moving away from the goal a fixed negative value and making the avoidance factor as 1000 from 100 has made the performance of the bot a lot better and the scores have increased. The bot still collides after a thousand episodes quite frequently therefore trying 2000 and 1500 as the avoidance factor to compare the results and find the better avoidance factor for our result. These simulations were also done in the 4 obstacle environment.

E. Experimenting with Avoidance Factor

Reward Function:

Fig. 34. Distance plot

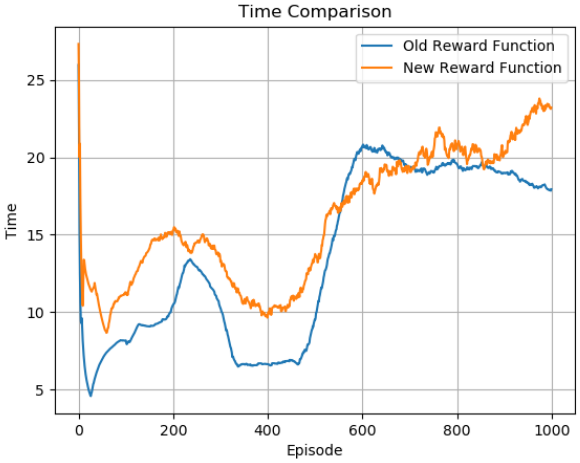

Fig. 35. Time plot

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -avoid\_factor * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -avoid\_factor * avoid\_rate & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Factor: $avoid\_factor$ = 2000 or 1500
Avoidance Rate: $avoid\_rate$ = 0.09-$dist\_obs$
Noise Factor: σ = 0.15
Time limit to travel from one goal to another = 40 seconds

1) Result: The distance travelled and time taken is highest in the condition where avoidance factor is the highest that is 2000 but the average score after 1000 episodes is almost equal to the score for avoidance factor as 1000 showing that the bot has started to move around the obstacles a lot but not moving towards the target while doing so. The results can be seen in Fig. 36,37 and 38.

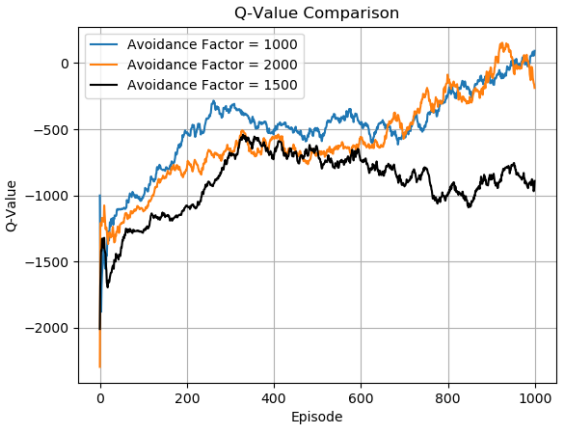

Fig. 36. Q-Value Plot

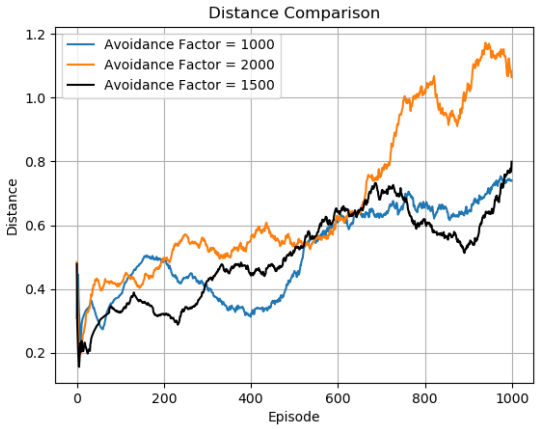

Fig. 37. Distance plot

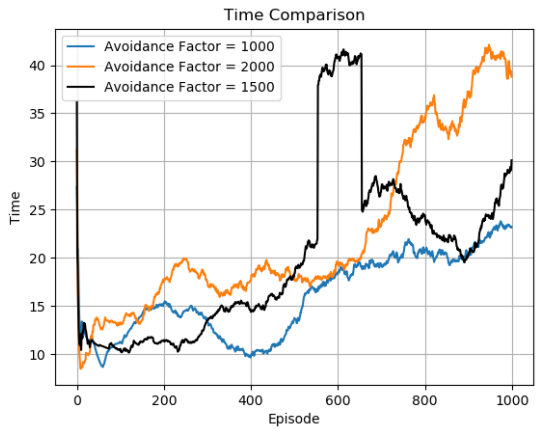

Fig. 38. Time plot

2) Conclusion: In an endeavor to further reduce the likelihood of collisions, experimentated with with different values for the avoidance factor. Initially, a value of 2000 was trialed, which resulted in the bot exhibiting excessive caution, often hesitating and oscillating without successfully reaching the goal within the designated timeframe. Subsequently, the avoidance factor was reduced to 1500, yet this adjustment proved ineffective as the bot continued to exhibit erratic behavior and occasional collisions. Therefore as the results of 1500 and 2000 were not satisfactory we will go ahead with avoidance factor as 1000 and to gain deeper insights into the bot's learning trajectory, an extensive training regimen comprising 4000 episodes was undertaken in an environment featuring four obstacles.

F. Increased Training Episodes

Increasing training episodes in reinforcement learning enhances exploration, improves policy convergence, boosts robustness to variability, enables exploration of complex strategies, and addresses under-exploration issues, ultimately enhancing agent performance and learning. With that intention the model was trained for 4000 episodes.
Reward Function:

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.09-$dist\_obs$
Noise Factor: σ = 0.15
Time limit to travel from one goal to another = 40 seconds

1) Result: The distance and time plot almost level off after 500 episodes showing that the bot must have encountered an obstacle that it was unable to learn to pass. The results can be seen in Fig 39,40 and 41.

2) Conclusion: Even after training for 4000 episodes, results showed that after about 1000 episodes, the bot's performance leveled off and it became more adept at avoiding obstacles than at achieving its objective of reaching or navigating towards the goal. As a result, training was carried out in the four-obstacle environment over 1000 episodes, and the avoidance factor was further decreased to 500 to see if it facilitates the bot moving towards the goal rather than just avoiding the obstacles. Reducing noise and adjusting the sigma parameter in reinforcement learning can help improve training stability and convergence.

G. Noise Reduction and Sigma Adjustment

Noise in the learning process can sometimes lead to erratic behavior or suboptimal policies. In analyzing the influence of noise on the bot's decision-making process, it was discovered that the magnitude of noise occasionally exceeded the maximum allowable action, thereby impeding learning efficacy. To fix this issue the sigma of the Ornstein-Uhlenbeck

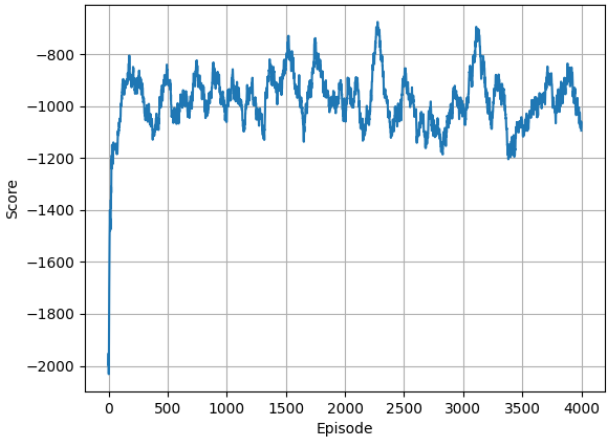

Fig. 39. Q-Value Plot

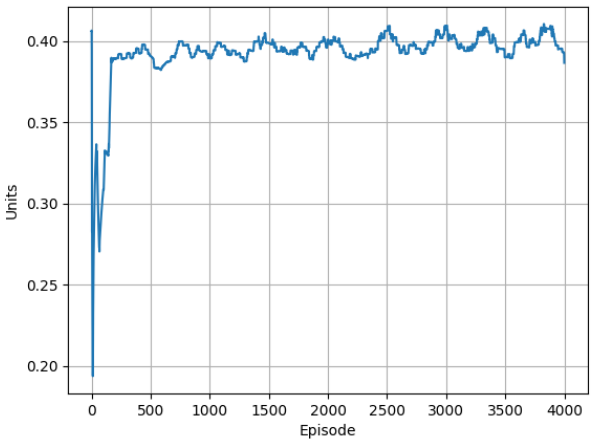

Fig. 40. Distance plot

Noise (OUNoise) was reduced from 0.15 to 0.05 to ensure noise levels remained within an optimal range during training iterations.

Reward Function:

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -500 * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -500 * avoid\_rate & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.09-$dist\_obs$
Noise Factor: σ = 0.05
Time limit to travel from one goal to another = 40 seconds

1) Result: The results can be seen in Fig 42,43 and 44.

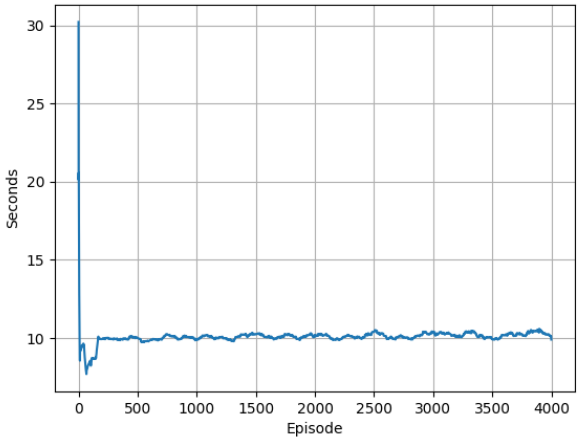

Fig. 41. Time plot

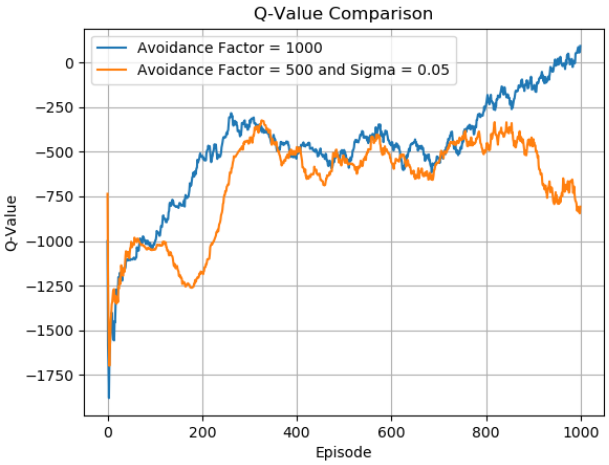

Fig. 42. Q-Value Plot

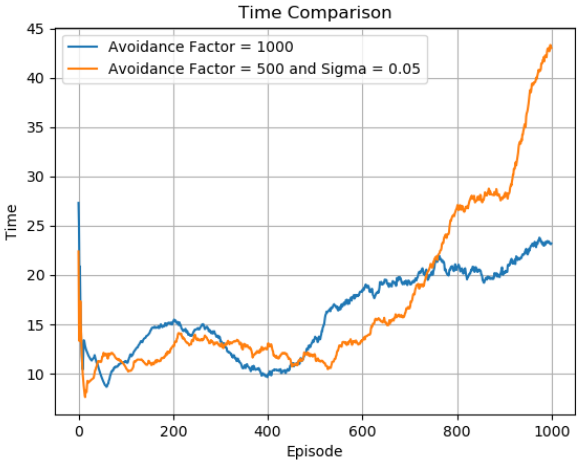

Fig. 44. Time plot

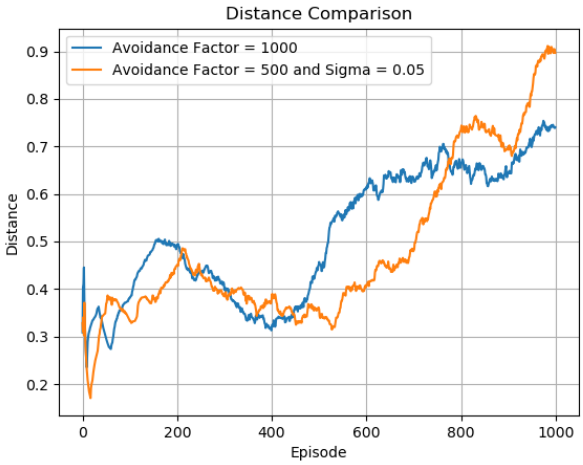

Fig. 43. Distance plot

2) Conclusion: Now final scores were still not very high and reducing the avoidance factor did not seem to help so now trying to give avoidance rate as a fixed value of 0.03 when the bot enters the set distance around an obstacle rather than dynamic based on the distance of the bot from the obstacle to check if it helps the bot in the learning process. The sigma parameter for the noise was also further reduced to about 0.025 to see its impact on the scores and remove any sort of extra unwanted variance in the training.

H. Fixed Avoidance Rate and Sigma Adjustment

Reward Function:

$$r(S_t, a_t) = \begin{cases} 1000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate < 0 \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.03
Noise Factor: σ = 0.025
Time limit to travel from one goal to another = 40 seconds

1) Result: The results can be seen in Fig 45,46 and 47.

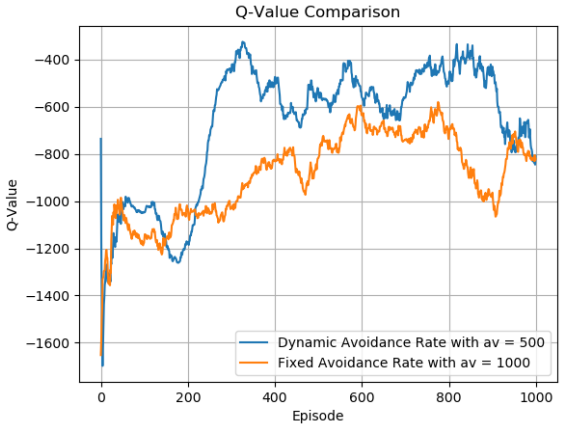

Fig. 45. Q-Value Plot

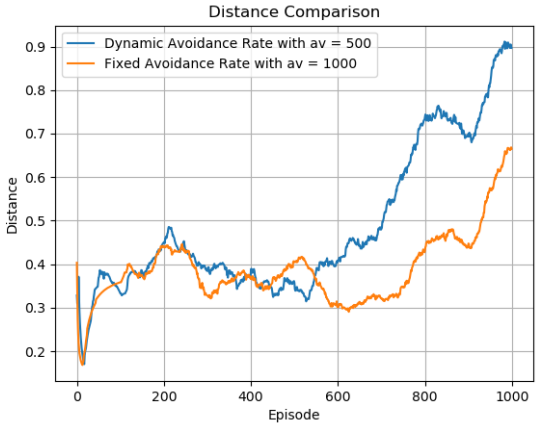

Fig. 46. Distance plot

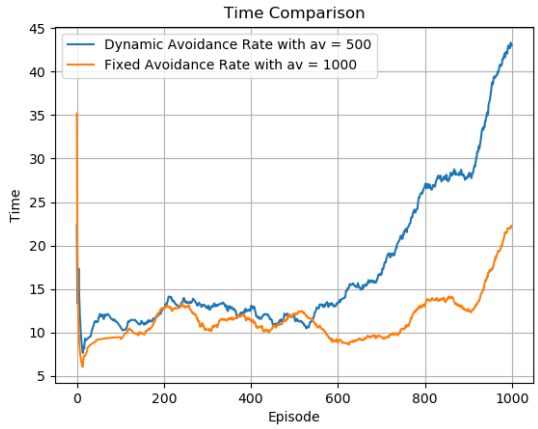

Fig. 47. Time plot

2) Conclusion: Changing avoidance rate to a fixed value seems to make navigation very difficult for the bot as it doesn't want to go anywhere near the obstacle. The bot avoided obstacles to such an extent that it failed to progress towards the goal. Consequently, the avoidance rate was reverted to a dynamic value, and the reward for reaching the goal was increased to 2000 to reinforce goal-directed behavior since the bot still does not reach the minimum of even 1 goal every episode. Also a bug in the code pertaining to penalty assignment for failure to reach the next goal within the allotted time was rectified, with a penalty of -250 now being imposed for such instances. The penalties for crossing the timeframe was being reflected in the plot but not in the learnings or the information given to the bot.

I. Back to Dynamic Avoidance Rate and Increased Goal Reward and Bug Fixes

Reward Function:

$$r(S_t, a_t) = \begin{cases} 2000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate < 0 \\ -250 & \text{goal not reach-ed within } 40sec \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.09 - $dist\_obs$
Noise Factor: σ = 0.025
Time limit to travel from one goal to another = 40 seconds

1) Result: The distance and time plots still seem to level out after some episodes making the bot get stuck at that decision. The results can be seen in Fig 48,49 and 50.

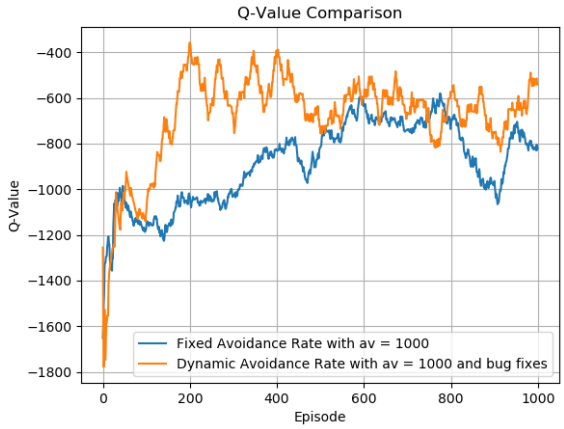

Fig. 48. Q-Value Plot

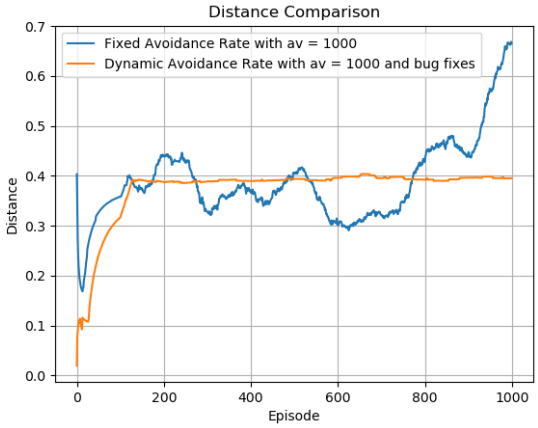

Fig. 49. Distance plot

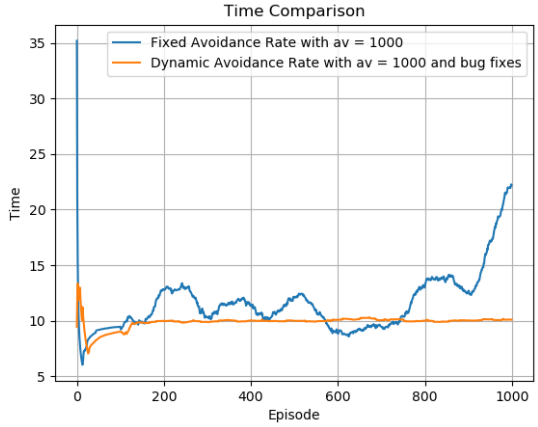

Fig. 50. Time plot

2) Conclusion: This configuration gives a satisfactory output at the end of 1000 episodes. The bot is still has not managed to reach at least 1 goal at an average even at the end of the 1000 episodes but it can be credited to the fact that the bot has too many actions or policies to explore which slows down the learning and even reaching one goal becomes a difficult task. And so the bot also gets stuck at obstacles that it fails to understand how to overcome. So taking this into consideration we have decided to try to teach the bot in steps. Teaching it to handle environments with lesser obstacles then introducing it to an environment with more obstacles.

J. Stacked Learning

After meticulous experimentation and parameter tuning, it became evident that despite extensive training over 1000 episodes in a four-obstacle environment, the bot exhibited limited learning progress. In light of this, a decision was made to leverage the most successful simulation results obtained from the two-obstacle environment as a foundation or checkpoint nodes for navigating the more complex four-obstacle environment.

Fig. 52. Distance plot

Reward Function:

$$r(S_t, a_t) = \begin{cases} 2000 & \text{if goal reached} \\ -1000 & \text{if collision} \\ 500 * dist\_rate + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate > 0 \\ -15 + 3 * angle\_rate & \\ -1000 * avoid\_rate & \text{if } dist\_rate < 0 \\ -250 & \text{goal not reach-} \\ & \text{-ed within } 40sec \end{cases}$$

Avoidance Rate: $avoid\_rate$ = 0.09 - $dist\_obs$
Noise Factor: σ = 0.025
Time limit to travel from one goal to another = 40 seconds

1) Result: The results can be seen in Fig 51,52 and 53. The stacked learning model clearly beats the model trained with random inital weights and biases in every category.

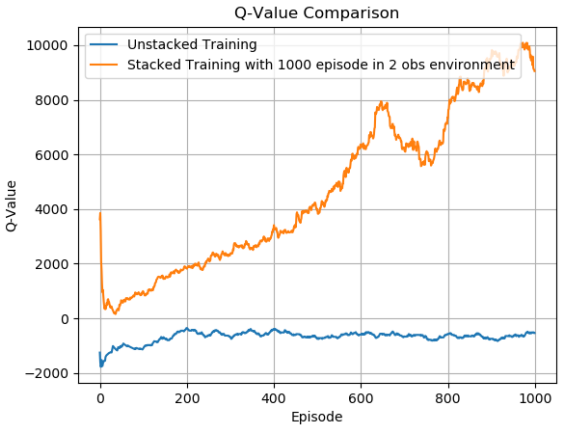

Fig. 51. Q-Value Plot

2) Conclusion: The outcomes of adopting a stacked learning approach have surpassed expectations, marking a significant leap in the bot's performance. With the incorporation of checkpoint nodes derived from the best-performing simula-

Fig. 53. Time plot

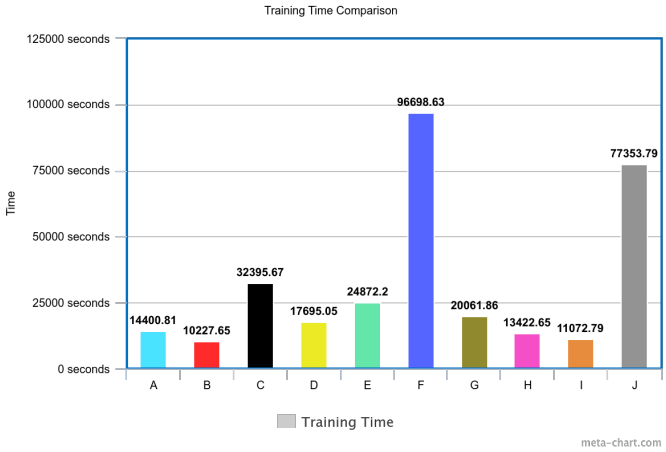

Fig. 54. Comparing the Total Training Time of all the subsections

tions in the two-obstacle environment, the bot has showcased remarkable progress in navigating the more complex four-obstacle configuration. Scores have surged to unprecedented heights, hovering around the 10000 mark, indicative of the substantial strides made in mastering intricate navigation challenges. This success underscores the profound benefits of stacking learning, whereby leveraging prior knowledge and accomplishments accelerates the bot's learning curve and enhances its adaptability to increasingly demanding environments. Such noteworthy advancements underscore the efficacy of strategic learning strategies in empowering autonomous systems to surmount complex real-world challenges with confidence and proficiency. Earlier attempts were made by us for stacked learning but they were only mild improvements and that might have been unable to produce this sort of a result as the noise factor was very high during their training leading to too much noise in the actions performed basically discarding all the knowledge that the checkpoint model used as a base had gained from before. The validation video of the simulation can be found here.

IX. Executing Theory: Deploying Code and Model Checkpoints for Real-World Evaluation

The deployment of the TurtleBot3 Waffle Pi marked a crucial milestone in our project, as it served as the physical embodiment of our trained neural network in a real-world setting. Upon setup and configuration, an unexpected discrepancy surfaced: the bot's LiDAR sensor outputted only 220 values instead of the expected 360, corresponding to the number of nodes in our neural network's input layer. To address this disparity, interpolation techniques were employed to fill in the missing values. Specifically, the missing values were approximated by calculating the average between two adjacent points, ensuring the preservation of spatial continuity in the LiDAR data.

Running of the neural network code with the TurtleBot3 Waffle Pi revealed operational challenges. While the number of LiDAR points now aligned with the neural network's input layer, the bot exhibited erratic behavior during runtime. At seemingly random intervals, the bot would abruptly halt, falsely indicating a collision despite no immediate obstacle presence. Amidst these interruptions, the bot still demonstrated an inclination towards goal-directed movement while cautiously navigating around visible obstacles.

This unexpected behavior prompted further investigation into the underlying causes of the sporadic stoppages and collision warnings. Despite the setbacks, the successful execution of the neural network code on the TurtleBot3 Waffle Pi underscored the potential for AI-driven navigation in real-world scenarios. The video of the turtlebot demo can be found here.

X. Conclusion

In conclusion, this research journey has provided invaluable insights into the intricate process of parameter tuning and real-world deployment of reinforcement learning algorithms for robotic navigation tasks. Through systematic experimentation and rigorous evaluation, we have successfully identified and optimized key parameters to develop a robust policy tailored to the complexities of our TurtleBot3 environment. All of the code for optimmized DDPG can be found here.

The simulated model, trained with the optimized parameters, yielded promising results, underscoring the efficacy of our approach in controlled environments. The weights for the trained model can be found here. However, the transition to real-world testing posed unforeseen challenges, necessitating adaptation and problem-solving in the face of sensor discrepancies and unexpected behavior.

Despite these challenges, our endeavors have yielded significant progress, demonstrating the potential of AI-driven navigation in practical applications. Each setback encountered serves as a valuable learning opportunity, informing future iterations and refining our approach towards the development of autonomous systems capable of navigating real-world environments with precision and reliability.

Moving forward, ongoing research endeavors hold promise for the continued advancement of robotics. By refining algorithms and addressing challenges inherent in real-world deployment, the field stands poised to unlock the full potential of intelligent and autonomous systems. Through perseverance and innovation, researchers are at the forefront of shaping a future where robotics revolutionize diverse industries and enhance human capabilities.

REFERENCES

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018.

[2] ROBOTIS. Turtlebot3 e-manual. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[3] ROBOTIS-GIT. turtlebot3_simulations. https://github.com/ROBOTIS-GIT/turtlebot3_simulations.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning:An Introduction*. Bradford, The MIT Press, Cambridge, Massachusetts, London, England, 2014, 2015. https://inst.eecs.berkeley.edu/~cs188/sp20/assets/files/SuttonBartoIPRLBook2ndEd.pdf.

[5] Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Daan Wierstra Timothy P. Lillicrap, Jonathan J. Hunt. Continuous control with deep reinforcement learning. pages 4,5, London, UK, 2016. Google Deepmind. https://arxiv.org/pdf/1509.02971.pdf.