Intelligent Robotic Navigation

Sai Jagadeesh Muralikrishnan
Robotics Graduate student
University of Maryland
College Park, Maryland
jagkrish@umd.edu

Varun Lakshmanan
Robotics Graduate student
University of Maryland
College Park, Maryland
varunl11@umd.edu

***Abstract*—The goal of this project is to improve robotic navigation with advanced deep reinforcement learning techniques in environments with dynamic obstacles. To enhance the performance of a TurtleBot Burger in the Gazebo simulation environment, we implement a Double DQN with a Dueling architecture, building upon the existing Deep Q-Network (DQN) algorithm. By addressing the overestimation bias and stability problems present in the original DQN, the modifications are intended to improve navigation efficiency and reduce the number of training episodes needed. The value and advantage functions are kept apart in the Dueling Double DQN, which improves policy updates and increases decision-making accuracy. We show significant gains in success rates, lower collision rates, and more stable reward trends over the vanilla DQN through extensive testing and analysis.**

***Index Terms*—DQN, Double DQN, Dueling DQN, ROS2, TurtleBot3**

I. Introduction

One of the most important problems in the field of autonomous systems is robotic navigation in dynamic environments. The goal of this project is to use cutting-edge deep reinforcement learning techniques to improve the TurtleBot Burger robot's navigational skills inside the Gazebo simulation environment. The main goal is to enhance the Deep Q-Network (DQN) algorithm, an online, off-policy, model-free reinforcement learning technique. Although DQN is well-known for its ability to calculate the cumulative long-term reward of actions in discrete action spaces, during training it experiences stability problems and overestimation bias.

We propose integrating Double DQN and Dueling architecture into the current DQN framework in order to overcome these difficulties. By breaking down the Q-value into state value and advantage functions, the Dueling architecture improves stability and learning efficiency, while Double DQN reduces the overestimation bias by employing separate networks for action selection and evaluation. With this method, the TurtleBot should be able to navigate through environments with dynamically moving obstacles more effectively, which should cut down on the amount of training episodes needed and enhance overall performance.

Our project involves applying these cutting-edge methods in a specially created TurtleBot Burger simulation environment, intended to evaluate the robustness and effectiveness of the updated algorithms. We compare the performance of the vanilla DQN with the proposed **Dueling Double DQN** through rigorous training and evaluation, showing notable gains in success rates, decreased collision rates, and more stable reward trends. This work provides a solid foundation for future advancements in autonomous navigation within constantly changing environments, in addition to advancing the state of robotic navigation algorithms.

II. Literature Review

With the introduction to our objectives in this project. We chose DQN over other predecessors like Q-learning algorithms because of its effectiveness in discrete action spaces, the Deep Q-Network (DQN) is one of the DRL algorithms that has been studied and used extensively. But because DQN is prone to instability and overestimation bias during training, researchers are looking into improved variants like Dueling DQN and Double DQN.

One of the key references for this project is a GitHub repository titled "A ROS2 framework for DRL autonomous navigation on mobile robots with LiDAR" by Tomas Vrana[1]. The framework, built using ROS2 and PyTorch, supports the training, storing, loading, and evaluation of navigation agents in both simulation and real-world environments.

Furthermore, Mohit Sewak's book "Deep Q Network (DQN), Double DQN, and Dueling DQN: A Step Towards General Artificial Intelligence" (2019)[2] made it feasible to comprehend the theoretical foundations of these advanced DQN versions. Sewak's work provides an in-depth analysis of how learning efficiency and performance are affected by Double DQN and Dueling DQN, along with a detailed explanation of how these techniques enhance the core DQN algorithm. This information was crucial to our project since it clarified the benefits and drawbacks of various techniques as well as how they might improve autonomous navigation.

Furthermore, the study by Ziyu Wang et al. (2016), titled "Dueling network architectures for deep reinforcement learning," presented at ICML 2016[3], introduces the Dueling DQN architecture. This work demonstrates how separating the estimation of state values and advantages can lead to better policy evaluation in deep reinforcement learning tasks. this helped us in the learning process in identification of limitations in the vanilla DQN.

Our effort intends to obtain greater performance in autonomous navigation with the TurtleBot Burger in dynamic environments by combining the theoretical insights from

Sewak's book Ziyu Wang;s paper along with the practical framework offered by the GitHub repository. It is anticipated that the improved algorithm will decrease training episodes and boost the robot's navigational effectiveness, increasing its adaptability in real-world situations.

III. Methodology

Algorithm 1 Deep Q-Network (DQN)

Input: Environment, rewards, states, neural network parameters, learning parameters, exploration parameters
Output: Trained network of DQN
Initialize single-stream DQN network that maps state inputs directly to Q-values for each action.
Create a target network for stable Q-value estimation.
Set the discount factor γ for future reward importance.
Set the learning rate for the optimizer and initialize ϵ for ϵ-greedy strategy.
for each episode **do**
    Select action with probability ϵ or compute Q-values to select higher Q-value action.
    Execute action, observe next state and reward, check episode end.
    Store transition in replay buffer.
    Sample a batch from the replay buffer.
    for each transition in batch **do**
        Predict Q-value of next state using target network.
        Calculate target Q-value.
        Compute current Q-values using primary network.
        Calculate mean squared loss between current and target Q-values.
        Backpropagate and update primary network weights.
        Update target network with weights from primary network.
    end for
    Reduce ϵ over predefined duration.
end for
Output the trained DQN network.

The methodology describes the research design, approach, and techniques used to conduct this study on enhancing autonomous navigation systems for mobile robots using advanced deep reinforcement learning (DRL) techniques. The following sections provide detailed information on data collection methods, tools, and procedures, ensuring that the study can be replicated and its validity assessed.

A. Design and Approach

This study's main objective is to enhance the TurtleBot Burger robot's navigational performance in map with dynamic obstacles. The strategy involves using Double DQN and Dueling DQN algorithms to modify the existing Deep Q-Network (DQN) algorithm which is used intially in navigation process. Gazebo is used to create a simulated environment for the study, while ROS2 is used as the middleware for communication and control of the robots.

1) Explanation of the Vanilla Deep Q-Network Algorithm: The Vanilla Deep Q-Network (DQN) algorithm is a fundamental reinforcement learning method used as a base for this study, which is used to learn how the framework works with the model to make the modification further the below mentioned explanation of the existing DQN algorithm as shown in Algorithm 1 is broke down for further clarification and to further discuss how it is been modified to better version that achieves higher success rate. In order to learn more about the modification, understanding the base algorithm is essential which is vanilla DQN in our case.

B. Vanilla Deep Q-Network (DQN) Algorithm Explanation

The DQN algorithm utilizes deep learning to approximate the optimal action-value function in a reinforcement learning setting. Below is a detailed step by step explanation for the algorithm 1, including the critical equation for updating the target Q-values and the loss function used during training.

Algorithm Steps:

1) **Input:** The algorithm needs the environment state, rewards, and various parameters, including neural network, learning, and exploration parameters.
2) **Initialization:** the neural network is initialized to map state inputs directly to Q-values for each possible action for the state. This network serves as the function approximator for the Q-function.
3) **Stable Q-value Estimation:** A secondary network, known as the target network, is created to estimate Q-values more stably during training updates.
4) **Discount Factor:** The discount factor γ is set to prioritize future rewards.This component affects how future state rewards are taken into account during the current decision-making process.
5) **Learning Rate and Exploration:** The learning rate for the optimizer is set first, along with the initialization of the exploration rate ϵ for the ϵ-greedy strategy. This strategy balances the exploration of new actions aolong with the exploitation of known information.
6) **Training Process:** For each episode:
    a) **Action Selection:** An action is chosen either randomly with probability ϵ or by selecting the action with the highest Q-value predicted by the network.
    b) **Execution and Observation:** The chosen action is executed, and the agent observes the next state and reward after this. The episode's end is also checked together.
    c) **Replay Buffer:** The all transition (state, action, reward, next state) is stored in the replay buffer, which helps to break the correlation between sequential observations.
    d) **Batch Update:** A batch of transitions is randomly sampled from the replay buffer for training.
        i) Predict the Q-value for the next states using the target network.

ii) **Target Q-value Calculation:** The target Q-values are calculated using the formula below:

$$Q_{\text{target}} = r + \gamma \max_{a'} Q_{\text{target}}(s', a') \cdot (1 - \text{done})$$

iii) Compute the Q-values for the current states and actions using the primary network.

iv) **Loss Calculation:** Calculated the loss between the current and target Q-values using the Mean Squared Error (MSE) Loss formula:

$$\text{LOSS} = \frac{1}{N} \sum_{i=1}^{N} (Q_{\text{predicted},i} - Q_{\text{target},i})^2$$

v) Backpropagate the loss through the network and update the weights before next step.

e) **Update Target Network:** Periodically, the weights from the primary network are copied to the target network for stabilizing the learning process.

7) **Exploration Reduction:** The exploration rate ϵ is gradually reduced over time to shift from exploring the environment to exploiting learned behaviors in the map.
8) **Model Output:** The trained network capable of estimating Q-values for given states is output which represents the learned policy.

C. Dueling Double Deep Q-Network (Dueling Double DQN) Algorithm

This section explains the steps of the Dueling DQN algorithm and the integration of the Double DQN approach within it, which shown in the Algorithm 2.

***Algorithm Steps**:*

1) **Initialization:** The architecture initializes with two separate streams within the neural network. One stream estimates the state value $V(s)$, and the other calculates the advantages for each action $A(s, a)$.
2) **Combining Outputs:** The Q-values for each action are computed by combining the state value and the advantages of each action. Specifically, the Q-value for a state-action pair (s, a) is given by:

$$Q(s, a) = V(s) + (A(s, a) - \text{mean}(A(s, a)))$$

where the mean subtraction stabilizes the learning by centering the advantage values.
3) **Target Network Creation:** A target network, which is a clone of the primary network, is created to provide stable Q-value estimates during learning. The weights of the target network are updated less frequently to maintain stability.
4) **Learning Process:** In each episode, actions are selected using an ϵ-greedy policy, balancing between exploration and exploitation. The chosen action is executed, and the resulting new state and reward are observed.
5) **Experience Replay:** Transitions consisting of the current state, action, reward, and next state are stored in a replay buffer. Batches of these transitions are sampled randomly to update the network weights, reducing correlations between updates.
6) **Batch Learning:** For each sampled transition, the primary network predicts the best action for the next state and the target network provides the Q-value estimate for this action, which serves as the target for updates.
7) **Loss Calculation and Backpropagation:** The loss is computed as the mean squared error as shown before between the predicted Q-values and the target Q-values. This loss is then backpropagated through the primary network to update its weights.
8) **Network Synchronization:** Periodically, the weights of the primary network are copied to the target network to synchronize them.
9) **Exploration Reduction:** The exploration rate ϵ is reduced over time to encourage exploitation of the learned

Algorithm 2 Dueling Deep Q-Network (Dueling DQN)

Input: Environment, rewards, states, neural network parameters, learning parameters, exploration parameters
Output: Trained network Dueling DQN
Initialize dual-stream neural network DQN with random weights.
One stream predicts state value $V(s)$.
Other stream predicts advantage of each action $A(s, a)$.
Combine outputs to compute Q-values: $Q(s, a) = V(s) + (A(s, a) - \text{mean}(A(s, a)))$
Create a target network for stable Q-value estimation.
Set the discount factor γ.
Set the learning rate and initialize ϵ for ϵ-greedy strategy.
for each episode **do**
    Select action with probability ϵ based on Q-values from primary network.
    Execute action, observe next state and reward, check episode end.
    Store transition in replay buffer.
    Sample a batch from the replay buffer.
    for each transition in batch **do**
        Use primary network to select best action for next state.
        Use target network to calculate Q-value for chosen action.
        Combine streams to compute target Q-values.
        Calculate current Q-values.
        Calculate mean squared loss between current and target Q-values.
        Backpropagate and update primary network weights.
        Update target network with weights from primary network.
    end for
    Reduce ϵ over predefined duration.
end for
Output the trained Dueling DQN network.

policy over exploration.

10) **Output:** The trained network, optimized for selecting actions in given states in every episode, is output after the training process finishes.

This procedure demonstrates how the Dueling DQN algorithm improves learning by modeling the benefits of actions and the state value independently. This method works especially well in environments where there are a lot of actions with values that are similar..

*1) **Robot Model**:*

- **TurtleBot Burger:** The robot used for this study, equipped with LiDAR sensors and odometry capabilities, providing the necessary data for DRL training.

D. Tools and Procedures

*1) **Framework**:*

- **A ROS2 Framework for DRL Autonomous Navigation:** The GitHub repository by Tomas Vrana is utilized as the base framework. This framework supports the development and experimentation with DRL algorithms for autonomous navigation on mobile robots where custom made algorithms can be tested in custom made different worlds.

2) Libraries Used:

- **Numpy:** For numerical operations.
- **PyTorch:** Used for building neural networks.

E. Set up of the Simulation Environment

1) ROS2 Foxy and Gazebo is used on an Ubuntu system.
2) Configured the TurtleBot Burger model with LiDAR and odometry sensors for better search of goal.
3) Deploying the custom world with dynamic obstacles in Gazebo.

F. Implementation and Training of the Algorithms

1) Cloned the GitHub repository "A ROS2 framework for DRL autonomous navigation on mobile robots with LiDAR." for initializing the gazebo world that is compatible with ROS2 for the simulations in which the custom based moving obstacles can be added and placed inside the world in which the TutleBot3 Burger is spawned for the training process
2) Implemented the Double DQN, and Dueling DQN algorithms proposed as shown in Algorithm 2. But in order to compare the training process and study the performance after testing we used the existing vanilla DQN algorithm as shown in the Algorithm 1.
3) Using the data gathered from the simulation environment, the models were trained, and if necessary, the hyperparameters were adjusted for optimal performance. However, the study's main goal is to compare the performance of the updated algorithm with the original one to achieve the navigation process.

G. Evaluation of the Models

1) Tested the trained models, which have been trained more than 1700 or more episodes in phase 1 which showed the actual progress in order to compare the both algorithms while visualizing the Burger bot moves towards goal while avoiding the dynamic obstacles in the simulation environment.
2) Collected performance metrics after the fixed number of episodes while testing and compare the results to validate the enhancements between the DQN and Dueling Double DQN.

The detailed contributions and results of implementing Dueling Double DQN are explored extensively in the contributions section and result section of this report.

IV. Contributions

During the early stages of our study to asses the problems faced by the vanilla DQN algorithm which is used by the author Tomas Vrana as the base DRL algorithm in the GitHub repository, and we found that the Vanilla Deep Q-Network (DQN) algorithm exposed a number of serious flaws after testing, especially when dealing with dynamic obstacles.

A. Problems faced with the vanilla DQN

The Vanilla Deep Q-Network (DQN) algorithm encounters several significant challenges during its application in reinforcement learning environments. The main problems were:

- **Overestimation Bias:** Vanilla DQN suffers from overestimation bias where it tends to overestimate Q-values due to the usage of the max operator, which always chooses the higher Q-values. This sometimes may not reflect the true values of the state-action pair accurately, leading to suboptimal policies.
- **Significant Instability During Training:** This issue frequently results in the creation of policies that are not optimum and in slow rates of convergence. These difficulties make it far more challenging for the robot to learn efficient navigational techniques.

By applying a double Q-learning algorithm to reduce overestimation errors and a dueling network design to evaluate state values and action advantages independently, this sophisticated approach alters the conventional DQN. These improvements are essential to achieving learning stability, estimation accuracy and enhancing the performance and reliability of autonomous navigation systems.

B. Contributions to Vanilla DQN with Dueling Double DQN Algorithms

In comparison to the conventional Vanilla DQN, the Dueling Double DQN algorithm is a major improvement in a number of important areas. This section shows our contributions to the project by comparing the two algorithms and highlighting the improvements incorporated into the Dueling Double DQN.

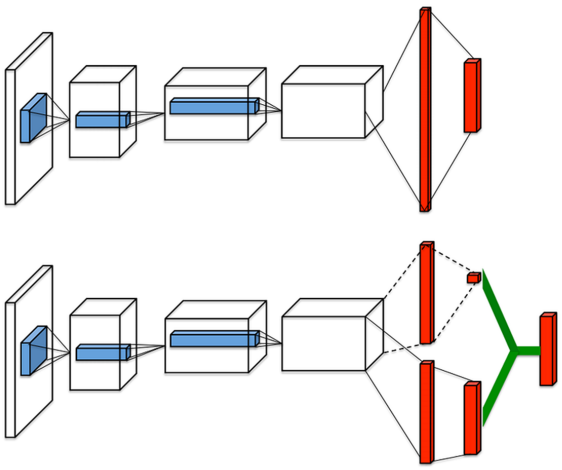

Fig. 1. Image from the Dueling DQN paper[3]. Top: Regular DQN architecture. Bottom: Dueling DQN architecture.

1) Vanilla DQN: With the Vanilla DQN technique, state inputs are directly mapped to Q-values for every potential action using a single neural network. The essential actions consist of:

- Initializing a neural network with random weights.
- Using an ϵ-greedy policy for action selection to balance the both exploration and exploitation.
- Updating the network weights based on the loss calculated from the difference between predicted Q-values and target Q-values, which are computed using a separate target network for stabilizing training.

While effective, Vanilla DQN suffers from significant overestimation of Q-values and can be slow to converge, particularly in environments with complex or similar-valued actions.

2) Dueling Double DQN: Two major improvements are added to the Vanilla DQN design by the Dueling Double DQN algorithm: a dueling network architecture, the Double DQN method, and an alteration to the activation function. Among these modifications are:

- **Dueling Network Architecture:** This architecture uses two separate streams within the neural network before the output layer like shown in **Fig. 1** bottom part:
  1) One stream estimates the value of each state $V(s)$.
  2) Another estimates the advantage of each action $A(s, a)$.

  The final Q-values are calculated by combining these streams using the formula shown below:

$$Q(s, a) = V(s) + (A(s, a) - \text{mean}(A(s, a)))$$

  With this method, the network can determine which states are useful (or not) without regard to the action executed, leading to more accurate value assessments.
- **Double DQN Method:** To address the overestimation bias seen in Vanilla DQN, the Double DQN approach decouples the action selection from the Q-value evaluation and does it seperately:
  1) The action is selected using the primary network based on the highest Q-value.
  2) The Q-value of this selected action is evaluated using the target network.

  This decoupling significantly reduces the risk of overestimation bias. The **modified Bellman equation** used in Double DQN is as follows:

$$Q(s, a; \theta) = r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta')$$

  This method ensures that the selection of the best action is based on the primary network, while the evaluation of its potential reward is based on the target network, thereby reducing overestimation biases and leading to more stable and reliable learning outcomes.
  This separation reduces overestimation by preventing the same network from both selecting and evaluating an action.
- **Activation Function Update:** To prevent the issue of dying neurons often associated with the **ReLU** activation function, the network employs the Exponential Linear Unit (ELU) activation function. **ELU** helps in maintaining neuron activity over time, ensuring that negative inputs can still contribute to the learning process, enhancing the network's ability to generalize from the input data.
- **Additional Layers:** In our enhancements to the traditional Dueling DQN architecture, one of the significant changes was the addition of layers to the neural network. This modification is aimed at increasing the network's capacity to learn and adapt to more complex environments. Originally, the Vanilla DQN consisted of **two linear layers**. These layers are sufficient for simpler scenarios but might lack the necessary depth for capturing complex relationships in the data in more challenging environments. Where, in the **Enhanced Dueling DQN Architecture** we extended the architecture by adding two additional linear layers, making a total of four linear layers.

C. Advantages achieved over vanilla DQN

- **Enhanced Stability:** The method delivers more consistent and predictable training outcomes by separating action selection from value estimate, which may lead to improved success rates for the Burger bot in completing the task without encountering any obstacles.
- **Improved Efficiency:** More sophisticated updates to the Q-values are possible due to the separation of the state value and action advantage calculations, which speeds up learning and improves convergence rates while lowering loss and raising rewards.
- **Better Policy Development:** The refined architecture facilitates a deeper understanding of the environment dynamics, leading to the development of superior decision-making policies.

In addition to mitigating the drawbacks of the Vanilla DQN, these improvements raise the success rate, boost overall learning process speed and reliability, and lower the likelihood that the TurtleBot bot would encounter obstacles. We further move to the analysis of the performance between the new algorithm versus the old algorithm in the experiments and results section using the performance metric derived after training.

V. Experimental Results

A. Experimental Setup

Initially we started to study the frameworks used from the repository to setup the ROS2 foxy version in the Ubuntu 20.04 along with the latest version of Gazebo. Later we tried using different plugins for making the obstacles to move in both random and predefined path, but we used randomly moving obstacles for the training and testing at the end, we deployed the TurtleBot3 stage 4 world which is the suitable version for adding mix of dynamic obstacles with presence of inner walls which serves as the static obstacles for the Burger bot to have better training process.

Once the setup is done, the framework of the github repository is adjusted so it can accommodate our modified Dueling Double algorithm alongside in the same package where the vanilla DQN exist for ease to switch between the algorithms to test at the end of different episodes intervals. We used Burger bot with LiDAR and odometery topics subscribed and also we adjusted the thresholds for the LiDAR range for it to detect precisely once it approaches the boundaries or the obstacles. Also the goal points are made to spawn randomly at each episode inside the walls defined for the robot to move towards it.

Initially we split the training and testing into 2 phases, where the initial phase is trained till 1700 episode to verify the correctness of the modified algorithm, while in the phase -2 we asses the models after training more than 3000 episodes. Once Phase two is complete we tested the performance metric and fairly comparison has been drafted and checked again.

B. Training Phase - 1

During the first phase of training, we aimed to train the models from 0 to 1700 episodes to assess their performance. The training process began with the Vanilla DQN, which took approximately **15 hours** to complete. The extended training duration was primarily due to the training without GPU which cause multiple issues which is discussed in problems faced section later, another reason is complexity of the map, which featured numerous dynamic obstacles, and the frequent timeouts encountered. These timeouts occurred when the robot took more than 50 seconds to locate the goal point within the map.

On the other hand, the Dueling Double DQN model took less than **14 hours** to finish the training. During this phase, there were a lot of collisions and timeouts for both models at first.

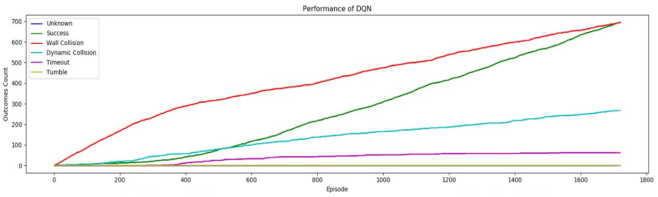

Fig. 2. Performance of vanilla DQN with non-efficient results after training of 1700 episodes

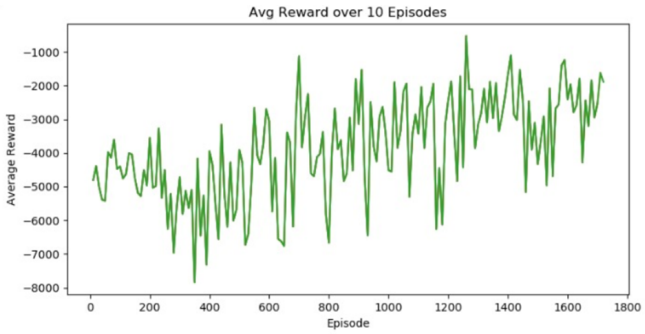

Fig. 3. Average rewards over every ten episodes of vanilla DQN at phase 1

1) Comparison of Model Performances after 1700 episodes: **Vanilla DQN Performance:** The performance of the Vanilla DQN model showed moderate results through the the training at 1700 episodes. The model was less efficient compared to the Dueling method, with the robot's success rate(green line) in reaching the goal only gradually increasing. Additionally, the robot encountered frequent collisions with walls(red line) and dynamic obstacles which could be observed from **Fig 2**. Moreover, the average rewards over every 10 episodes remained below zero as shown in **Fig. 3**, indicating a negative performance trend.

Dueling Double DQN Performance: In contrast, the Dueling Double DQN model demonstrated a steep improvement in performance. The success rate(green line) of the robot in reaching the goal steadily increased, and the number of collisions(red line) significantly decreased by the end of 1400 episodes. By 1700 episodes which can be noticed in the **Fig. 4**, along side the rewards already progressed to the positive values after 1000 episodes itself which is shown in **Fig. 5**. The Vanilla DQN still could not match the performance levels achieved by the Dueling Double DQN.

These observations underscore the enhanced efficiency and capability of the Dueling Double DQN model in navigating complex environments with higher precision and fewer errors, as evidenced by the positive trends in success rates and reduced collision incidents. Moreover, to view the video results of the Phase - 1 training simulation follow the drive link https://drive.google.com/file/d/1qH-McJhEq93v7Rdqwxw6BZ8j4bVdsPM-/view

TABLE I
TESTING RESULTS FOR BOTH MODELS OVER 20 EPISODES AFTER 3000 EPISODES OF TRAINING

	Dueling Double DQN				Vanilla DQN			
Episode	Success	Wall Collision	Obstacle Collision	Timeout	Success	Wall Collision	Obstacle Collision	Timeout
3001 to 3005	3	1	1	0	2	3	0	0
3006 to 3010	3	1	1	0	1	0	1	3
3011 to 3015	2	2	1	0	0	3	2	0
3016 to 3020	2	2	1	0	2	2	1	0
Total	10	6	4	0	5	8	4	3
Total Percentage	50%	30%	20%	0%	25%	40%	20%	15%

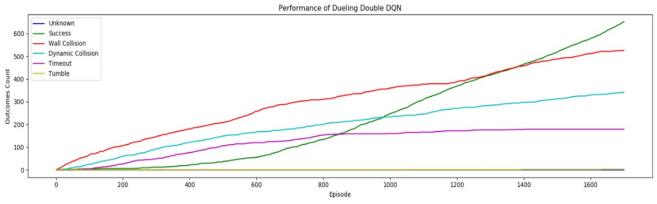

Fig. 4. Performance of Dueling Double DQN efficient results after training of 1700 episodes

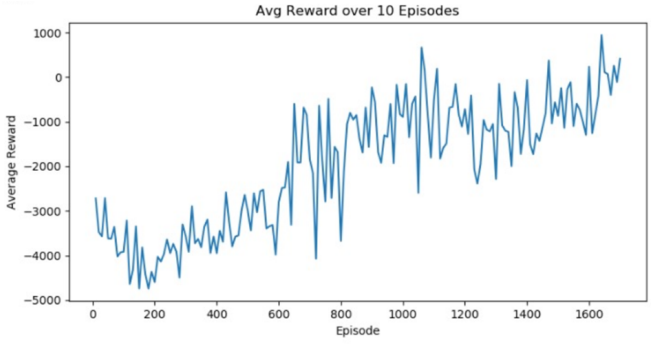

Fig. 5. Average rewards over every ten episodes of Dueling Double DQN at phase 1

C. Training Phase - 2

Both the Vanilla DQN and Dueling Double DQN models were reset and started from episode 0 to verify the performance observed in Phase 1 and to acquire fresh results by the end of Phase 2. The training time for Vanilla DQN from the 0th episode to the 3000th episode took approximately 28 hours, while Dueling Double DQN completed the training in about 26 hours. This phase was designed to test the model to check performance after 3000 episodes of training.

D. Comparison After 3000 Episodes

- **Vanilla DQN:** The performance of the Vanilla DQN, as observed from **Fig. 6**, showed that the success rate started increasing gradually above the collision rate, but the increase was not significant. After 1700 episodes, there was a dramatic increase in success, but the collision rate with the robot also increased. The average rewards after 3000 episodes for Vanilla DQN remained below zero, in the negative side, still not matching the performance of the Phase 1 Dueling Double DQN model.
- **Dueling Double DQN:** This model performed beyond expectations where the collision rate drastically decreased while the success rate significantly increased. This can be observed in **Fig. 8**, where the success rate (green line) increased steeply, while the collision rate (red line) decreased. Additionally, the rewards were maintained above zero with positive values, as observed from **Fig. 9**.

Furthermore, to view the video results of the phase - 2 training simulation that over 3000 episodes for the both models follow the drive link https://drive.google.com/file/d/1tsFBgujWl74BMfZTGFvLvy3U1bwCoKbR/view

E. Testing After 3000 Episodes

Both models were tested over the next 20 episodes in a more complex environment than what they were trained on previously. This new environment featured 6 moving obstacles instead of 4, with increased space inside the map to accommodate the additional dynamic obstacles and the burger bot.

- **Vanilla DQN:** During testing, the burger bot reached the goal points but not consistently, reflecting a lower success rate as it collided with walls and dynamically moving obstacles most of the time. This also led to an increased rate of timeouts.
- **Dueling Double DQN:** After training, the robot effectively navigated to randomly spawning goal points in every episode, occasionally hitting dynamic obstacles but rarely colliding with wall obstacles and experiencing very few timeouts over the 20 episodes tested. This performance underscores the robustness and reliability of the Dueling Double DQN model.

To view the video results of the testing simulations and observations follow the drive link https://drive.google.com/file/d/12PErRUwAVW8FRP7aUm8Ux1sWXIeSSM-0/view

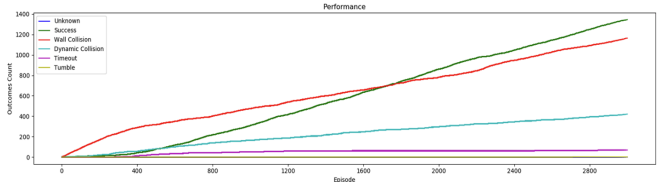

Fig. 6. Performance of Vanilla DQN after 3000 episodes

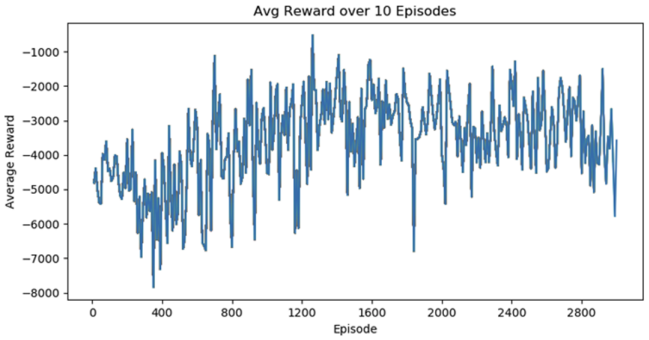

Fig. 7. Average rewards over every ten episodes of vanilla DQN at phase 2

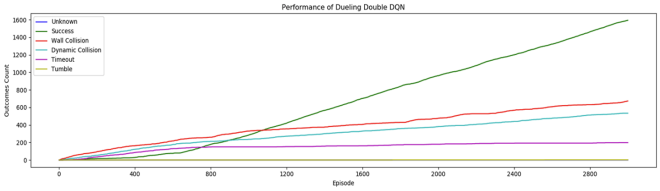

Fig. 8. Performance of Dueling Double DQN after 3000 episodes

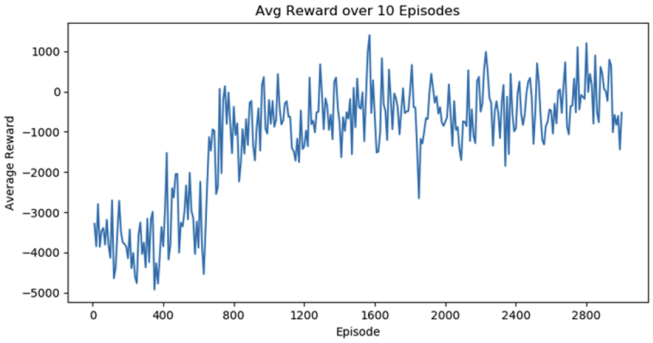

Fig. 9. Average rewards over every ten episodes of Dueling Double DQN at phase -2

TABLE II
TOTAL STEPS TAKEN BY THE ROBOT WHILE TESTING FROM 3001ST TO 3020TH EPISODE AND THE AVERAGE STEPS FOR 20 ITERATIONS.

Episodes from 3001	Dueling DQN	Vanilla DQN
Total steps from 3001 to 3005	4,692	7,259
Total steps from 3001 to 3010	10,342	22,619
Total steps from 3001 to 3015	15,605	31,358
Total steps from 3001 to 3020	19,015	35,239

F. Analysis of Table I

As shown in Table I, the testing results from Phase 2 indicate significant differences in performance between the Dueling Double DQN and Vanilla DQN models. The Dueling Double DQN model demonstrates higher success rates with fewer collisions and no timeouts across the evaluated episodes, underscoring its enhanced capability and robustness in handling complex environments with dynamic obstacles. In contrast, the Vanilla DQN model exhibits a higher rate of collisions and timeouts, with a slower improvement in success rates.

This disparity highlights the effectiveness of the Dueling architecture in improving decision-making accuracy and operational efficiency, as the Dueling Double DQN model not only learns faster but also adapts better to complex scenarios, maintaining positive reward outcomes more consistently than the Vanilla model.

G. Analysis of Table 2

The Table 2 shows the number of steps took by the robot to reach the random goal point from the spawn point. For simplification the total steps have been added for every 5 iterations. Fewer steps shown by the Dueling Double DQN indicate that it performs more efficiently than the vanilla DQN. The results in the Table 2 suggests that the agent of Dueling Double DQN is reaching its goals more quickly or solving the episodes with less trial and error. Because Dueling Double DQN is better at evaluating which actions lead to higher long-term rewards by separately estimating state value and the advantages of each action.

VI. EXTENDED TRAINING AND TESTING WITH FINAL PERFORMANCE ANALYSIS

As the performance metrics from the initial phases demonstrated the efficiency and robustness of the Dueling Double DQN algorithm, we decided to extend the training for only Dueling Double DQN by an additional 2000 episodes, bringing the total to 5000 episodes. This decision was made to further refine the TurtleBot's ability to navigate complex environments while efficiently avoiding both moving and static obstacles.

A. Outcomes of Extended Testing

This extensive testing phase produced positive results. The effectiveness of the Dueling Double DQN model to reduce collisions and increase successful navigation to the objective was

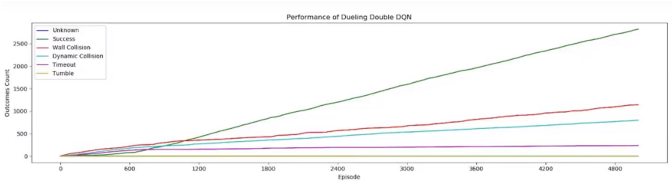

Fig. 10. Final Test performance of Dueling Double DQN after 5000 episodes

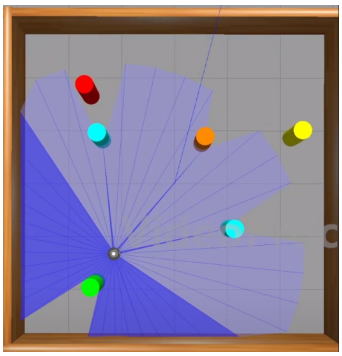

Fig. 11. Custom world used for testing after 5000 episodes

noticeably improved. In particular, the success rate increased to 75%, demonstrating the algorithm's adaptability and performance even in the face of increasing map complexity. These gains are evident in the performance after 5000 episodes, as seen in Fig. 10.

To view the video results of the extended testing of Dueling Double DQN follow the drive link https://drive.google.com/file/d/1NFG_7DQoTRUQjtM-QYpQCgTMvu_C78V3/view

These findings confirm that the enhancements made to the Dueling Double DQN algorithm have not only met but exceeded our initial expectations, which showed more optimal performance than what it was before with the vanilla DQN.

VII. Problems Faced

During the entire process, from initializing the world in Gazebo to conducting the testing analysis, we encountered several challenges that required more time than initially planned. These issues not only delayed our progress but also provided significant learning opportunities. Here, we detail the major problems faced and the steps taken to address them:

1) **GPU Training Interruptions:** Our initial attempts to train the model using the GPU were met with unexpected system issues, causing the training to stop before completion. This was particularly problematic as it occurred in the middle of the planned number of episodes, disrupting our training . Eventually, we decided to proceed without the help of the GPU. Although this approach increased the training duration, it enabled us to achieve the planned outcomes without further interruptions.
2) **Debugging the Dueling Double DQN:** Debugging the modified algorithm, the Dueling Double DQN, proved to be a toughest task. Each debugging session after every 1000 episodes was time-consuming and required to examine closely the algorithm's performance and its interactions within the environment. These debugging efforts were important in refining the model and ensuring its robustness.

VIII. Conclusion

This project set out with the aim of enhancing the navigational capabilities of autonomous robots in a dynamic obstacle based environment using a modified and refined version of the Deep Q-Network (DQN) algorithm. By addressing the limitations of the Vanilla DQN, such as overestimation bias and training instability, we have successfully demonstrated the effectiveness of a more sophisticated algorithmic approach with Dueling Double DQN.

The Dueling Double DQN algorithm, with its advanced architecture including additional layers and an updated activation function (Exponential Linear Unit, ELU), has shown significant improvements over the traditional Vanilla DQN model. The introduction of these enhancements facilitated better learning efficiency and decision-making accuracy, which are evident from the extensive testing and performance evaluations conducted.

A. Performance Outcomes

The modified algorithm outperformed the original in dynamic testing environments.The extended testing phase, which included up to 5000 episodes, further validated the robustness and reliability of the Dueling Double DQN. The results clearly indicated an 75% improvement in success rates and a substantial decrease in collision rates, which shows the superior performance of the modified algorithm in achieving the effective navigation.

In conclusion, the enhancements made to the DQN algorithm have not only fulfilled but exceeded the initial objectives of this project, providing a strong foundation for further advancements in the field of robotic navigation. The Dueling Double DQN stands as a testament to the potential of targeted modifications to significantly uplift the performance of existing algorithms in practical applications.

References

[1] T. Vrana, "A ROS2 Framework for DRL Autonomous Navigation on Mobile Robots with LiDAR," GitHub repository, Access Year. Available: https://github.com/username/repository
[2] M. Sewak, "Deep Q Network (DQN), Double DQN, and Dueling DQN: A Step Towards General Artificial Intelligence," 2019.
[3] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in Proceedings of the 33rd International Conference on Machine Learning - Volume 48, ICML'16, pp. 1995–2003, 2016.
[4] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, pp. 2094–2100, 2016.