Autonomous Navigation with Deep Reinforcement Learning and Dynamic Obstacle Avoidance

Joseph Thomas
UID: 119399545
joseph10@umd.edu

Rishikesh Jadhav
UID: 119256534
rjadhav1@umd.edu

Abstract

This report presents an investigation into autonomous vehicle navigation using reinforcement learning (RL) techniques, specifically focusing on Deep Q-Networks (DQN) and Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithms. The study was conducted using the TurtleBot3 robot in a simulated ROS Gazebo environment. The primary objective was to train the TurtleBot3 to navigate autonomously through an environment while avoiding obstacles. The research involved developing and implementing DQN and TD3 algorithms to enable the robot to learn optimal navigation policies through trial and error. The performance of both algorithms was evaluated and compared based on several criteria, including convergence speed, robustness to environmental changes, and overall navigation efficiency. Results indicate that while both algorithms successfully trained the robot to navigate autonomously, they exhibited distinct characteristics in terms of learning efficiency and stability. DQN demonstrated quicker initial learning but showed limitations in handling more complex navigation scenarios. In contrast, TD3 provided more stable and consistent performance, particularly in dynamic and challenging environments. This comparative analysis highlights the strengths and weaknesses of each algorithm, providing insights into their applicability for different autonomous navigation tasks. The findings of this study contribute to the broader understanding of reinforcement learning applications in robotics, offering guidance for future research and development in autonomous vehicle navigation.

1. Introduction

Autonomous vehicle navigation has become a pivotal area of research in the field of robotics, driven by the potential to enhance safety, efficiency, and accessibility in various domains including transportation, logistics, and personal robotics. The ability of a robot to navigate autonomously in an environment involves complex decision-making processes that require robust perception, planning, and control mechanisms. Among the various approaches to tackle this challenge, reinforcement learning (RL) has emerged as a promising technique due to its capability to enable agents to learn optimal behaviors through interactions with their environment.

Reinforcement learning involves training an agent to make a sequence of decisions by rewarding it for desirable actions and penalizing it for undesirable ones. This learning paradigm is particularly well-suited for autonomous navigation tasks, where the robot must continuously adapt to its surroundings and make real-time decisions to reach its destination while avoiding obstacles. In this study, we explore the application of two prominent RL algorithms, Deep Q-Networks (DQN) and Twin Delayed Deep Deterministic Policy Gradient (TD3), for the navigation of a TurtleBot3 robot in a simulated ROS Gazebo environment.

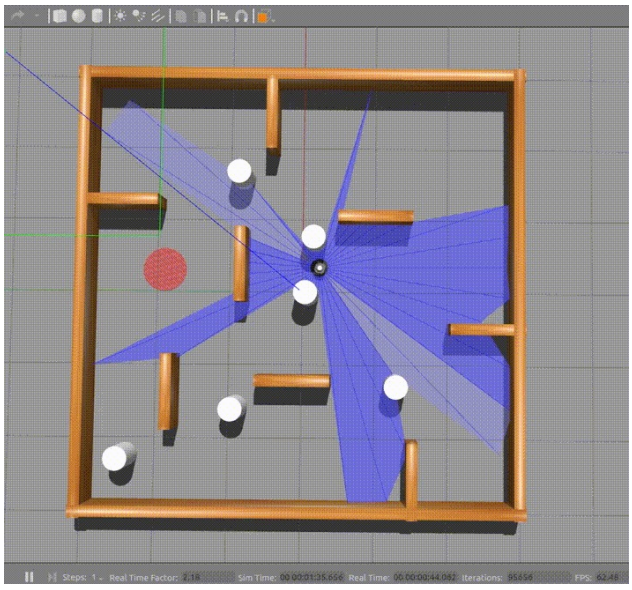

Figure 1: Turtlebot in the Gazebo Environment with dynamic obstacles

The Deep Q-Network (DQN) algorithm combines Q-learning with deep neural networks, enabling the agent to handle high-dimensional state spaces by approximating the Q-value function. This approach has shown significant success in various domains, including gaming and robotics. However, DQN is known to suffer from instability and inefficiency in learning, especially in continuous action spaces.

To address some of the limitations of DQN, the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm was developed. TD3 is an actor-critic method designed for continuous action spaces, incorporating strategies to reduce overestimation bias and improve learning stability. By comparing DQN and TD3, this study aims to provide a comprehensive analysis of their performance in autonomous navigation tasks.

The objective of this report is to detail the methodology, implementation, and comparative analysis of DQN and TD3 for training the TurtleBot3 to navigate autonomously. We begin with a review of related work in the field of autonomous navigation and reinforcement learning, followed by a detailed description of the simulation environment and the RL algorithms used. The results section presents the performance metrics and qualitative observations, leading to a discussion on the strengths and weaknesses of each algorithm. Finally, we conclude with insights gained from the study and potential directions for future research.

By systematically analyzing the application of DQN and TD3 in a simulated autonomous navigation task, this report contributes to the growing body of knowledge on reinforcement learning in robotics and provides valuable guidance for researchers and practitioners in the field.

2. Literature Survey

The application of reinforcement learning (RL) in autonomous vehicle navigation has gained significant traction due to its ability to handle complex and dynamic environments. Several research efforts have explored various RL algorithms and their effectiveness in different scenarios.

Deep Reinforcement Learning for Autonomous Navigation: Nan Yan, Subin Huang, and Chao Kong proposed a deep reinforcement learning-based method for autonomous navigation and obstacle avoidance in unmanned surface vehicles (USVs) under partially observable conditions. Their approach utilizes a double Q-network to achieve end-to-end control from raw sensor inputs to discrete rudder actions. They also employ long short-term memory (LSTM) networks to enhance the vehicle's ability to remember and react to environmental changes. The experiments demonstrated that their method ensures optimal path planning and collision avoidance in complex ocean environments, outperforming traditional DQN and random control policies in terms of convergence speed, sailing distance, and rudder angle steering consumption.

Reinforcement Learning-Based Obstacle Avoidance: C.S. Arvind and J. Senthilnath explored the use of reinforcement learning for obstacle detection and avoidance in autonomous vehicles. Their research compared conventional Q-learning with a Q-learning approach enhanced by a multi-layer perceptron neural network (MLP-NN). The experimental results indicated that the latter significantly improved the accuracy of obstacle prediction and vehicle navigation. The integration of MLP-NN addressed the continuous action space problem, providing a more robust solution for static obstacle detection in urban scenarios.

Autonomous Navigation in Complex Environments: The work of B. B. Elallid and colleagues focused on applying deep reinforcement learning for vehicle control in autonomous driving, particularly for intersection management without relying on vehicle-to-vehicle (V2V) communication or centralized systems. They utilized the Twin-Delayed DDPG (TD3) algorithm, which incorporates enhancements such as twin Q-networks and delayed policy updates, to navigate intersections safely and efficiently. Their simulations, conducted using the CARLA simulator, demonstrated the model's capability to reduce travel delay and collision rates.

Comparative Studies and Applications: A comprehensive survey by B. B. Elallid et al. reviewed various deep and reinforcement learning approaches in autonomous driving. Their analysis highlighted the potential of RL techniques to optimize navigation and obstacle avoidance in dynamic environments. The survey also underscored the importance of combining RL with other methods, such as supervised learning and sensor fusion, to enhance the robustness and reliability of autonomous systems.

Another significant contribution is the development of RL-based methods for mobile robot navigation. Xue et al. introduced a deep reinforcement learning method based on Double DQN for collision avoidance. Their approach demonstrated substantial improvements in navigating complex environments and avoiding obstacles compared to traditional methods.

Overall, these studies illustrate the diverse applications and effectiveness of reinforcement learning in autonomous vehicle navigation. The integration of advanced neural networks, such as LSTM and MLP, with RL algorithms like DQN and TD3, has proven to be a successful strategy for addressing the challenges of dynamic and uncertain environments.

3. Methodology

The methodology employed to achieve autonomous vehicle navigation using reinforcement learning (RL) algorithms, specifically Deep Q-Networks (DQN) and Twin Delayed Deep Deterministic Policy Gradient (TD3), is detailed in the following sections. The TurtleBot3 robot was simu-

lated in a ROS2 Foxy and Gazebo environment. The subsequent subsections describe the setup, implementation, and training processes involved in this project.

3.1. Environment Setup

The project utilized Ubuntu 20.04 LTS with ROS2 Foxy Fitzroy and Gazebo 11.0 for the simulation environment. The necessary dependencies, including PyTorch (version 1.10.0), were installed to support the RL algorithms. The simulation environment was configured to include obstacles and goals for the TurtleBot3 to navigate.

3.2. Algorithm Implementation

3.2.1 Deep Q-Network (DQN)

DQN is a model-free, off-policy RL algorithm that approximates the Q-value function using a deep neural network. The Q-network receives the current state, s_t, as input and outputs Q-values for all possible actions, a_t. The action with the highest Q-value is selected using an ϵ-greedy policy.

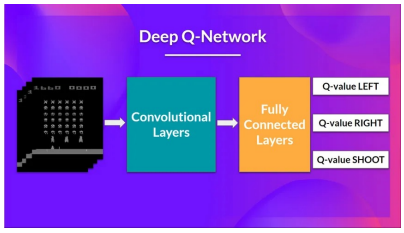

Figure 2: DQN Architecture Diagram

3.2.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is an actor-critic algorithm designed for continuous action spaces. It extends the Deep Deterministic Policy Gradient (DDPG) by addressing overestimation bias and improving learning stability. TD3 uses twin Q-networks to reduce overestimation and delayed policy updates to stabilize training.

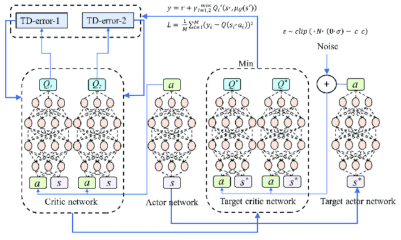

Figure 3: TD3 Architecture Diagram

3.3. Enhancements and Hyperparameter Tuning

Several enhancements and hyperparameter tuning techniques were employed to improve the performance and stability of both DQN and TD3 algorithms:

- **Learning Rate Scheduler:** A learning rate scheduler was incorporated to adjust the learning rate dynamically during training, helping to converge to an optimal solution more efficiently.
- **Batch Normalization:** Batch normalization layers were added to the neural networks to stabilize and accelerate training by normalizing the inputs of each layer.
- **Epsilon Decay (for DQN):** The ϵ-greedy policy was employed, where ϵ (the probability of choosing a random action) decays over time. This balances exploration and exploitation, gradually shifting from exploration to exploitation as the training progresses.
- **Target Update Frequency (for DQN):** The target Q-network, used to generate target Q-values for training, was updated at a fixed frequency to provide more stable target values.
- **Policy Update Frequency (for TD3):** The policy network in TD3 was updated less frequently than the Q-networks to prevent destabilizing updates and ensure more stable learning.

3.4. Training Procedure

The training process involved the following steps:

1. **Initialization:** Initialize the environment, the Q-network (for DQN), the actor, and the critic networks (for TD3). Initialize replay buffers to store experience tuples (s_t, a_t, r_t, s_{t+1}).
2. **Experience Collection:** At each time step, t, the TurtleBot3 interacts with the environment by selecting actions based on the current policy. The resulting experiences (s_t, a_t, r_t, s_{t+1}) are stored in the replay buffer.
3. **Network Updates:** Periodically sample mini-batches of experiences from the replay buffer to update the neural networks. For DQN, update the Q-network using the Bellman equation. For TD3, update the twin Q-networks and the policy network using the sampled experiences.
4. **Hyperparameter Tuning:** Fine-tune hyperparameters such as learning rates, batch sizes, epsilon decay rates, and update frequencies through a series of experiments to achieve optimal performance.

5. **Evaluation:** Periodically evaluate the performance of the trained policies by testing the TurtleBot3 in the simulated environment. Metrics such as the success rate of reaching the goal and the number of collisions were used to assess performance.

4. Contribution

This section highlights the specific contributions made to the repository and the overall project on autonomous vehicle navigation using reinforcement learning algorithms. The primary contributions include enhancements to the DQN and TD3 algorithms, the implementation of additional features for improved training performance, and extensive hyperparameter tuning.

- **Integration of Learning Rate Scheduler:** One of the key contributions was the integration of a learning rate scheduler into the training process for both DQN and TD3 algorithms. The learning rate scheduler dynamically adjusts the learning rate based on the training progress, allowing for more efficient convergence to optimal policies. This helps in avoiding potential pitfalls of using a fixed learning rate, such as slow convergence or overshooting the optimal solution.
- **Addition of Batch Normalization Layers:** To stabilize and accelerate the training of neural networks, batch normalization layers were added to both the Q-network (for DQN) and the actor-critic networks (for TD3). Batch normalization normalizes the inputs of each layer, mitigating the internal covariate shift and enabling faster and more stable training. This enhancement significantly improved the performance and robustness of the trained policies.
- **Extensive Hyperparameter Tuning:** Significant efforts were made in hyperparameter tuning to optimize the performance of both DQN and TD3 algorithms. Key hyperparameters such as learning rates, batch sizes, epsilon decay rates, target update frequencies, and policy update frequencies were systematically tuned through a series of experiments. This rigorous tuning process ensured that the algorithms performed optimally in the simulated environment, leading to efficient and robust autonomous navigation.
- **Comparative Analysis of DQN and TD3:** A comprehensive comparative analysis of the DQN and TD3 algorithms was conducted. The performance of both algorithms was evaluated based on various criteria, including convergence speed, navigation efficiency, and robustness to environmental changes. This analysis provided valuable insights into the strengths and weaknesses of each algorithm, contributing to the broader understanding of their applicability in autonomous navigation tasks.

The implemented modifications and optimizations, such as the learning rate scheduler and batch normalization, provide valuable insights for improving the training process of similar models. Additionally, the extensive hyperparameter tuning offers a systematic approach for optimizing RL algorithms, which can be applied to other robotic systems. Furthermore, the code utilized for training and testing the DQN and TD3 algorithms on TurtleBot3 is publicly available on the following repository: `https://github.com/ROBOTIS-GIT/turtlebot3_drlnav/tree/main`.

5. Results

Here, we present a detailed analysis of the results obtained from training the TurtleBot3 using the Deep Q-Network (DQN) and Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithms. The performance of these algorithms was evaluated with and without hyperparameter tuning. We analyze key metrics such as:

Navigation Outcomes: Success rate of reaching the goal and collision rates with static (walls) and dynamic obstacles. Average Network Losses: Average critic loss (for both DQN and TD3) and average actor loss (for TD3 only). Average Rewards: Average reward per episode.

Graphical representations are included to visualize the performance differences.

5.1. DQN Algorithm Performance

5.1.1 Without Hyperparameter Tuning

The initial training of the TurtleBot3 with the DQN algorithm without hyperparameter tuning resulted in a high number of collisions with obstacles (both static and dynamic). The outcomes graph (Figure 4) indicates a low success rate in navigation tasks compared to collision rates. The average actor loss remained zero throughout the episodes since DQN does not employ an actor network. The average critic loss (Figure 4) exhibited a high initial value, gradually decreasing over time. This suggests some level of learning, but the variance remained high, indicating learning instability.

The average reward over 10 episodes (Figure 4) showed significant fluctuations, further suggesting instability. The frequent ups and downs imply that the robot's policy was not consistently improving, likely due to the lack of systematic exploration and exploitation.

5.1.2 With Hyperparameter Tuning

Hyperparameter tuning significantly improved DQN performance. The outcomes graph (Figure 5) demonstrated a noticeable increase in successful navigations and a reduction

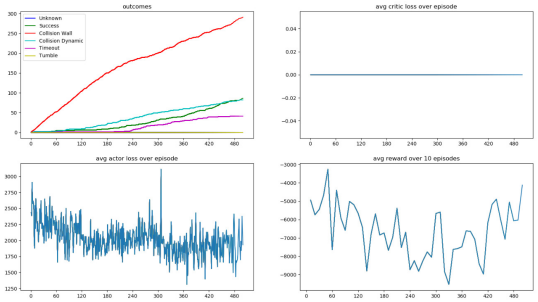

Figure 4: DQN Untuned

in collisions. This improvement highlights the positive impact of tuning parameters such as learning rate, batch normalization, and epsilon decay.

The average actor loss remained zero, as expected. The average critic loss (Figure 5) decreased more rapidly and stabilized at a much lower value, indicating more effective Q-network learning. This stability is crucial for consistent policy improvement.

The average reward over 10 episodes (Figure 5) showed a more consistent upward trend, reflecting improved learning stability and performance. The reduced fluctuations imply that the tuned DQN algorithm balanced exploration and exploitation more effectively, leading to steady improvements in navigation.

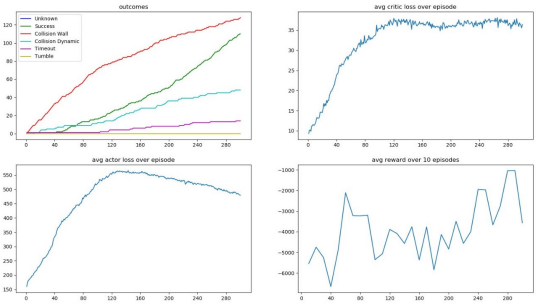

Figure 5: DQN Tuned

5.2. TD3 Algorithm Performance

5.2.1 Without Hyperparameter Tuning

Similar to the untuned DQN, the initial performance of the TD3 algorithm without hyperparameter tuning exhibited a high number of collisions with walls and dynamic obstacles (refer to results section for figures). The outcomes graph (refer to results section for figures) shows a low number of successful navigations, highlighting the challenges faced by the robot in learning effective navigation strategies in a complex environment.

The average critic loss (Figure 6) increased initially, which is expected as the algorithm attempts to estimate the Q-values. Over time, the critic loss stabilized (Figure 6), indicating some level of learning. However, the stabilization occurred at a higher loss value, suggesting that the learning process was not optimal. The average actor loss (Figure 6) showed a decreasing trend but with significant variance, indicating instability and inconsistency in policy improvement.

The average reward over 10 episodes (Figure 6) exhibited substantial fluctuations, reflecting the instability observed in the actor and critic losses. The inconsistent rewards suggest that the policy was not reliably improving, which is a common issue when the algorithm is not adequately tuned.

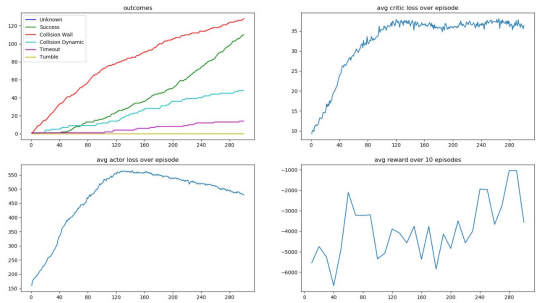

Figure 6: TD3 Untuned

5.2.2 With Hyperparameter Tuning

The performance of the TD3 algorithm improved markedly with hyperparameter tuning. The outcomes graph (refer to results section for figures) revealed a significant reduction in collisions and a higher number of successful navigations, indicating that the tuned TD3 was more effective in learning robust navigation strategies. This improvement underscores the importance of hyperparameter tuning in enhancing the learning efficiency of RL algorithms.

The average critic loss (Figure 7) increased initially and then stabilized at a lower value compared to the untuned version, indicating more efficient learning and better Q-value estimation. The average actor loss (Figure 7) showed a steady decrease and stabilized at a lower value, reflecting improved stability and consistency in the policy updates.

The average reward over 10 episodes (Figure 7) demonstrated a consistent upward trend, indicating enhanced learning performance and stability. The reduced fluctuations in the reward graph suggest that the tuned TD3 al-

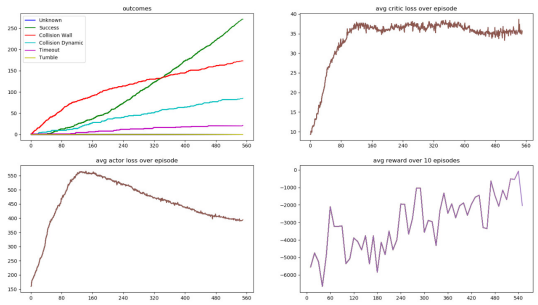

Figure 7: TD3 Tuned

gorithm was able to maintain a more stable learning process, leading to reliable improvements in navigation performance.

6. Comparative Analysis

The comparative analysis evaluates the performance of the Deep Q-Network (DQN) and Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithms in achieving autonomous navigation for the TurtleBot3. This comparison focuses on learning efficiency, stability, and overall navigation performance.

6.1. Learning Efficiency

DQN exhibited slower initial learning compared to TD3. The primary reason for this is the nature of the DQN algorithm, which relies on a discrete action space and an epsilon-greedy strategy for exploration. Early in the training process, DQN required extensive exploration to discover optimal actions, which prolonged the convergence time. Careful tuning of the epsilon decay rate was necessary to balance exploration and exploitation effectively.

In contrast, TD3, with its continuous action space, demonstrated faster learning efficiency. The algorithm benefitted from policy smoothing and delayed policy updates, which contributed to rapid convergence. These features allowed TD3 to make more refined adjustments to the policy, leading to quicker identification of effective navigation strategies.

6.2. Learning Stability

During training, DQN showed less stability. The algorithm required frequent adjustments to the learning rate and exploration parameters to maintain consistent performance. The single Q-network approach of DQN occasionally led to overestimation biases, which destabilized the learning process. As a result, DQN's learning curves were often more erratic, necessitating additional tuning to achieve stable training.

TD3, on the other hand, demonstrated higher stability. The use of clipped double Q-learning mitigated overestimation biases by utilizing two Q-networks and taking the minimum value for policy updates. This approach provided more reliable value estimates and improved the algorithm's overall stability. Additionally, the delayed policy updates and target policy smoothing techniques further enhanced training stability, resulting in smoother and more predictable learning curves.

6.3. Navigation Performance

In terms of navigation performance, DQN enabled the TurtleBot3 to navigate the simulated environment effectively, but only after an extended training period. The policies learned by DQN allowed the robot to avoid obstacles, but the navigation paths were sometimes suboptimal. This suboptimal performance indicated that the discrete action space of DQN limited the precision of the policy, resulting in less efficient navigation routes.

TD3, however, delivered superior navigation performance. The continuous action space provided by TD3 allowed the TurtleBot3 to make finer control adjustments, leading to more efficient and smoother paths to target locations. The enhanced control granularity facilitated better obstacle avoidance and overall navigation efficiency. The TurtleBot3, guided by TD3, consistently found optimal paths, demonstrating the algorithm's capability to handle complex navigation tasks more effectively.

7. Conclusion

7.1. Key Findings

This study successfully demonstrated the application of reinforcement learning (RL) algorithms, specifically Deep Q-Networks (DQN) and Twin Delayed Deep Deterministic Policy Gradient (TD3), for autonomous navigation of the TurtleBot3 robot. The integration of a learning rate scheduler, batch normalization layers, and comprehensive hyperparameter tuning significantly improved the performance and stability of both models. The tuned models enabled the TurtleBot3 to navigate the simulated ROS Gazebo environment efficiently, avoiding obstacles and reaching designated goals.

7.2. Contributions to the Field

This project contributes to the field of autonomous robotics by enhancing the effectiveness of RL algorithms for real-world applications. The implemented modifications and optimizations, such as the learning rate scheduler and batch normalization, provide valuable insights for improving the training process of similar models. Additionally, the extensive hyperparameter tuning offers a systematic approach for optimizing RL algorithms, which can be applied

to other robotic systems.

7.3. Limitations

Despite the successful outcomes, the study has some limitations. The training and evaluation were conducted in a simulated environment, which may not fully capture the complexities of real-world scenarios. The transferability of the trained models to actual TurtleBot3 hardware remains to be tested. Additionally, the study focused on DQN and TD3 algorithms; exploring other RL approaches could provide further improvements.

7.4. Future Research

Future research could address the limitations identified in this study by testing the trained models on actual TurtleBot3 hardware and evaluating their performance in real-world environments. Investigating the application of other reinforcement learning algorithms, such as PPO (Proximal Policy Optimization) or SAC (Soft Actor-Critic), could offer additional insights and improvements. Furthermore, exploring the integration of multi-agent systems and collaborative learning could enhance the capabilities of autonomous robots in more complex scenarios.

This project demonstrates the feasibility and effectiveness of using reinforcement learning algorithms to train autonomous robotic systems. The enhancements and optimizations implemented contribute valuable knowledge to the field, providing a foundation for future research and development. The findings underscore the potential of reinforcement learning in advancing the capabilities of autonomous robots, highlighting the significance of this study in the broader context of robotics and artificial intelligence.

References

[1] J. Escobar-Naranjo, G. Caiza, P. Ayala, E. Jordan, C. A. Garcia, and M. V. Garcia. Autonomous navigation of robots: Optimization with DQN. *Appl. Sci.*, 13(12):7202, 2023.

[2] X. Wang, Y. Sun, Y. Xie, J. Bin, and J. Xiao. Deep reinforcement learning-aided autonomous navigation with landmark generators. *Front. Neurorobot.*, 17:1200214, 2023.

[3] A. K. Mackay, L. Riazuelo, and L. Montano. RL-DOVS: Reinforcement learning for autonomous robot navigation in dynamic environments. *Sensors*, 22(10):3847, 2022.

[4] M.-F. R. Lee and S. H. Yusuf. Mobile robot navigation using deep reinforcement learning. *Processes*, 10(12):2748, 2022.

[5] OpenAI Gym. TurtleBot3 Navigation with Deep Reinforcement Learning. [Online]. Available: `https://github.com/tomasvr/turtlebot3_drlnav/tree/main` [Accessed: 2024-05-16].

[6] Neptune AI. How to choose a learning rate scheduler. [Online]. Available: `https://neptune.ai/blog/how-to-choose-a-learning-rate-scheduler` [Accessed: 2024-05-16].