Adaptive Robot Navigation Using Randomized Goal Selection with Twin Delayed Deep Deterministic Policy Gradient

robotics

MDPI

Article

Adaptive Robot Navigation Using Randomized Goal Selection with Twin Delayed Deep Deterministic Policy Gradient

Romisaa Ali [1,*], Sedat Dogru [2], Lino Marques [2] and Marcello Chiaberge [3]

[1] Department of Computer and Control Engineering (DAUIN), Politecnico di Torino, 10129 Turin, Italy
[2] Institute of Systems and Robotics, Department of Electrical and Computer Engineering, University of Coimbra, 3000-214 Coimbra, Portugal; sedat@isr.uc.pt (S.D.); lino@isr.uc.pt (L.M.)
[3] Department of Electronics and Telecommunications (DET), Politecnico di Torino, 10129 Turin, Italy; marcello.chiaberge@polito.it
* Correspondence: romisaa.ali@polito.it

Abstract: The primary challenge in robotic navigation lies in enabling robots to adapt effectively to new, unseen environments. Addressing this gap, this paper enhances the Twin Delayed Deep Deterministic Policy Gradient (TD3) model's adaptability by introducing randomized start and goal points. This approach aims to overcome the limitations of fixed goal points used in prior research, allowing the robot to navigate more effectively through unpredictable scenarios. This proposed extension was evaluated in unseen environments to validate the enhanced adaptability and performance of the TD3 model. The experimental results highlight improved flexibility and robustness in the robot's navigation capabilities, demonstrating the ability of the model to generalize effectively to unseen environments. Additionally, this paper provides a concise overview of TD3, focusing on its core mechanisms and key components to clarify its implementation.

Keywords: reinforcement learning; TD3; SAC; jackal robot; robust navigation; transferability

Academic Editor: Giovanni Muscato

Received: 26 January 2025
Revised: 18 March 2025
Accepted: 25 March 2025
Published: 31 March 2025

Citation: Ali, R.; Dogru, S.; Marques, L.; Chiaberge, M. Adaptive Robot Navigation Using Randomized Goal Selection with Twin Delayed Deep Deterministic Policy Gradient. *Robotics* **2025**, *14*, 43. https://doi.org/10.3390/robotics14040043

1. Introduction

Reinforcement learning (RL) has become a powerful tool in developing adaptive control policies for autonomous robots navigating in complex environments [1]. As autonomous systems, particularly mobile robots, are deployed in dynamic and unpredictable real-world settings, robust navigation strategies become crucial. This challenge is especially relevant for skid-steered robots, widely known for their durability and versatility in various robotic applications [2,3]. Despite their potential, making these robots navigate efficiently and safely through diverse and unseen environments remains a significant research challenge.

Recent deep reinforcement learning (DRL) methods [4], such as DreamerV2 [5] and OPAC [6], have demonstrated significant advancements. However, DRL methods still require time to fully evolve and effectively integrate with professional machine learning tools. While methods like DQN [7] and DDPG [8] paved the way for modern DRL-based navigation, methods like DreamerV2 and MvP have further expanded model-based RL techniques.

Methods like TD3, SAC [9], and PPO [10] have gained wide adoption due to their compatibility with modern machine learning frameworks. For example, Roth et al. [11,12] have successfully deployed PPO in real-world scenarios and utilized XAI techniques to enhance interpretability. Additionally, its effectiveness in progressively complex dynamic environments has been demonstrated. Akmandor et al. [13,14] specifically highlighted

PPO's capability in handling increasingly dynamic environments by leveraging occupancy-based representations. Similarly, a previous study by Ali et al. [15] compared TD3 and SAC, demonstrating TD3's superior consistency and stability in navigating unseen environments. The empirical success of TD3 is further supported by studies demonstrating its ability to outperform earlier approaches such as DDPG in benchmarked OpenAI Gym environments [16].

The choice of a DRL algorithm is critical in developing navigation policies that are both efficient and adaptable. TD3, introduced by Fujimoto et al. [17] at NeurIPS, addresses overestimation bias and improves policy stability in continuous control tasks through dual critic networks and delayed updates. This prevents overestimation errors and enhances the stability of learned policies, while noise signals promote effective exploration, reducing the likelihood of premature convergence to suboptimal solutions. Leveraging these strengths, various studies have explored the use of TD3 in navigation. For instance, Cimurs et al. [18,19] applied TD3 to autonomous exploration, where randomized goal selection improves trajectory diversity and mapping efficiency. However, these studies did not explicitly assess the effect of randomization on learning efficiency and generalization. Anas et al. [20,21] also trained a TD3-based model in an indoor scenario with multiple obstacles, achieving a high success rate in simulation but facing deployment challenges due to manually controlled obstacle placement.

Randomized training techniques have emerged as a powerful strategy to enhance generalization and adaptability in reinforcement learning-based navigation. One widely used approach involves randomized start and goal points, preventing overfitting to specific navigation paths and improving robustness in unseen environments. Studies have demonstrated the effectiveness of this approach in different contexts. In robotic motion planning, Wong et al. [22] showed that randomized start and goal positions reduce self-collision occurrences and improve joint limit handling. However, while such findings were observed in robotic arm applications, mobile robot navigation presents distinct challenges. In mobile robot settings, Akmandor et al. [13] incorporated randomized start and goal selection in PPO-based systems, focusing on occupancy-based representations rather than explicitly analyzing the impact of randomization. Similarly, Cimurs et al. [18] utilized randomized goal selection in TD3-based navigation for exploration and mapping, but without directly evaluating the impact of randomization of start and goal points.

Building upon these insights, this work systematically investigates the role of randomized start and goal points in improving trajectory execution and robust navigation in static environments. While prior studies have incorporated randomization in different forms, its direct impact on learning efficiency and generalization remains underexplored. This work fills this gap by evaluating performance of TD3 under randomized conditions in customized static environments, providing a comprehensive analysis of how structured variability enhances reinforcement learning-based navigation.

The primary contribution lies in updating environment setup and designing new scenarios to enable the robot to navigate effectively in a new setup, enhancing the adaptability and generalization of TD3. For this end, randomized start and goal points are introduced during training, exposing the robot to various navigation scenarios. This improves the robot's ability to handle unseen environments by enabling more flexible and adaptive navigation strategies. Despite advancements in DRL for robot navigation, many approaches struggle to generalize to diverse and unseen environments, especially due to reliance on fixed start and goal points during training, which limits adaptability in unexpected scenarios.

To address these research gaps, this work extends prior RL-based navigation strategies by enabling the robot to experience diverse navigation scenarios during training, as

discussed in Section 3.1. This significantly improves adaptability and robustness in unseen environments. As demonstrated in Section 6, this randomization technique enhances generalization, enabling the model to successfully navigate in previously unseen environments. Building on existing RL-based methods, this research improves the model's ability to generalize beyond training conditions, ensuring better adaptability and transferability to real-world scenarios.

The rest of this paper is arranged as follows: Section 2 provides a background of the TD3 algorithm and key concepts related to its robustness in robot navigation tasks. Section 3 outlines the methodology and experimental setup used to assess the performance of the extended TD3 model. Section 4 outlines the experimental design and the evaluation metrics used to compare the performance of the two TD3 models. Section 5 presents the training results and comparative analysis of the extended TD3 model with non-extended setups. Finally, Section 6 evaluates the model's ability to generalize across unseen environments, and Section 7 concludes the paper with key findings and recommendations for future work in DRL-based robot navigation.

2. Background

2.1. TD3 Architecture and Principles

The architecture of TD3 consists of several key components as defined by Scott Fujimoto et al. [17]: the actor network, two critic networks, and their corresponding target networks (see Figure 1). The following is a detailed explanation of each component:

- Policy Network (Actor Network): The actor network, denoted as π_ϕ, is responsible for selecting actions given the current state. It approximates the policy function and is parameterized by ϕ. The actor network outputs the action that the agent should take in a given state to maximize the expected return.
- Critic Networks (Q_{θ_1} and Q_{θ_2}): TD3 employs two critic networks, Q_{θ_1} and Q_{θ_2}, to estimate the Q-values for state–action pairs (s, a). Each network outputs a scalar value that represents the expected return for a specific state–action pair. The use of two critics helps reduce overestimation bias, which can occur when function approximation is used in reinforcement learning. By employing double Q-learning, TD3 ensures more accurate value estimates, enhancing the stability of policy updates.
- Critic Target Networks: The critic target networks, $Q_{\theta'_1}$ and $Q_{\theta'_2}$, are delayed versions of the primary critic networks and are updated at a slower rate. These target networks provide stable Q-value targets for the critic updates, which helps to reduce variance and prevents divergence during training. By maintaining more conservative Q-value estimates, the critic target networks further mitigate the overestimation bias found in reinforcement learning models.
- Actor Target Network: The actor target network, $\pi_{\phi'}$, is a delayed copy of the actor network and is essential for generating stable action targets for the critic updates. The actor target network changes gradually over time, ensuring that the target actions used in critic updates are consistent and stable. This stability is critical for reducing the variance in the Q-value targets provided by the critic target networks, leading to more reliable and stable training of the critics. The actor target network's slow updates help in maintaining a steady learning process and improve the overall performance of the TD3 algorithm by preventing rapid and unstable policy changes.

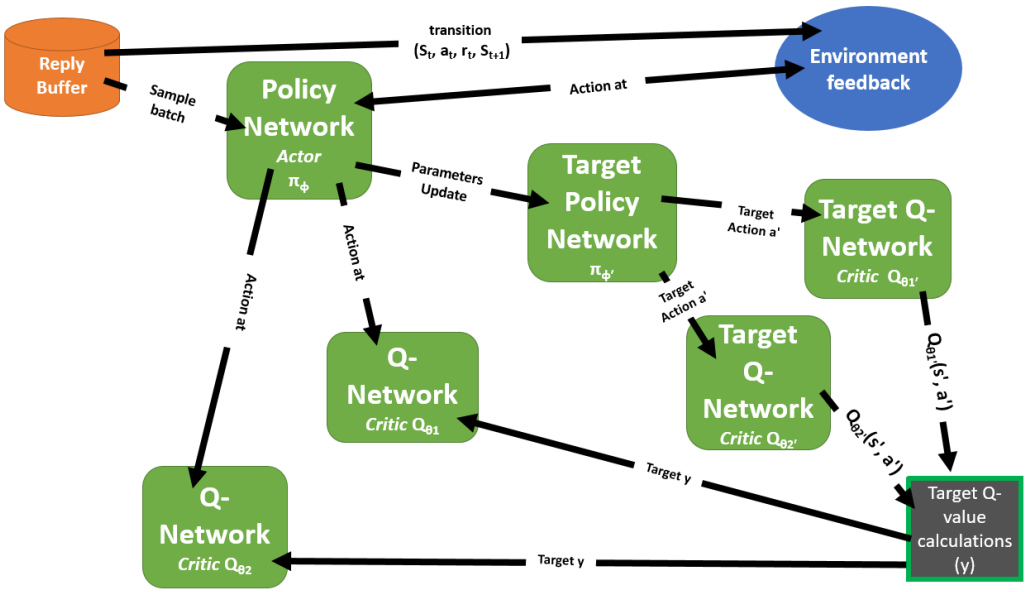


Figure 1. TD3 algorithm architecture illustrating the connections between the policy network (actor), the twin critic networks, and the target networks.

2.2. TD3 Networks and Updates

The TD3 algorithm integrates actor and critic networks, each paired with corresponding target networks, to achieve stable and efficient policy updates. The actor network (π_ϕ) determines actions that maximize the expected return $J(\phi)$, and its parameters are updated based on the computed policy gradient $\nabla_\phi J(\phi)$. This gradient combines the Q-value gradient ($\nabla_a Q_{\theta_1}(s,a)$) from the primary critic and the policy gradient ($\nabla_\phi \pi_\phi(s)$).

The critic networks (Q_{θ_1} and Q_{θ_2}) evaluate the expected returns for state–action pairs and minimize overestimation bias by employing Clipped Double Q-learning. They calculate a target value $y = r + \gamma \min(Q_{\theta'_1}(s', \tilde{a}), Q_{\theta'_2}(s', \tilde{a}))$, where $\tilde{a} = \pi_{\phi'}(s') + \epsilon$, and ϵ represents added noise for regularization. The critic parameters θ_1 and θ_2 are optimized by minimizing the mean squared error between predicted Q-values and y.

To ensure stable training, the target networks ($\pi_{\phi'}$, $Q_{\theta'_1}$, $Q_{\theta'_2}$) are updated via a soft update mechanism. Their parameters gradually approach the parameters of the current networks through $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$ for the critics and $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$ for the actor. This mechanism reduces variance and stabilizes policy updates by ensuring reliable Q-value and action targets during training.

Symbol Definitions

- π_ϕ: Actor network responsible for selecting actions.
- $Q_{\theta_1}, Q_{\theta_2}$: Critic networks estimating Q-values for state–action pairs.
- $\pi_{\phi'}$, $Q_{\theta'_1}$, $Q_{\theta'_2}$: Target networks providing stable targets for updates.
- $J(\phi)$: Expected return, optimized by the actor network.
- r: Immediate reward received after taking an action.
- γ: Discount factor for future rewards.
- $\tilde{a}$: Noise-regularized action used in target Q-value calculation.
- τ: Soft update rate for target network parameters.
- ϵ: Regularization noise added to target actions.
- θ_1, θ_2: Parameters of the primary critic networks.
- ϕ: Parameters of the actor network.

3. Methodology and Experimental Setup

The research methodology operates within a Singularity container [23], integrating the Noetic ROS system framework with a simulation of the Jackal robot and a custom OpenAI Gym environment. These components work together with DRL, which was developed using the PyTorch library (version 1.10.1, developed by the PyTorch Foundation under the Linux Foundation, San Francisco, CA, USA) to train the robot for navigation tasks. All simulations are conducted in the Gazebo simulator, using a custom motion control module, called the *MotionControlContinuousLaser*. This module is designed for the Jackal robot and it provides a continuous action space consisting of linear and angular velocities, enabling smooth robot movement. The module also processes LiDAR data as part of the observation space, allowing the agent to access sensory information for decision-making. By integrating both motion control and sensor data processing, the module allows for evaluating reinforcement learning algorithms in realistic and challenging scenarios. Training data are collected through interactions with the simulated environments and saved locally. The data are stored in a replay buffer, allowing for randomized sampling during the training process. The parameters of the model are updated continuously, and new policies are saved locally until the model converges.

The computational power for this training was provided by an AMD Ryzen 9 3900X CPU (12 cores, 24 threads), an NVIDIA GeForce GTX 1660 Ti GPU, and 32 GB of RAM (Coimbra, Portugal). However, only the CPU was utilized for the training process, while the GPU remained unused. The training relied entirely on the available CPU cores, running six models concurrently, each within a Singularity container. The training process took approximately five days to complete.

3.1. Jackal Robot Dynamics and Sensors

The Jackal robot is equipped with two primary sensors directly connected to the DRL algorithm and the robot's control system:

- Laser Sensor:The LiDAR sensor provides essential data for navigation by scanning the environment and detecting obstacles. It generates a 720-dimensional laser scan, which is directly used by the DRL algorithm to inform the robot's decision-making process. These LiDAR data allow the robot to avoid collisions and select efficient paths to reach its goal. In the Gazebo simulation, the LiDAR sensor replicates real-world sensor dynamics, ensuring that the robot can adapt to complex environments during training.
- Velocity Sensor: The velocity sensor monitors the robot's movement by tracking the linear and angular velocities issued by the DRL node. These commands control the robot's speed, ensuring that the actual movement matches the intended commands. This feedback helps maintain stable movement and highlights any deviations, such as low velocity, that may affect navigation performance.

3.2. Training and Evaluation Scenarios

In this work, the training process of the TD3 algorithm was expanded by modifying the environment setup source codes to enhance adaptability and generalization in navigation tasks. This extension is based on the work of Xu et al. [16,24], which originally defined a fixed goal-oriented navigation framework. The proposed approach introduces significant modifications, including randomized start and goal points, object placement variations, and increased path diversity, as detailed in Section 3.2.2. These enhancements expose the model to a wider variety of navigation conditions, allowing it to adapt more effectively to unpredictable scenarios and reducing the risk of overfitting to fixed paths. Since the actor network selects actions based on its observations, it is forced to continuously learn how to generate optimal actions dynamically rather than relying on pre-learned shortcuts.

The critic networks, which are responsible for estimating Q-values, also benefit from this expanded exploration space, because they receive a more diverse set of state–action pairs. These allow them to generate more accurate Q-value estimates across different navigation scenarios. By increasing state–action diversity, the TD3 model optimizes action selection more effectively. This improves both learning efficiency and generalization, ultimately helping the model discover the most optimal solutions in complex and unpredictable navigation scenarios. The main contribution and extension introduced in this research is the expansion of the training methodology, specifically by adding additional scenarios with randomized start and goal points, which significantly increases the variability and complexity of training conditions compared to the original fixed-point scenarios. This expansion improves the model's exploration capabilities, robustness, and adaptability to unpredictable paths. To clearly distinguish and validate the real efficiency of the expanded training process, TensorBoard was used as a graphical controller to monitor model convergence, training stability, and performance metrics, enabling real-time visualization of the algorithm's learning progress and generalization capabilities. All details regarding the implementation of this extension, including environment modifications and training setup, are available in [25].

3.2.1. Static Box Environments

In line with previous work [16,24], the static box environment measuring 4.5 m × 4.5 m was used for training. This environment consists of stationary obstacles arranged in various patterns, providing a controlled setting to assess the robot's navigation performance under static conditions. Figure 2 shows the layout of the static box environment as originally used by Xu et al. [26].

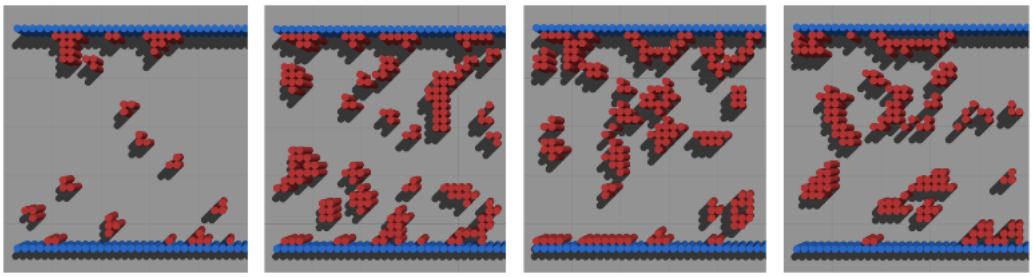


Figure 2. Original training setup: Layout of four different static box environments (4.5 m × 4.5 m) for training.

3.2.2. Custom Static Environments with Randomized Start and Goal Points

These custom environments consist of 16 unique scenarios, each with distinct designs and varying object positions, creating a diverse range of challenges for the robot. Unlike the original 4.5 m × 4.5 m static boxes, which limited flexibility due to their small size, these environments are expanded to 16 m × 16 m, providing wide space for randomization of start and goal points in each episode. This increased area facilitates better exploration and ensures more robust generalization by exposing the robot to a wider variety of paths and obstacles. Each scenario is specifically tailored with unique layouts, different difficulty levels, and diverse obstacle configurations, all designed to enhance the robot's adaptability, robustness, and overall navigation performance in complex settings.

The navigation scenarios consist of two types of static objects: boxes and cylinders. Box objects have a fixed dimension of 1 m (length) × 1 m (width) × 1 m (height) and are used as obstacles within the navigation area to add variety to the layouts. Cylinder objects come in five distinct dimensions: Cylinder 1 (radius: 0.14 m, height: 1 m), Cylinder 2 (radius: 0.22 m, height: 1 m), Cylinder 3 (radius: 0.30 m, height: 2.19 m), Cylinder 4 (radius: 0.13 m, height: 1 m), and Cylinder 5 (radius: 0.30 m, height: 2.19 m). The border walls of

each scenario are constructed using Cylinder 5, which is also utilized as an obstacle inside the navigation area to further diversify the layouts. All objects are static but fully collidable, and any collision with the robot during navigation is registered in the evaluation metrics. The objects are distributed strategically within the 16 scenarios, ensuring a variety of paths, densities, and obstacle arrangements. The environments were designed to challenge the robot and enhance its generalization capabilities.

In these environments, start and goal points are generated randomly and validated based on specific criteria to ensure feasibility. The randomization process ensures that the distance between the start and goal points is at least 8 m and no more than 11 m, providing a balanced level of challenge. Additionally, both points are required to maintain a 1.7 m distance to obstacles in order to avoid invalid zones, and prevent overlaps or unreachable paths. The process uses a random function to generate these points, followed by a validation check. If the points do not meet the criteria, new points are generated until valid positions are found. The robot's heading is randomized at the beginning of each new path, similar to the randomization of start and goal points. This ensures variability in the robot's starting direction, enhancing its adaptability and ability to handle diverse scenarios. Randomization is applied at the start of each new path and after failures, ensuring a fresh start with newly randomized parameters.

The same randomization and validation process is applied during testing, ensuring consistency across extended and non-extended models in identical unseen environments. Figure 3 illustrates several examples of these custom environments.

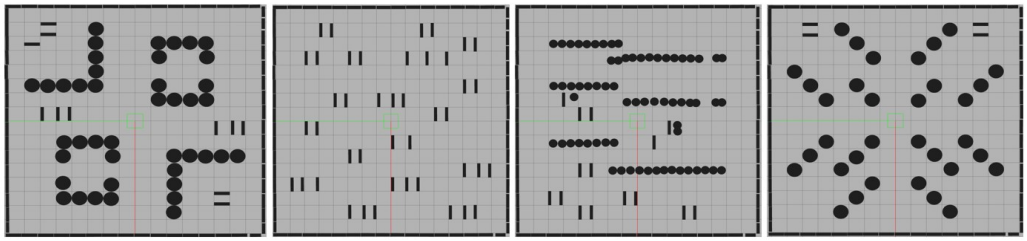


Figure 3. Layout of the four different 16 m × 16 m custom static environments used in training.

To further clarify the specific contribution of this research, Figure 4 provides a detailed block diagram illustrating the expanded TD3 training methodology.

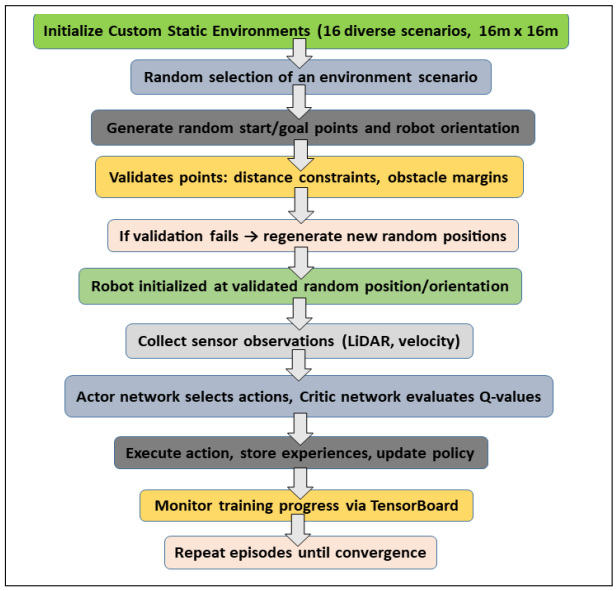


Figure 4. Block diagram of the expanded TD3 training methodology with randomized start and goal points.

3.3. Graphical Monitoring of Training Process

To monitor training performance and efficiency, TensorBoard was utilized to track key metrics such as success rate and collision rate during training. The training process was recorded in event logs generated during model training, as detailed in [25]. These logs allowed for the visualization of performance trends, including the progression of success and collision rates over training steps. Figures 5 and 6 in the results section illustrate these changes, providing insights into how randomized start and goal points influenced the learning process. This graphical monitoring approach ensures a comprehensive assessment of model training dynamics and convergence behavior.

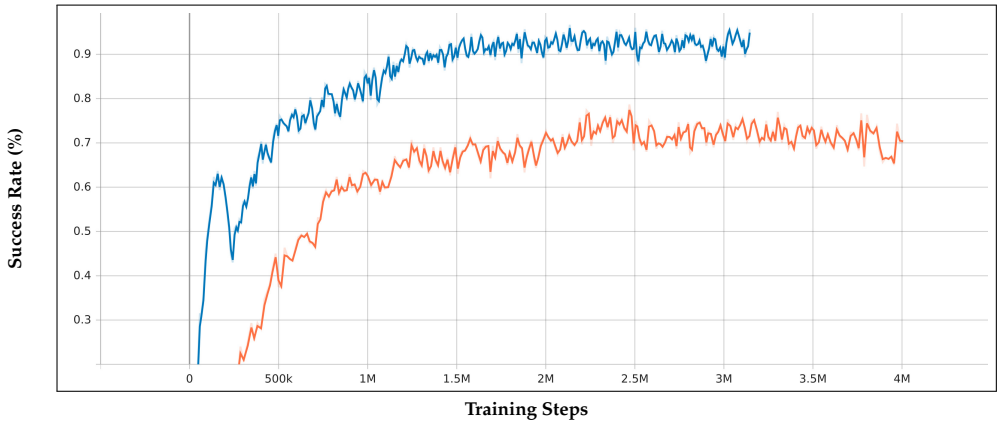


Figure 5. Success rates of the extended model (blue) and the non-extended model (orange) over training steps.

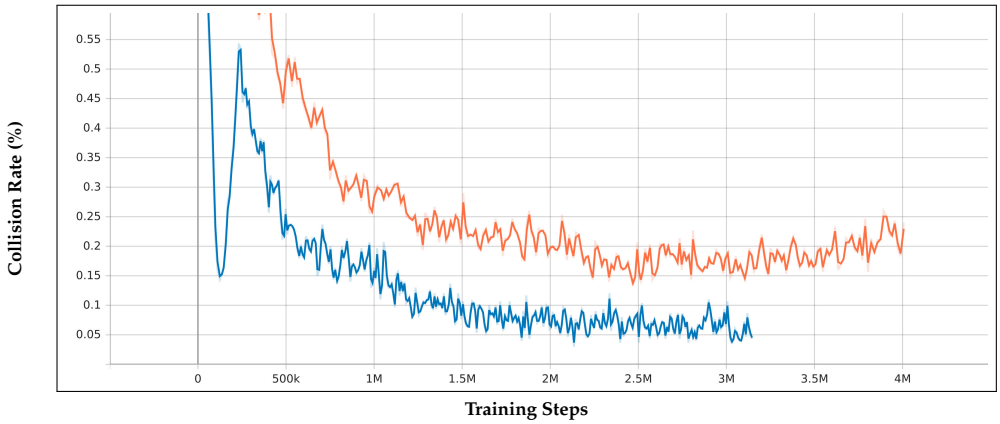


Figure 6. Collision rates of the extended model (blue) and the non-extended model (orange) over training steps.

4. Experimental Comparison of TD3 Models

The goal of this experiment is to compare the performance of the improved TD3 model, which was trained using 16 custom static environments defined in Section 3, with the baseline model trained by Xu et al. [24] on 300 static environments from the original setup. By comparing these two models, this study examines the impact of environment diversity and training methodology on overall performance.

4.1. Training and Evaluation Metrics

For the evaluation of the extended TD3 model, a comprehensive set of performance metrics is used to provide a thorough analysis of the behavior of the model in unseen environments. The key metrics include the following:

- Success Rate: The percentage of episodes where the robot successfully reaches the goal without collisions.
- Collision Rate: The percentage of episodes where the robot collides with obstacles.
- Episode Length: The average duration (in terms of steps) taken by the robot to complete a task.
- Average Return: The cumulative reward earned by the robot over an episode, averaged over all episodes, to evaluate the learning efficiency of the algorithm.
- Time-Averaged Number of Steps: The average number of steps the robot takes to complete each episode, providing insight into the efficiency of the navigation strategy.
- Total Distance Traveled: This metric, calculated during the test phase, measures the total distance (in meters) the robot travels to complete a path. It provides an additional layer of analysis by assessing the efficiency of the robot's movements in terms of path length.

4.2. Baseline Comparison in Static Environments

A baseline comparison is conducted using tests in simulation to assess the effectiveness of the proposed training method, focusing on randomized start and goal selection in DRL-based navigation. This comparison includes only DRL-based approaches tested in static environments, ensuring consistency in the evaluation. For Cimurs et al. [18], multiple simulation evaluations were conducted, but only the results from static environments are considered. The GDAE method (TD3 with a global strategy) is used as the baseline for comparison. Similarly, Akmandor et al. [13] conducted evaluations in both static and dynamic environments; however, for fairness, only results from static settings are included. The two DRL-based methods tested (Tentabot FC and Tentabot 1DCNN FC, both trained using PPO without a global strategy) are averaged to provide a single representative performance value. The proposed method, utilizing TD3 with a global strategy, was tested in a maze-like static environment with long walls, allowing a structured comparison against other approaches tested in static environments. Table 1 summarizes the results of this baseline comparison.

Table 1. Comparison of simulation results in static environments. Bold values indicate better performance for the corresponding metric.

Author	Method Used	Obstacle Type and Difficulty	Success Rate (%)	Time (s)	Distance (m)
Cimurs et al. [18]	TD3 (With Global Strategy)	A static environment with smooth walls and multiple local optima	**100**	88.03	41.42
Akmandor et al. [13]	PPO (Without Global Strategy)	Static environments with distinct structured obstacles	55	-	-
Proposed Method	TD3 (With Global Strategy)	Maze-like static environment with long walls	80.0	**16.36**	**22.87**

Cimurs et al. [18] achieved a 100% success rate in structured static environments, demonstrating the effectiveness of TD3 combined with a global strategy. However, their method required a longer time to reach the goal. In contrast, Akmandor et al. [13], using PPO without a global strategy, attained only 55% success, highlighting the limitations of a purely local approach in static environments. The proposed method, employing TD3 with a global strategy, achieved 80% success in a more complex maze-like static environment while demonstrating higher efficiency with shorter time and distance compared to Cimurs et al.

5. Training Results and Comparison

This section compares the performance of the extended model (blue plot) and the non-extended model (orange plot), focusing on success rate, collision rate, and average return.

5.1. Reward Calculation and Success Rate Comparison

The reward function used in this study follows the formulation defined by Xu et al. [16,24]. The weighting coefficients were originally designed to balance progress, collision avoidance, and successful goal-reaching. Since this work focuses on enhancing the model's generalization through start and goal point randomization, the original reward function was retained as part of the existing framework.

At each step within an episode, the reward is calculated using a combination of different factors. The reward at each step is

$$r = (-0.1 \times \text{num\_steps}) + (\text{goal\_reward} \times \text{progress}) - 1.0 \times \text{collision} + 20.0 \times \text{success} - 10.0 \times \text{failure} \quad (1)$$

where:

- Slack Penalty: -0.1 per step.
- Progress reward: $+1.0$ per meter closer to the goal.
- Collision penalty: -1.0 per collision.
- Success reward: $+20.0$ for reaching the goal.
- Failure penalty: -10.0 if the episode ends unsuccessfully.

The total reward for one episode is the sum of all step rewards:

$$R = \sum(-0.1 + (\text{goal\_reward} \cdot \text{progress}) - 1.0 \cdot \mathbf{1}_{\text{collision}} + 20.0 \cdot \mathbf{1}_{\text{success}} - 10.0 \cdot \mathbf{1}_{\text{failure}}) \quad (2)$$

The total cumulative reward across multiple episodes is the sum of the cumulative rewards from all episodes currently in the buffer. If you are considering a buffer that holds up to 300 episodes, the total cumulative reward is

$$R_{\text{total}} = R_{\text{Episode 1}} + R_{\text{Episode 2}} + \ldots + R_{\text{Episode 300}} \quad (3)$$

This means each episode contributes its cumulative reward (the sum of its step rewards) to the overall total. When new episodes are added and old ones are removed, the total cumulative reward reflects the most recent set of up to 300 episodes. The rewards ultimately reflect the model's performance, as higher cumulative rewards align with increased success rates and more efficient navigation.

As shown in Figure 5, the extended model achieved a success rate of 95%, converging before 2 million steps, while the non-extended model reached 70% after 3 million steps. This indicates the extended model's faster learning speed and higher efficiency due to the randomization of start and goal points.

5.2. Collision Rate Analysis

Figure 6 shows that the extended model had a final collision rate of 4%, compared to 24% of the non-extended model. Both models saw improvements early in training, but the non-extended model's collision rate increased again after 3.5 million steps, suggesting overtraining. The extended model maintained a more consistent collision reduction.

5.3. Training Efficiency and Convergence

The extended model demonstrated faster convergence and better learning efficiency, reaching optimal performance with fewer training steps. The randomization of start and goal points enabled the extended model to explore more diverse environments, resulting in

higher success rates. In contrast, the non-extended model, which was trained with fixed start and goal points, showed slower progress and lower success rates. Additionally, some of the environments used by the non-extended model were extremely narrow, which may have limited the robot's ability to achieve a higher success rate.

6. Transfer to Test Environments Using TD3 and Global Path Planning

To evaluate the generalizability of the extended model, transfer tests were conducted using the Competition Package [27]. This package, developed by previous researchers and specifically designed for testing, was used in its original form without any modifications or extensions in this study. However, the test environment, Race World [28], which was not included in the original package, was integrated into it to facilitate evaluation. The Race World environment features a maze-like structure with long walls, creating a challenging scenario for navigation. Both the extended model and the non-extended model were evaluated in this integrated test environment. The contributions of this work are focused entirely on the training phase, where improvements were made to the training setup within a separate package. The Competition Package, enhanced with the integrated Race World environment, served as a reliable benchmark for evaluating the models' ability to generalize to complex scenarios.

In these tests, the move_base package [29] was utilized to provide a global path planning strategy, while the main navigation was handled using the TD3 algorithm. The move_base package was responsible for generating a global path from the start to the goal, providing a reference path for the robot. The TD3-based controller then followed this global path, adapting locally in real time to avoid obstacles and navigate effectively. This hybrid strategy allowed the robot to utilize the global path for overall guidance while leveraging the adaptability of the TD3-based navigation to handle local challenges. Note that the MoveBase global planner used in this study does not rely on a predefined map or global localization. It operates solely on odometry (odom) and real-time LiDAR data, enabling navigation without prior knowledge of the environment. This ensures the DRL-based controller adapts locally in unknown environments.

6.1. Evaluation Results

Tables 2 and 3 present the evaluation results for the extended and non-extended models, respectively. These tables provide key metrics, including the total distance traveled, the number of collisions, and the success status for each of the 10 test paths. The test paths used for evaluation are detailed in Figure 7. The results highlight the superior performance of the extended model, which consistently achieved a higher success rate and fewer collisions compared to the non-extended model. Additionally, the extended model demonstrated more efficient navigation by achieving closer-to-optimal distances on most paths. In contrast, the non-extended model struggled significantly on certain complex paths, often leading to failed navigation attempts or a higher number of collisions.

6.2. Analysis of Results

Success Rate and Distance Traveled: The extended model consistently outperformed the non-extended model in terms of success rate and total distance traveled. For instance, in Path 1, the extended model completed the navigation in 18.72 m with no collisions, achieving a success metric of 100%. In contrast, the non-extended model failed, covering a distance of 33.53 m with four collisions, resulting in a navigation metric of 0%. This highlights the extended model's ability to stay closer to the optimal path, demonstrating superior generalization and efficient navigation without deviations. The non-extended model, however, faced significant challenges adapting to these scenarios.

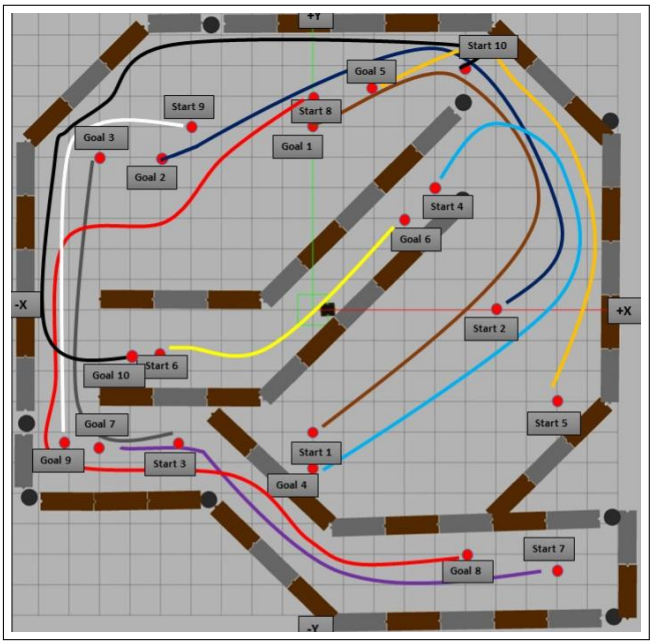


Figure 7. The environment and the 10 pairs of start and goal points used for evaluation in the simulator. The shown paths are intended only to indicate the connection between each start point and the corresponding goal point for both the extended and non-extended models. They do not represent the actual paths taken by the robot during evaluation. Each model followed a different route based on its learned navigation strategy.

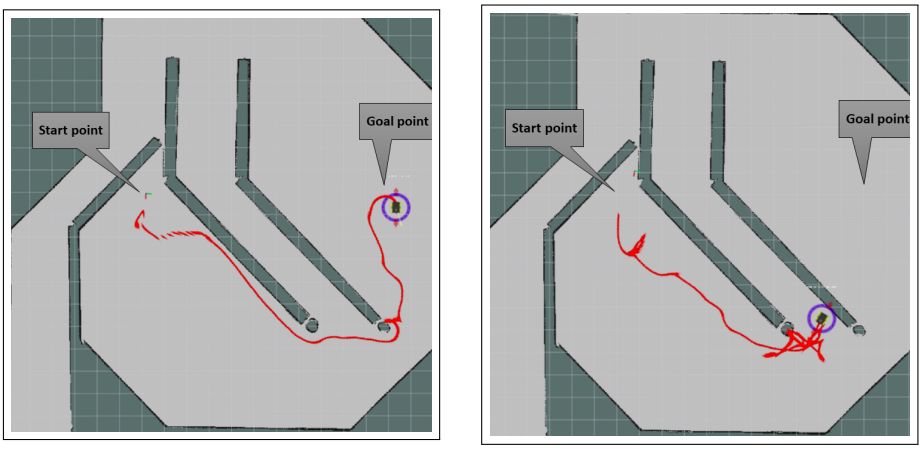
Collision Rate and Penalties: The extended model exhibited a considerably lower collision rate across all paths, with either zero or minimal collisions, whereas the non-extended model encountered multiple collisions in most paths. For example, in Path 4, the extended model recorded four collisions and failed, achieving a metric of 0%, while the non-extended model also failed with three collisions and an even higher distance penalty, resulting in similarly low performance metrics. In contrast, the extended model achieved 100% on paths where collisions were avoided, showing superior collision avoidance and efficient navigation.

Time and Distance Efficiency: The extended model demonstrated better time and distance efficiency in all paths. For instance, in Path 5, the extended model completed the navigation in 9.26 s, covering 15.51 m, while the non-extended model took 42.20 s, covering a less optimal path of 32.09 m. These differences confirm the extended model's ability to make precise navigation decisions in real time, completing tasks faster while adhering closely to the optimal route.

Generalization and Adaptability: The extended model's superior generalization capability to navigate unseen environments was evident in its successful completion of complex paths with minimal collisions and reduced distances, as shown in Tables 2 and 3. In Figures 8 and 9, where the paths taken by the models for Path 1 and Path 2 are shown, visually confirm this distinction. The extended model's trajectory is smooth and efficient, whereas the non-extended model displayed significant deviations and struggled to adapt to the scenario.

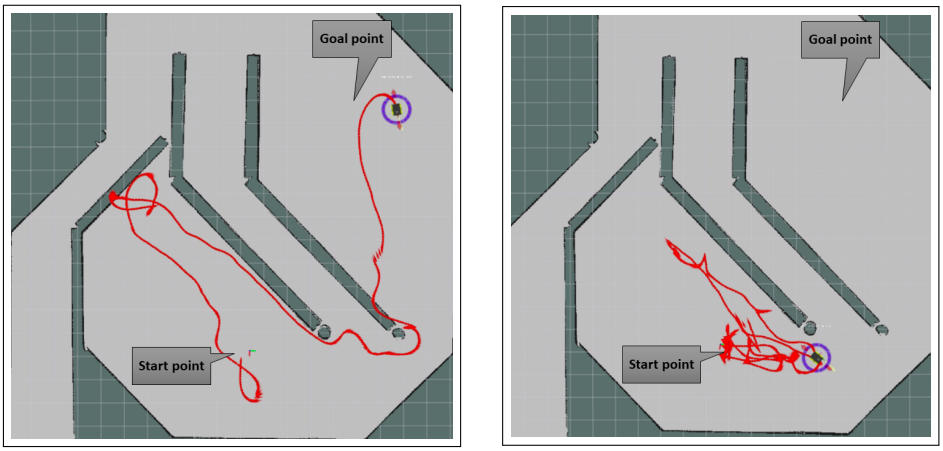
Challenges with Non-Straightforward Goals: The non-extended model showed significant difficulty when the goal was not directly aligned with the start point, reflecting a limitation in its ability to navigate complex paths. Trained with fixed start and goal points, the non-extended model primarily learned straightforward navigation, which proved insufficient in the test environments featuring complex turns and long walls (e.g., Paths 2 and 8). In Path 2, for example, the extended model navigated efficiently with a success rate of

57.60%, completing the path without timing out, whereas the non-extended model timed out after traveling a distance of 41.97 m.



(**a**) Extended Model (**b**) Non-Extended Model

Figure 8. Paths taken by both models for Path 1 in one of the evaluation runs, as defined in Tables 2 and 3. The paths are shown in red. The extended model on the left demonstrates a more efficient navigation, while the non-extended model on the right shows significant deviations and exceeded the maximum allowed collisions without reaching the goal. The displayed maps are for visualization purposes only and were not used by the DRL algorithm or MoveBase for navigation.



(**a**) Extended Model (**b**) Non-Extended Model

Figure 9. Paths taken by both models for Path 2 in one of the evaluation runs, as defined in Tables 2 and 3. The paths are shown in red. The extended model on the left demonstrates a more efficient navigation, while the non-extended model on the right shows significant deviations and the test timed out before reaching the goal. The apparent overlap of the robot's path with the walls in the RViz visualization seen in (**a**) is a rendering artifact and does not represent any actual collisions during the robot's navigation. The displayed maps are for visualization purposes only and were not used by the DRL algorithm or MoveBase for navigation.

The extended model's ability to navigate non-straightforward paths is attributed to its training with randomized start and goal points, which fostered a broader exploration capability and adaptability to unexpected environments, ultimately resulting in higher success rates and reduced penalties during testing, The navigation performance can be viewed at [30].

Table 2. Test evaluation results for the extended model. Bold values indicate better performance compared to the non-extended model for each metric (Distance, Collisions, Goal Reached, and Time).

Path	Distance (m)	Collisions	Goal Reached	Time (s)
Path 1	**18.72**	**0**	**Yes**	**9.77**
Path 2	**47.16**	1	**Yes**	**31.66**
Path 3	**13.81**	**0**	**Yes**	**8.28**
Path 4	**26.41**	4	No	**21.79**
Path 5	**15.51**	**0**	**Yes**	**9.26**
Path 6	8.12	**0**	**Yes**	**4.19**
Path 7	**15.04**	3	**Yes**	9.87
Path 8	37.74	**0**	No	**35.49**
Path 9	**15.66**	1	**Yes**	**9.98**
Path 10	**30.54**	**0**	**Yes**	**23.33**

Table 3. Test evaluation results for the non-extended model. Bold values indicate better performance compared to the extended model for each metric (Distance, Collisions, Goal Reached, and Time).

Path	Distance (m)	Collisions	Goal Reached	Time (s)
Path 1	33.53	4	No	33.79
Path 2	41.97	**0**	No	80.00
Path 3	10.65	4	No	14.88
Path 4	39.40	**3**	No	67.59
Path 5	32.09	**0**	**Yes**	42.20
Path 6	8.12	**0**	**Yes**	4.24
Path 7	15.22	**3**	**Yes**	**9.55**
Path 8	**37.75**	**0**	No	39.67
Path 9	44.95	**0**	No	56.42
Path 10	58.16	**1**	No	68.19

6.3. Custom Navigation Performance Assessment Score Comparison

Evaluating navigation performance in real-world scenarios requires a metric that accounts for more than binary success or failure. The proposed custom navigation performance score metric better reflects real-world robot behavior by grading performance on a scale from 0% to 100%. This metric evaluates the robot's efficiency and safety in reaching its goal by considering three key factors: distance traveled, time taken, and the number of collisions. Unlike traditional binary metrics that label navigation attempts as either successful or failed, this score allows for partial successes, acknowledging scenarios where the robot reaches the goal despite minor inefficiencies or collisions. Such graded scoring is critical for realistic evaluations, as in practical applications, minor collisions or suboptimal paths may still be acceptable. By integrating these factors, the custom metric provides a more nuanced evaluation of robot performance, reflecting real-world decision-making scenarios where navigation is judged not only by reaching the goal but also by the efficiency and safety of the path taken. This approach complements traditional success metrics by offering deeper insights into the robot's adaptability and robustness. Each test starts with a base score of 100%, with penalties applied based on performance deviations:

$$\text{Score} = 100 - \text{total\_penalty}$$

where

$$\text{total\_penalty} = P_d + P_t + P_c \tag{4}$$

Each penalty term is defined as follows:

Distance Penalty (P_d): Applied if the path length exceeds twice the optimal distance. The distance penalty is proportional to the excess distance traveled over twice the optimal distance:

$$P_d = \left(\frac{\text{path\_length} - 2 \times \text{optimal\_distance}}{\text{max\_allowed\_distance} - 2 \times \text{optimal\_distance}} \right) \times 33 \quad (5)$$

where max_allowed_distance is four times the optimal distance.

Time Penalty (P_t): Applied if the actual navigation time exceeds 40 s. This penalty increases proportionally with time beyond 40 s:

$$P_t = \left(\frac{\text{actual\_time} - 40}{40} \right) \times 33 \quad (6)$$

Collision Penalty (P_c): Applied based on the number of collisions, with a penalty of 11% per collision, up to a maximum of three collisions:

$$P_c = \text{collision\_count} \times 11 \quad (7)$$

All penalties (P_d, P_t, and P_c) are individually capped at a maximum of 33%.

In this scoring system, a score of 0% is assigned if the robot fails to reach the goal, exceeds four times the optimal distance, incurs more than three collisions, or exceeds the maximum allowed navigation time of 80 seconds. Tables 4 and 5 show that the extended model consistently achieved higher scores, demonstrating more efficient navigation, fewer collisions, and superior adaptability. This is further illustrated in Figures 8 and 9, which compare the trajectories of the extended and non-extended models for Paths 1 and 2 using RViz visualization. The figures highlight how the extended model navigates more efficiently with fewer collisions and smoother trajectories, while the non-extended model struggles with significant deviations and higher collision rates during navigation.

Table 4. Navigation performance assessment metric scores for the extended model.

Path	Metric (%)	Description
Path 1	100.00	No collisions, within max allowed distance
Path 2	57.60	1 collision, exceeded optimal distance with penalty
Path 3	100.00	No collisions, optimal path followed
Path 4	0.00	4 collisions, exceeded max allowed distance
Path 5	100.00	No collisions, within optimal range
Path 6	100.00	No collisions, optimal path
Path 7	67.00	3 collisions, distance exceeded slightly
Path 8	0.00	Exceeded max allowed distance
Path 9	70.00	1 collision, slight penalty
Path 10	98.33	No collisions, distance exceeded slightly

Table 5. Navigation performance assessment metric scores for the non-extended model.

Path	Metric (%)	Description
Path 1	0.00	4 collisions, exceeded max allowed distance
Path 2	0.00	Timeout, exceeded max distance
Path 3	0.00	4 collisions, failed navigation
Path 4	0.00	Exceeded max allowed distance with 3 collisions
Path 5	85.77	No collisions, slight distance and time penalties
Path 6	100.00	No collisions, optimal path followed
Path 7	10.00	3 collisions, slight penalty
Path 8	0.00	Exceeded max allowed distance
Path 9	0.00	Exceeded max allowed distance, timeout
Path 10	0.00	Exceeded max allowed distance with 1 collision

7. Conclusions and Recommendations

The evaluation of both extended and non-extended environments reveals a significant research gap in robot navigation using deep reinforcement learning (DRL). While models tend to perform well in familiar environments, they often struggle to generalize to unseen environments, highlighting the need to improve the adaptability and robustness of DRL-based navigation systems.

The key insight from this study is that robust generalization is better achieved by training with varied and dynamic start and goal points rather than increasing the number of training scenarios, as done in the non-extended model. This diversity allows the robot to adapt more effectively to unforeseen environments, a perspective not often emphasized in recent studies.

In terms of performance, the non-extended model demonstrates an advantage in navigating extremely narrow environments, likely due to specific training for such conditions. However, it struggles with unpredictable goal points, even in simpler environments. In contrast, the extended model, which emphasizes start and goal diversity, adapts well to unpredictable navigation tasks but may face challenges in extremely narrow environments. This highlights the complementary strengths of both approaches.

To address the concern regarding training time, retraining a pre-trained model instead of training from scratch can significantly reduce the time required to achieve convergence. Additionally, as shown in Figure 4, the extended model achieved a higher success rate in a shorter time compared to the non-extended model. However, this difference in time efficiency cannot be solely attributed to the superiority of the extended model, as the training environments and setups for the two models were fundamentally different. The non-extended model faced environments with extremely narrow objects, which may have hindered its ability to achieve higher success rates. In contrast, the extended model focused on improving generalization through randomized goal positions, prioritizing adaptability to unseen environments. Future experiments could involve training the extended model in scenarios with extremely narrow objects, similar to those encountered by the non-extended model, to conclusively evaluate its time efficiency under such conditions. However, this may require reducing the variety of randomization to effectively incorporate narrow and extremely narrow objects while maintaining valid zones for random start and goal points.

The baseline comparison highlights key differences in DRL-based navigation approaches. Cimurs et al. [19] achieved the highest success rate in structured static environments, while Akmandor et al.'s method [13], which relied on a local strategy, exhibited lower performance in similar conditions. The proposed method demonstrated adaptability in a maze-like static environment with long walls, achieving a balance between success rate and navigation efficiency. These results emphasize the impact of different training conditions and obstacle structures on navigation performance, reinforcing the need for diverse test environments to assess generalization and adaptability.

The experiments were conducted in a simulated environment using ROS and the Gazebo simulator, which provides realistic physics-based interactions and real-time simulation to closely approximate real-world conditions. However, certain limitations must be acknowledged. Simulated environments cannot fully capture real-world variability, including terrain differences, dynamic obstacles, and sensor noise. While the evaluation tested generalization in unseen simulated environments, further validation in diverse real-world scenarios is necessary. Future work should focus on deploying the model on a physical robot to assess its real-world performance. Specifically, testing in structured indoor spaces and unstructured outdoor environments, incorporating both static and dynamic obstacles, would provide deeper insights into the model's adaptability and robustness. Additionally, experiments should account for real-world uncertainties, such as sensor

inaccuracies, environmental disturbances, and unexpected object movements, to ensure the model's practical applicability.

We recommend that future work combines the strengths of both models. Training should include diverse start and goal points, as in this extension, while also incorporating narrow and extremely narrow environments to enhance performance in highly constrained spaces. Additionally, retraining a pre-trained model can reduce training time, providing a practical approach to addressing the additional training time needed when integrating these elements.

Author Contributions: Conceptualization, R.A.; methodology, R.A.; software, R.A.; validation, S.D. (primary), L.M. and M.C.; formal analysis, R.A.; investigation, R.A. and S.D.; resources, R.A.; data curation, R.A.; writing—original draft preparation, R.A.; writing—review and editing, R.A., S.D. (primary), L.M. and M.C.; visualization, R.A. and S.D.; supervision, S.D. (lead), L.M. and M.C.; project administration, R.A.; coordination and overall review, S.D.; English language review, S.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Politecnico di Torino. The Article Processing Charge (APC) was partially covered by the PhD fund Additional financial support was provided by the AM2R project, "Mobilizing Agenda for Business Innovation in the Two Wheels Sector", funded by the PRR–Recovery and Resilience Plan and the Next Generation EU Funds, under reference C644866475-00000012 | 7253.50_DIM37_ALI.

Data Availability Statement: The datasets and code supporting the findings of this study are publicly available in the Extended-ROS-Jackal-Environment repository on GitHub at: https://github.com/Romisaa-Ali/Extended-ROS-Jackal-Environment (accessed on 28 March 2025).

Acknowledgments: The authors express their sincere gratitude to the SCUDO Office at Politecnico di Torino for their vital support and assistance throughout this research. Special thanks are extended to REPLY Concepts company for their valuable contributions and collaboration. The authors also acknowledge the Department of DAUIN at Politecnico di Torino and the team at PIC4SeR Interdepartmental Centre for Service Robotics—www.pic4ser.polito.it (accessed on 28 March 2025) for their continued support and cooperation. Further appreciation is given to the Electrical and Computer Engineering Department of the University of Coimbra and the Institute for Systems and Robotics (ISR-UC) for their collaboration and assistance.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

Abbreviation	Definition
TD3	Twin Delayed Deep Deterministic Policy Gradient
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
PPO	Proximal Policy Optimization
SAC	Soft Actor-Critic
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
ROS	Robot Operating System
XAI	Explainable Artificial Intelligence
CP	Collision Probability
NeurIPS	Conference on Neural Information Processing Systems
GDAE	Goal-Driven Autonomous Exploration
Tentabot FC	Fully Connected Neural Network Model in Tentabot Framework
Tentabot 1DCNN FC	1D Convolutional Neural Network Model with Fully Connected Layers in Tentabot Framework
OPAC	Opportunistic Actor-Critic

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2018; ISBN 978-0-262-03924-6. Available online: https://mitpress.mit.edu/9780262039246/reinforcement-learning/ (accessed on 24 March 2025).
2. Dogru, S.; Marques, L. An improved kinematic model for skid-steered wheeled platforms. *Auton. Robot.* **2021**, *45*, 229–243. [CrossRef]
3. Chen, Y.; Rastogi, C.; Norris, W.R. A CNN-Based Vision-Proprioception Fusion Method for Robust UGV Terrain Classification. *IEEE Robot. Autom. Lett.* **2021**, *6*, 7965–7972. [CrossRef]
4. Lapan, M. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*; Packt Publishing Ltd.: Birmingham, UK, 2018; ISBN 978-1-78883-930-2.
5. Hafner, D.; Lillicrap, T.; Norouzi, M.; Ba, J. Mastering Atari with Discrete World Models. *arXiv* **2020**, arXiv:2010.02193. [CrossRef]
6. Roy, S.; Bakshi, S.; Maharaj, T. OPAC: Opportunistic Actor-Critic. *arXiv* **2020**, arXiv:2012.06555. [CrossRef]
7. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602. [CrossRef]
8. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous Control with Deep Reinforcement Learning. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016. [CrossRef]
9. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of the International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; pp. 1861–1870. [CrossRef]
10. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
11. Roth, A.M. JackalCrowdEnv. GitHub Repository. 2019. Available online: https://github.com/AMR-/JackalCrowdEnv (accessed on 24 March 2025).
12. Roth, A.M.; Liang, J.; Manocha, D. XAI-N: Sensor-Based Robot Navigation Using Expert Policies and Decision Trees. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September 2021–1 October 2021; pp. 2053–2060. [CrossRef]
13. Akmandor, N.Ü.; Li, H.; Lvov, G.; Dusel, E.; Padir, T. Deep Reinforcement Learning Based Robot Navigation in Dynamic Environments Using Occupancy Values of Motion Primitives. In Proceedings of the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 23–27 October 2022; pp. 11687–11694. [CrossRef]
14. Akmandor, N.U.; Dusel, E. Tentabot: Deep Reinforcement Learning-Based Navigation. GitHub Repository. 2022. Available online: https://github.com/RIVeR-Lab/tentabot/tree/master (accessed on 24 March 2025).
15. Ali, R. Robot exploration and navigation in unseen environments using deep reinforcement learning. *World Acad. Sci. Eng. Technol. Int. J. Comput. Syst. Eng.* **2024**, *18*, 619–625.
16. Xu, Z.; Liu, B.; Xiao, X.; Nair, A.; Stone, P. Benchmarking Reinforcement Learning Techniques for Autonomous Navigation. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 9224–9230. [CrossRef]
17. Fujimoto, S.; van Hoof, H.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; Stockholmsmässan: Stockholm, Sweden, 2018; Volume 80, pp. 1582–1591. [CrossRef]
18. Cimurs, R.; Suh, I.H.; Lee, J.H. Goal-Driven Autonomous Exploration Through Deep Reinforcement Learning. *IEEE Robot. Autom. Lett.* **2022**, *7*, 730–737. [CrossRef]
19. Cimurs, R. DRL-Robot-Navigation. GitHub Repository. 2024. Available online: https://github.com/reiniscimurs/DRL-robot-navigation (accessed on 24 March 2025).
20. Anas, H.; Hong, O.W.; Malik, O.A. Deep Reinforcement Learning-Based Mapless Crowd Navigation with Perceived Risk of the Moving Crowd for Mobile Robots. *arXiv* **2023**, arXiv:2304.03593. [CrossRef]
21. Zerosansan. TD3, DDPG, SAC, DQN, Q-Learning, SARSA Mobile Robot Navigation. GitHub Repository. 2024. Available online: https://github.com/zerosansan/td3_ddpg_sac_dqn_qlearning_sarsa_mobile_robot_navigation (accessed on 24 March 2025).
22. Wong, C.-C.; Chien, S.-Y.; Feng, H.-M.; Aoyama, H. Motion Planning for Dual-Arm Robot Based on Soft Actor-Critic. *IEEE Access* **2021**, *9*, 26871–26885. [CrossRef]
23. Sylabs. Installing SingularityCE. 2024. Available online: https://docs.sylabs.io/guides/latest/admin-guide/installation.html#installation-on-linux (accessed on 24 March 2025).
24. Daffan, F. ros_jackal. GitHub Repository. 2021. Available online: https://github.com/Daffan/ros_jackal (accessed on 24 March 2025).
25. Ali, R. Extended-ROS-Jackal-Environment. GitHub Repository. 2024. Available online: https://github.com/Romisaa-Ali/Extended-ROS-Jackal-Environment (accessed on 24 March 2025).

26. Xu, Z.; Liu, B.; Xiao, X.; Nair, A.; Stone, P. Benchmarking Reinforcement Learning Techniques for Autonomous Navigation. Available online: https://cs.gmu.edu/~xiao/Research/RLNavBenchmark/ (accessed on 24 March 2025).
27. Daffan, F. ROS Jackal: Competition Package. GitHub Repository. 2023. Available online: https://github.com/Daffan/ros_jackal/tree/competition (accessed on 24 March 2025).
28. Clearpath Robotics. Simulating Jackal in Gazebo. 2024. Available online: https://docs.clearpathrobotics.com/docs/ros1noetic/robots/outdoor_robots/jackal/tutorials_jackal/#simulating-jackal (accessed on 24 March 2025).
29. Open Source Robotics Foundation. move_base. ROS Wiki. n.d. Available online: https://wiki.ros.org/move_base (accessed on 24 March 2025).
30. Zenodo. Robot Navigation Using TD3 with MoveBase Integration in ENV2. February 2025. Available online: https://zenodo.org/records/14881795 (accessed on 24 March 2025).