Highlights

Optimizing Path Planning using Deep Reinforcement Learning for UGVs in Precision Agriculture

Laukik Patade,Rohan Rane,Sandeep Pillai,Sujata Kulkarni,Kiran Talele

- This study presents a novel approach to optimizing path planning for Unmanned Ground Vehicles (UGVs) in precision agriculture, employing deep reinforcement learning (DRL) techniques in both discrete and continuous action spaces.
- Traditional path planning methods like A* and Dijkstra's algorithms are compared with DRL techniques, highlighting the latter's superior adaptability in dynamic agricultural environments.
- The research explores both 2D and 3D environments, with 2D simulations conducted using Pygame and 3D experiments carried out in ROS and Gazebo, where DRL models like Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), and Twin Delayed Deep Deterministic Policy Gradient (TD3) are evaluated.
- DRL models, particularly those handling continuous action spaces, demonstrated significant improvements in path efficiency, obstacle avoidance, and real-time adaptability in complex, dynamically changing agricultural fields.
- The results emphasize the importance of continuous learning algorithms for UGVs in agricultural tasks, allowing more efficient navigation and enhancing performance over traditional algorithms.
- This study is a part of the larger project "Drones for Smart Agriculture: An Efficient Method for Pesticide Application to Diseased Crops", funded by IEEE, and contributes specifically to optimizing UGVs for pesticide spraying tasks in real-world agricultural scenarios.

Optimizing Path Planning using Deep Reinforcement Learning for UGVs in Precision Agriculture

Laukik Patade, Rohan Rane, Sandeep Pillai, Sujata Kulkarni and Kiran Talele

Sardar Patel Institute of Technology, Bhavan's Campus, Andheri West, Mumbai, 400058 MH, India

ARTICLE INFO

Keywords:
Unmanned Ground Vehicle (UGV)
Precision Agriculture
Path Planning
Deep Reinforcement Learning (DRL)
Continuous Action Space
Deep Q-Network (DQN)
Policy Gradient
ROS
Gazebo

ABSTRACT

This study focuses on optimizing path planning for Unmanned Ground Vehicles (UGVs) in precision agriculture through deep reinforcement learning (DRL) techniques in continuous action spaces. The research begins by reviewing traditional grid-based methods, such as A* and Dijkstra's algorithms, and highlights their limitations in dynamic agricultural environments, underscoring the need for adaptive learning strategies. Transitioning to DRL approaches, the study explores methods like Deep Q-Networks (DQN), which demonstrate improved adaptability and performance in 2D simulations. Enhancements, including Double Q-Networks and Dueling Networks, are evaluated to further refine decision-making. Building on these insights, the focus shifts to continuous action space models, specifically Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), which are tested in increasingly complex environments. Experiments conducted in a 3D environment using ROS and Gazebo showcase the effectiveness of continuous DRL algorithms in navigating dynamic agricultural challenges. Notably, the pretrained TD3 agent achieves a 95% success rate in dynamic environments, demonstrating the robustness of the proposed approach in handling moving obstacles while ensuring no harm to crops or the robot.

1. Statement and Declarations

This research is part of the project titled "Drones for Smart Agriculture: An Efficient Method for Pesticide Application to Diseased Crops" funded by the IEEE Aerospace and Electronic Systems Society (AESS) under the Distributed Sensor Technology and Education Initiative (DSTEI), with a grant of $25,000. The project comprises three modules:

1. **UAV Module:** A drone monitors crops by capturing images and videos over the agricultural landscape.
2. **Image Processing Module:** Captured data is processed at a ground station to detect crop diseases and relay location information to the ground robot.
3. **UGV Module:** The ground robot receives the location data, plans a path, and executes targeted pesticide spraying.

This paper focuses on optimizing the path-planning capabilities of the ground robot, ensuring:

1. No damage to the robot.
2. No harm to surrounding crops.
3. Minimized vulnerability to external factors.

The results contribute directly to the highlighted section in the fig. 1 module diagram of the overall project.

2. Introduction

Robotics and Artificial Intelligence (AI) are transforming automation across diverse domains, unlocking unprecedented opportunities for innovation. Among these, the advancements in unmanned systems—particularly Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs)—have been remarkable. Leveraging these advancements, this paper focuses on optimizing the design of UGVs for agricultural applications, with an emphasis on robust

laukikpatade22@gmail.com (L. Patade); ryrane.in@gmail.com (R. Rane); sandeepxpillai@gmail.com (S. Pillai); sujata_kulkarni@spit.ac.in (S. Kulkarni); kiran_talele@spit.ac.in (K. Talele)
ORCID(s):

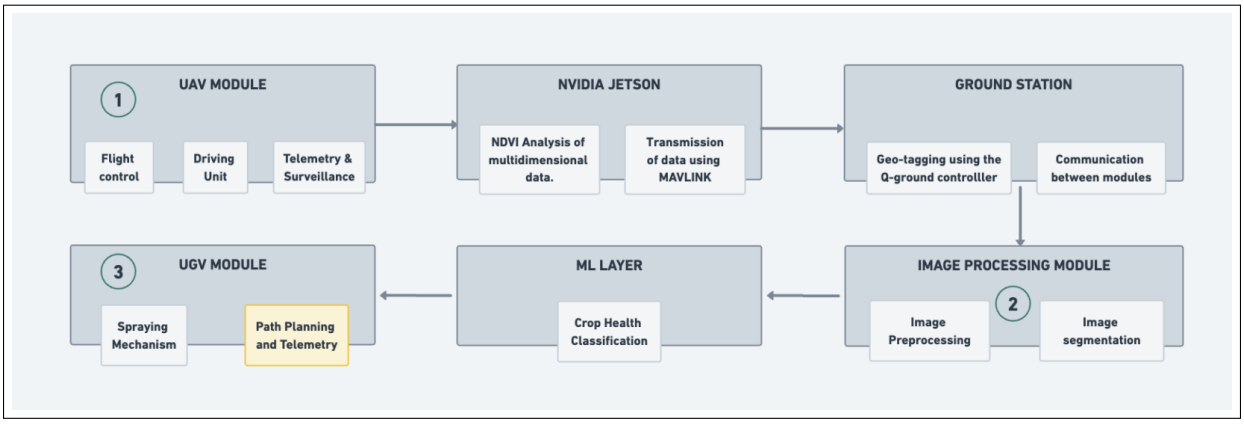


Figure 1: Drones for Smart Agriculture - Module Diagram

path-planning algorithms driven by Deep Reinforcement Learning (DRL). UGVs play a critical role in various fields, including defense, surveillance (Gadekar et al., 2023), and precision agriculture, where they contribute to operational efficiency and sustainability.

Precision agriculture has significantly benefited from AI-driven technologies that enhance resource allocation and crop management. Studies such as Talaviya et al. (2020) underscore how innovations like drones, robotics, and soil sensors optimize irrigation, pest control, and soil preservation, promoting sustainability and improving crop quality. For example, AI-powered systems that dynamically adjust pesticide dosages based on real-time assessments have minimized chemical usage while maximizing pest management efficiency (Tewari et al., 2020).

Building on these advancements, our research proposes an adaptive path-planning framework for UGVs operating in agricultural landscapes. This system is designed to complement UAVs, which capture high-resolution crop images for analysis at a central ground station. The processed data informs the UGV's path planning to precisely navigate to diseased crop areas and apply targeted treatments. Such integrated systems have been successfully built before (Mammarella et al., 2022) and we intend to enhance the performance of UGV subsystems to ensure smooth integration and resource optimization. This would encompass efforts centered exclusively around optimizing UGV navigation and decision-making through DRL.

Traditional path-planning algorithms like A* have proven inadequate in handling the dynamic and uncertain nature of agricultural environments. To address these challenges, this paper investigates the application of DRL as a robust alternative. DRL-based methods allow UGVs to learn and adapt to real-world complexities, such as obstacle-rich terrains and unpredictable conditions. For instance, OrtataŞ et al. (2023) demonstrated the utility of machine learning in autonomous navigation, showcasing a GPS-independent UGV equipped with stereo cameras for robust obstacle detection in unstructured settings. Such advancements highlight the potential of ML-based approaches to overcome limitations inherent in conventional path-planning techniques.

Moreover, recent innovations in DRL-based path planning, including prioritized experience replay and expert-driven learning (Liu et al., 2024b), have further enhanced the efficiency and adaptability of autonomous systems. These methods enable UGVs to optimize navigation strategies dynamically, improving task efficiency in challenging agricultural environments. This study advances DRL-based path-planning techniques for UGVs to tackle challenges in precision agriculture. By improving autonomous navigation, it contributes to developing intelligent, efficient, and sustainable agricultural systems.

3. Related Work

Path planning for mobile robots, including Unmanned Ground Vehicles (UGVs), has been widely researched, exploring various techniques for navigating complex environments. Traditional methods like A* and Rapidly-exploring Random Trees (RRT) are commonly used but face challenges in dynamic settings like precision agriculture. For instance, A* is efficient in static environments (Zhi and Xiumin, 2019), but struggles with dynamic changes and moving obstacles, which are common in agricultural fields. Frequent path re-computation adds computational overhead, making A* less practical for real-time, dynamic scenarios.

Similarly, RRT-based approaches offer flexibility by exploring the search space randomly (Li, 2021), but require recalculations when new obstacles arise. While effective in cluttered environments, they struggle to adapt in precision agriculture, where dynamic changes are frequent. Although RRT has been extended for dynamic environments, its need for continuous replanning limits its efficiency in real-time scenarios.

To address these limitations, recent research has shifted towards Deep Reinforcement Learning (DRL), enabling continuous adaptation without repetitive recalculations. The Deep Q-Network (DQN) algorithm (Mnih et al., 2015a) introduced a significant advance by allowing learning from high-dimensional inputs like raw sensory data, crucial in precision agriculture where data is often noisy or incomplete. DQN's experience replay stabilizes learning by reducing correlations between consecutive observations, enabling UGVs to adapt to real-time changes. Chu et al. (2023) applies a double deep Q-network (DDQN) to underwater vehicle navigation, showing effective adaptation in dynamic environments.

Further improving real-time path planning, Yan et al. (2020) explored Dueling Double Deep Q-Networks (D3QN) for UAV path planning in dynamic environments. This approach incorporates a situation assessment model to map global threats and a hybrid action selection policy, outperforming DDQN and DQN in cumulative rewards and safe path generation. This demonstrates DRL's potential for dynamic path planning, not only in UAVs but also in UGVs in similar environments.

DRL has also been applied to multi-robot coordination. Double Q-learning with n-step returns has been used in multi-robot systems in partially known environments, reducing path errors and enhancing efficiency (Yang et al., 2020). Studies have also explored DRL for emergency vehicles navigating congested urban roads (Yan et al., 2021), highlighting DRL's adaptability across different applications. This research emphasizes road architecture and vehicle queue dynamics, showcasing DRL's potential for planning in complex traffic scenarios. A review by Adzhar et al. (2020) highlights the advantages of machine learning approaches over classical methods in dynamic environments.

For more complex scenarios, actor-critic methods like Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3) have been applied in continuous action spaces. These techniques are useful in precision agriculture, where environments require continuous actions. For example, Josef and Degani (2020) applied soft actor-critic methods for local path planning on rough terrain, enabling vehicles to navigate uneven landscapes. Such capabilities are crucial for agricultural UGVs, which must avoid dynamic obstacles. SAC has also been used to enhance training efficiency and handle sparse reward challenges (Zhao et al., 2023). In another study, DDPG was applied for rough terrain navigation, allowing UGVs to make real-time decisions based on sensory inputs (Guo et al., 2020).

(Chen et al., 2022) shows that combining SAC with Prioritized Experience Replay (PER) is highly effective for real-time path planning and dynamic obstacle avoidance. This approach outperforms DDPG and A3C in path smoothness, success rates, and computational efficiency. Additionally, (Deshpande et al., 2024) proposed integrating DDPG with a Differential Gaming (DG) strategy for exploration in path planning. The DG approach ensures collision-free target reaching by incorporating positive learning episodes into the memory buffer. This strategy improved success rates, reduced collisions, and increased convergence rates, demonstrating the effectiveness of combining exploration strategies like DG with DRL methods for dynamic path planning.

In summary, while classical algorithms like A* and RRT have laid the groundwork for UGV path planning, they struggle in dynamic and unpredictable environments. In contrast, DRL algorithms like DQN, SAC, and actor-critic methods like TD3 offer the adaptability and precision needed for agricultural applications. By enabling continuous learning and adaptation, DRL methods effectively address dynamic obstacle avoidance and real-time decision-making, advancing UGV path planning for precision agriculture.

4. Methodology

The experimentation is divided into three distinct stages(fig. 2) to thoroughly analyze the performance of deep reinforcement learning techniques for unmanned ground vehicles (UGVs) in the context of precision agriculture. Each stage systematically explores different action space representations and environmental setups, starting with simpler discrete environments and gradually progressing to more complex and realistic agricultural simulations. This structured approach as outlined in table 1 allows for a comprehensive evaluation of the algorithms' effectiveness and adaptability, as they face varying challenges that closely resemble real-world scenarios encountered in agricultural tasks.

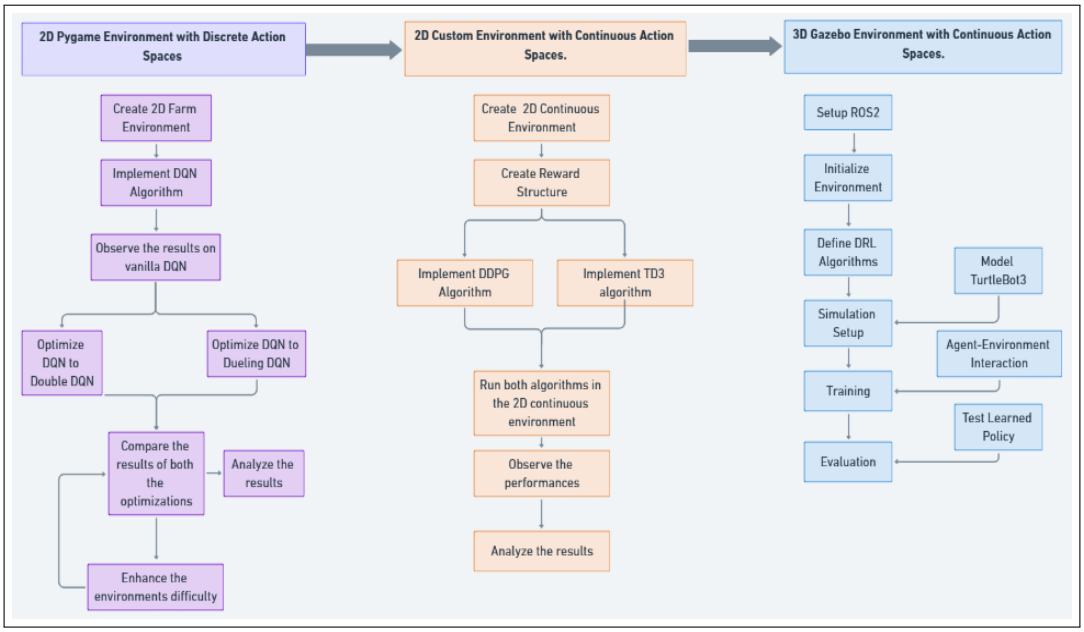


Figure 2: Methodology Diagram

Table 1
Experiment Stages in Path Planning for UGV

Stage	Environment Setup	Algorithm Implementation	Observations
Stage 1: DQN in Discrete Action Spaces	2D grid-based environment with random obstacles simulating agricultural terrain.	DQN for path planning, neural network approximates Q-value function.	Track episode length, rewards, and convergence speed. Compare DQN, Double DQN, and Dueling DQN.
Stage 2: Continuous Action Spaces	Extend 2D grid to continuous movement, allow varying speeds and angles.	DDPG and TD3 (Actor-Critic methods) for continuous action space.	Evaluate rewards, episode length, and training stability.
Stage 3: Realistic Agricultural Simulation	3D agricultural environment in Gazebo with crops, rocks, and uneven terrain.	Integrate optimized model in Gazebo for path planning.	Track success rate, rewards, and training loss. Evaluate model's effectiveness in a real scenario.

5. Experimentation - Discrete Action Spaces

5.1. Deep Q-Network

5.1.1. Environment

The environment is built on top of the existing FrozenLake - v1 environment (Farama Foundation) of the gymnasium package. A replica of a 2D grid-based agricultural landscape is constructed as shown in fig. 3.

There are 4 entities within the environment, namely:

1. The agent
2. Obstacles
3. Goal
4. Free space

The agent can be in 64 states and can take 4 actions (UP, DOWN, RIGHT, LEFT).

Figure 3: 2D Farm Environment in Pygame

5.1.2. Reward Structure

The reward structure utilized in this experiment mirrors that of the OpenAI Gym environment, which is characterized as a sparse reward structure. This type of reward shaping assigns rewards infrequently, primarily focusing on significant milestones rather than providing continuous feedback for every action taken. The rewards are assigned as follows:

- Reaching the goal crop: +1
- Colliding with an obstacle: 0 (terminates the episode)
- Reaching an open field: 0

In this context, the maximum achievable reward for each episode is 1. It should be noted that the robot is not penalized for its actions; instead, episodes are terminated when the robot is struck by an obstacle. The designation of "sparse" arises from the limited number of reward signals provided, which encourages the agent to explore the environment more extensively before receiving feedback.

5.1.3. Deep Q-Network (DQN) Architecture

The Deep Q-Network (DQN) (Mnih et al., 2015b) extends Q-learning by using a neural network to approximate Q-values, aiming to maximize cumulative reward through balanced exploration and exploitation.

Policy Network (Q-network) The policy network, $Q(s, a|\theta)$, estimates Q-values for actions in a given state s and is updated via gradient descent. Actions are selected using epsilon-greedy: with probability ϵ, a random action is chosen (exploration); otherwise, the action with the highest Q-value is selected (exploitation).

Target Network The target network $Q'(s, a|\theta^-)$ is a periodically updated copy of the policy network with weights θ^-. This target network stabilizes training by providing consistent Q-values for target calculations. Instead of updating with every policy change, the target network is updated less frequently, helping to smooth out learning and prevent oscillations.

Replay Memory In replay memory, transitions $(s_t, a_t, r_t, s_{t+1}$, terminated) are stored in a buffer. Mini-batch sampling from this buffer breaks the sequential correlations between transitions. By enabling the agent to learn from a diverse set of past experiences, replay memory enhances generalization and robustness.

Epsilon-Greedy Policy The epsilon-greedy policy manages the trade-off between exploration and exploitation. With probability ϵ, the agent selects a random action, promoting exploration of new strategies. Over time, ϵ decays, encouraging the agent to increasingly favor the optimal action and leverage its learned knowledge to maximize

performance. This controlled decay in ϵ allows for systematic exploration in the initial stages and refined action selection later on.

Bellman Equation and Loss Function The target Q-value y_i is computed as:

$$y_i = \begin{cases} r_i & \text{if terminal,} \\ r_i + \gamma \max_a Q'(s_{i+1}, a|\theta^-) & \text{otherwise.} \end{cases}$$

The Bellman equation describes this recursive target relationship, capturing how expected rewards propagate over time. The loss $L(\theta)$, calculated as the mean squared error between y_i and $Q(s_i, a_i|\theta)$, minimizes the difference between predicted and target values, allowing the network to more accurately estimate the optimal Q-values.

Experience Replay Experience replay further stabilizes training through random mini-batch sampling from stored transitions in replay memory. By allowing the agent to revisit past experiences, it mitigates the impact of highly correlated and sequential data. This process enables better representation of the state-action space by leveraging a broad array of past interactions.

Target Network Update The target network periodically receives updated weights from the policy network, $\theta^- \leftarrow \theta$, allowing it to reflect newer learning without the instability caused by frequent updates. This delayed update provides a consistent target for multiple steps, enhancing stability by smoothing the learning trajectory.

This approach encourages the agent to explore the environment through random actions in the initial episodes, balancing exploration and learning. As the episodes progress and ϵ decays, the agent focuses on maximizing cumulative rewards by leveraging the knowledge gathered from earlier exploration.

5.1.4. Neural Network Architecture

The Deep Q-Network (DQN) consists of the following key elements:

- **Input dimensions**: Match the environment's state space, representing the robot's current situation in terms of position, orientation, or sensor data. For an 8x8 grid environment, the input is a binary representation of the 64 possible states.
- **Output dimensions**: Correspond to the action space, with each output providing a Q-value for one possible action. In the 8x8 grid, the output includes Q-values for all four state-action pairs associated with the current state.(UP, LEFT, RIGHT, BOTTOM)
- **Input to the network**: The current state, represented as a feature vector (binary for the 8x8 grid), enables the DQN to calculate Q-values based on the robot's environment.
- **Output of the network**: Provides Q-values for all actions, with the highest Q-value corresponding to the action selected during decision-making.
- **Activation function**: ReLU (Rectified Linear Unit) is used in hidden layers to introduce non-linearity, enhancing the network's capacity to learn complex patterns.

The network as shown in fig. 4 consists of an input layer, two hidden layers with ReLU activations, and an output layer. Both the policy and target networks share this architecture, and the replay buffer stores the agent's past experiences, which are later sampled for training. The hyperparameters for both, the policy and the target network are tabulated in table 2.

5.1.5. Performance

The neural network is trained to optimize the approximation of target Q-values for 10,000 steps, with evaluation based on the trailing rewards over the last 100 episodes. The model is considered to have converged when it consistently reaches the goal in all 100 consecutive episodes. The illustration fig. 5b shows that the agent reaches the goal in the minimum number of steps, selecting one of the optimal paths. However, similar results can be achieved by simple grid-based algorithms like A* and Dijkstra's algorithm. But these algorithms lack the capacity to learn or adapt to

Algorithm 1 Deep Q-Network (DQN) Algorithm

Initialize replay memory R with capacity N
Initialize policy network $Q(s, a|\theta)$ with random weights
Initialize target network $Q'(s, a|\theta^-)$ with weights copied from the policy network
Set learning rate α, discount factor γ, and exploration rate ϵ
for each episode **do**
    Reset the environment and get the initial state s_0
    for each step t in the episode **do**
        With probability ϵ, select a random action a_t
        Otherwise, select $a_t = \arg\max_a Q(s_t, a|\theta)$
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Store transition $(s_t, a_t, r_t, s_{t+1}, \text{terminated})$ in replay memory R
        if replay memory R has enough samples **then**
            Sample a mini-batch of transitions $(s_i, a_i, r_i, s_{i+1}, \text{terminated}_i)$ from R
            for each sampled transition **do**
                if terminated_i is True **then**
                    Set target $y_i = r_i$
                else
                    Set target $y_i = r_i + \gamma \max_a Q'(s_{i+1}, a|\theta^-)$
                end if
                Compute the loss:

$$L = \frac{1}{N} \sum_i \left(y_i - Q(s_i, a_i|\theta)\right)^2 \tag{1}$$

            end for
            Perform a gradient descent step to minimize the loss
        end if
        Update state $s_t \leftarrow s_{t+1}$
        if step t is a target update step **then**
            Update target network weights $\theta^- \leftarrow \theta$
        end if
    end for
    Decay exploration rate ϵ
end for

environmental dynamics, which DQN does. This can be demonstrated by introducing noise into the environment. The FrozenLake-v1 environment includes an external disturbance parameter called `is_slippery`, which when enabled, causes the agent to deviate from the most optimal action.

A significant difference is observed in fig. 6a during training in the noisy environment, where the agent took 40,000 episodes to achieve 80% convergence. This implies that the agent reaches the goal in 8 out of 10 attempts on average. Testing in the noisy environment:

fig. 6b demonstrates that the agent requires significantly more steps due to the `is_slippery` factor, which continuously pushes it off course. By learning the dynamics of the noisy environment, the agent plans its path along the boundary to avoid obstacles—a clear sign of adaptive intelligence. This is especially valuable in complex environments like agricultural fields, where many disturbances cannot be quantified. Unlike simple algorithms such as A* or RRT, which rely solely on predefined paths, DQN's ability to learn and adapt from experience makes it more effective in such dynamic landscapes.

5.2. Towards improving DQN

While the basic Deep Q-Network (DQN) algorithm provides a solid foundation for path planning in discrete environments, it is essential to implement further optimizations to enhance its performance in precision agriculture

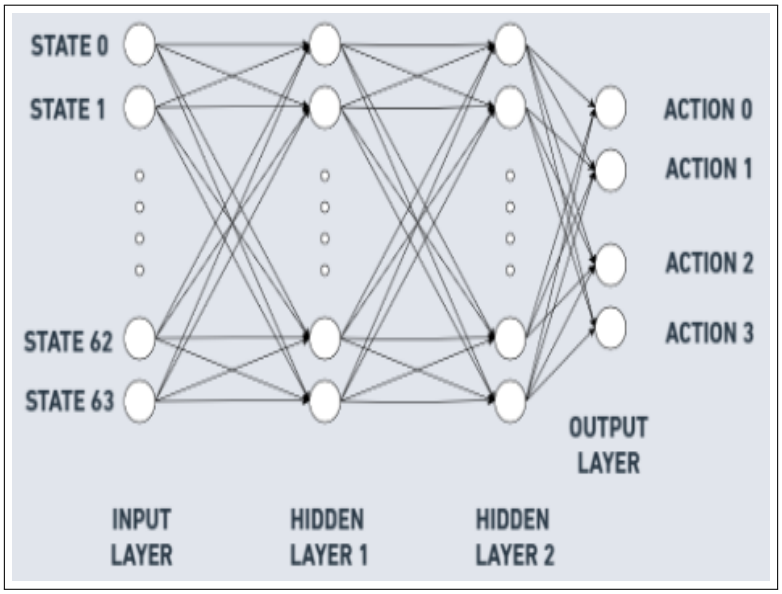


Figure 4: DQN Neural Network Architecture

Table 2
Hyperparameters - Deep Q Network (DQN)

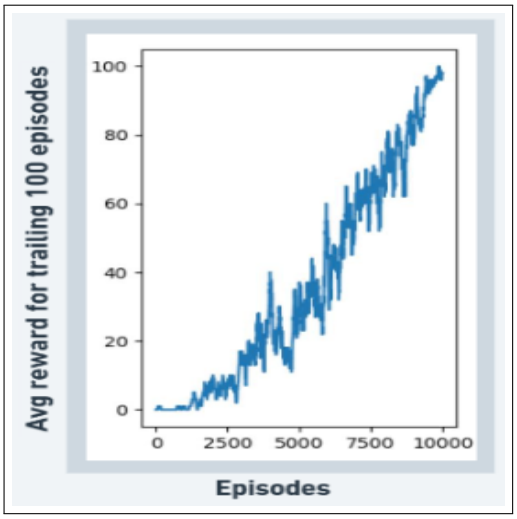
Hyperparameter	Value	Description
Learning Rate	0.001	Parameter update rate
Discount Factor (γ)	0.99	Future reward discount factor
Epsilon Decay Rate	0.995	Exploration decay rate
Minimum Epsilon (ϵ)	0.01	Minimum exploration rate
Replay Buffer Size	100,000	Max capacity for experiences
Batch Size	64	Samples per update
Target Network Update Freq.	1000	Steps between target updates

tasks. The goal of these optimizations is to improve the robustness, speed, and reliability of DQN-based Unmanned Ground Vehicle (UGV) navigation.

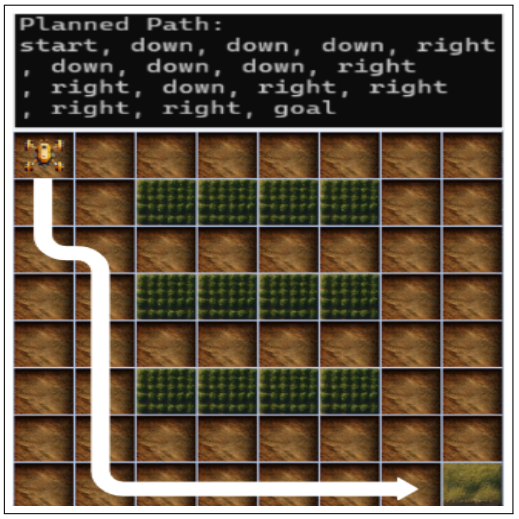
5.2.1. Double DQN

One significant improvement is the introduction of Double DQN (Hasselt et al., 2016), which addresses the issue of maximization bias inherent in the original DQN algorithm. In standard DQN, the action selection and action evaluation both happen with the target network $Q'(s_{i+1}, a'; \theta^-)$. This can lead to an overestimation of Q-values. It selects the action a' that maximizes Q', and then evaluates this action using the same network, which can cause over-optimistic estimates of the action's value. Double DQN mitigates this bias by simply using the policy network for selecting the best action and the target network to evaluate its Q-value, which results in a more stable and accurate update process. Action selection is done using the current policy network $Q(s_{i+1}, a'; \theta)$, but the evaluation of that action's value is done using the target network $Q'(s_{i+1}, a'; \theta^-)$. This decoupling reduces the risk of overestimating Q-values because the action selection and value evaluation are not based on the same function. In standard DQN, the target Q-value update rule is:

$$y_i^{DQN} = r_i + \gamma \max_{a'} Q'(s_{i+1}, a'; \theta^-)$$

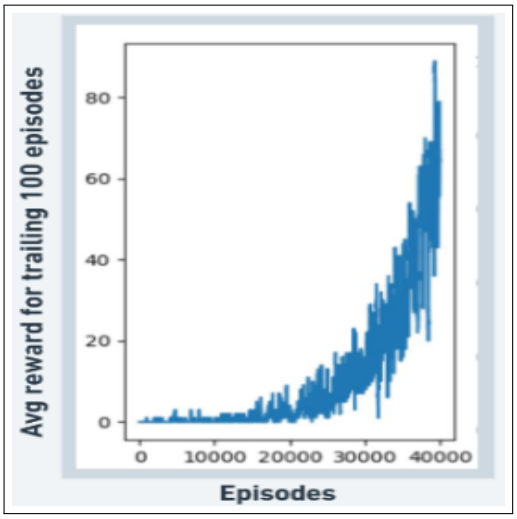


(a) Trailing 100 Rewards vs Episodes (DQN)

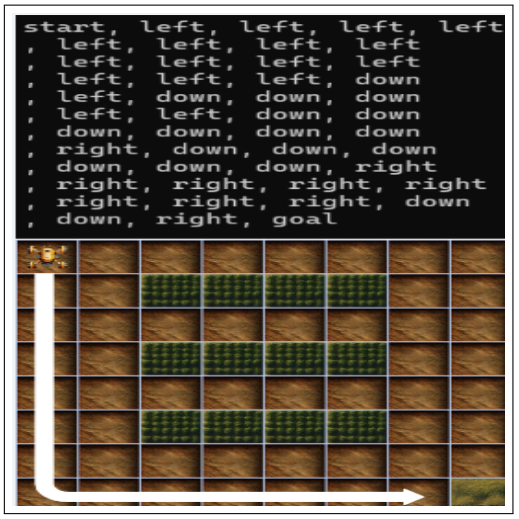


(b) Planned Path

Figure 5: Performance of DQN in 8X8 Environment



(a) Trailing 100 Rewards vs Episodes (Noisy Env)



(b) Planned Path in Noisy Environment

Figure 6: Performance of DQN in 8x8 Noisy Environment

In Double DQN, the update rule is modified to decouple action selection and action evaluation:

$$y_i^{DoubleDQN} = r_i + \gamma Q'(s_{i+1}, \arg\max_{a'} Q(s_{i+1}, a'; \theta); \theta^-)$$

Thus, instead of just selecting the state-action pair with the maximum Q value, we use the target network to evaluate the Q value and determine the optimal action. This leads to better Q-value estimation.

5.2.2. Dueling Networks

Dueling DQN (Wang et al., 2016) introduces a structural change (shown in fig. 7) where the estimation of the Q-value is split into two components: the value function and the advantage function. This separation helps the network learn more efficiently by focusing on two distinct aspects:

- **State Value** $V(s)$: This represents how valuable it is to be in a particular state, regardless of the action taken.

- **Advantage** $A(s, a)$: This represents how advantageous it is to take a specific action in a given state, relative to other actions.

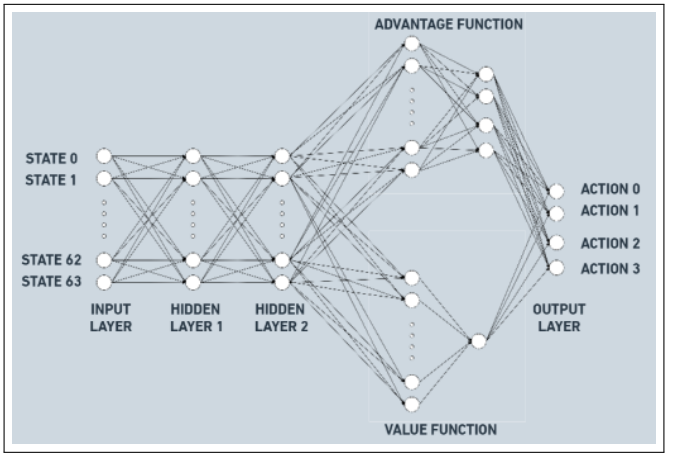


Figure 7: Dueling Networks Architecture

By splitting the network into these two streams, Dueling DQN allows the network to better evaluate which states are inherently valuable and which actions are preferable. This separation is particularly useful in environments where many actions may have similar outcomes in a given state, but certain states are more critical than others.

The Q-value is then computed by combining the value and advantage streams:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right)$$

This formulation helps reduce the impact of noisy or redundant action-value estimates by normalizing the advantage across actions. As a result, D3QN can more effectively prioritize learning the value of different states, even in situations where the action choices may not differ significantly. This modification enhances the agent's ability to differentiate between important states and actions, leading to improved performance, especially in complex environments where actions might not significantly vary in value across states.

5.3. Observations

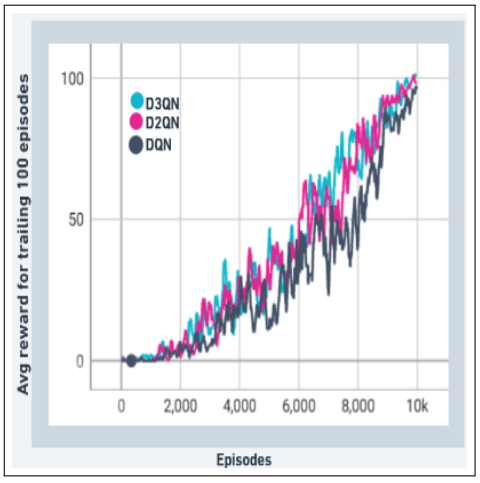
5.3.1. 8x8 Environment

Following the successful implementation of Double DQN and Dueling Double DQN, the environment was run on these new variants to observe their performance outcomes.

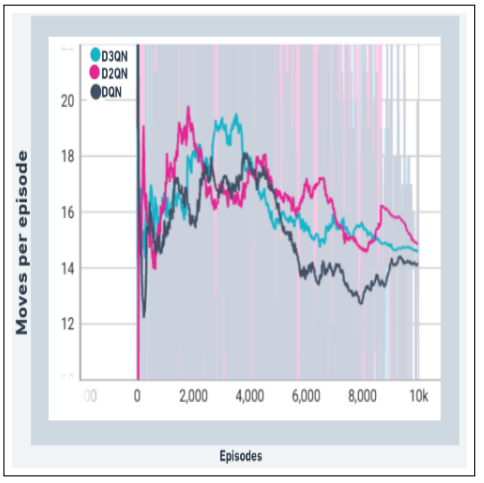
The analysis of fig. 8a and fig. 8b *(*smoothened by a factor of 0.99)* indicates that in a simple environment like the 8x8 agricultural map, all variants of the DQN algorithm demonstrate similar performance metrics. Notably, Dueling DQN exhibits slightly higher efficiency concerning the number of moves per episode, although the differences are not substantial. The straightforward nature of the map facilitates efficient learning for the DQN algorithm, enabling all optimized variants to reach comparable performance thresholds. Consequently, modifications to the environment are warranted, transitioning to a more complex 10x10 map to further evaluate the algorithms' capabilities.

5.3.2. 10x10 Environment

To thoroughly assess the optimization performance of the various algorithmic variants in a more complex environment compared to the simpler 8x8 grid, a new 10x10 grid (fig. 9a) was meticulously designed. This grid features a significantly increased number of obstacles, making the environment computationally more demanding and challenging for the agent. Despite this added complexity, all parameters within the map remain consistent with those used in the previous environment, with the sole alteration being the layout itself. This new configuration introduces more non-uniform obstacles, thereby enhancing the intricacy of the navigation task that the agent must successfully overcome.



(a) DQN vs D2QN vs D3QN - Trailing Rewards



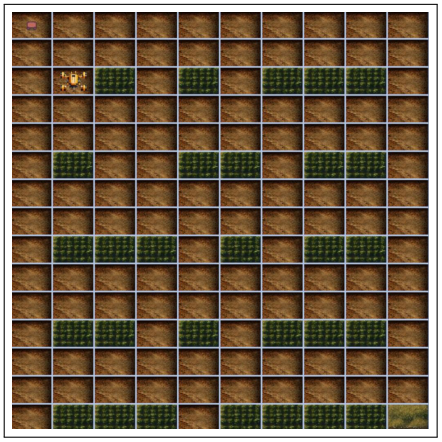
(b) DQN vs D2QN vs D3QN: Moves per Episode

Figure 8: Performance of DQN, D2QN & D3QN in 8x8 Environment

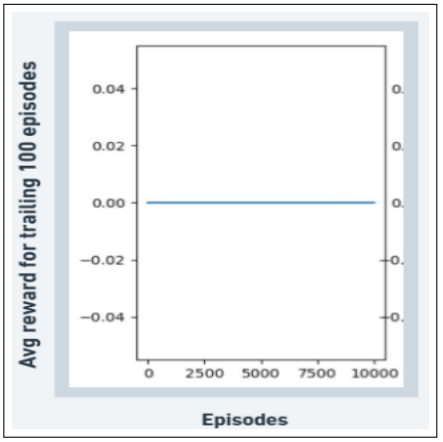
The following three critical parameters were fixed for this experiment, applicable to DQN, Double DQN, and Dueling DQN:

- Environment: 10x10 agricultural map
- Episodes: 10,000
- Learning Rate: 1e-3

DQN: Upon executing the DQN algorithm in the 10x10 map, it was observed that, despite the fixed episode count, the agent struggled to learn the map and achieve any rewards.



(a) 10x10 Map



(b) DQN Performance in 10x10 Map

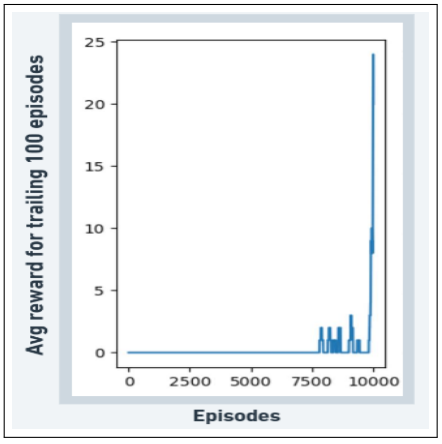
Figure 9: Visualization of the 10x10 map and corresponding DQN performance. (A) Map layout and (B) DQN results.

The agent continuously opted for random actions throughout the map, failing to reach its goal and, consequently, obtaining no rewards (as seen in fig. 9b). Thus, it is evident that DQN underperforms in environments with increased complexity.

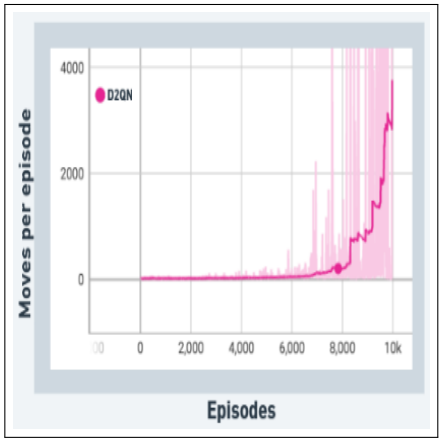
Double DQN: Using the double DQN algorithm to navigate the environment, a significant improvement over the traditional DQN approach was clearly observed. The agent demonstrated a marked enhancement in its ability to select

random actions strategically, allowing it to effectively reach its designated goal while simultaneously collecting rewards more efficiently.

It can be seen from fig. 10a that the double DQN algorithm is capable of accumulating intermittent rewards through numerous random actions; however, it has not fully learned the environment. A careful examination of fig. 10b *(* smoothed by a factor of 0.99)* reveals that, although the algorithm is nearing a learning threshold, the frequency of random actions taken remains significantly high. Hence, this approach is inefficient for path planning unless the episode count is increased, providing the algorithm with additional time to learn.



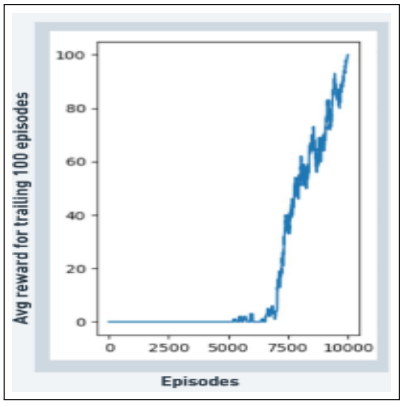
(a) Double DQN Performance in 10x10 Map



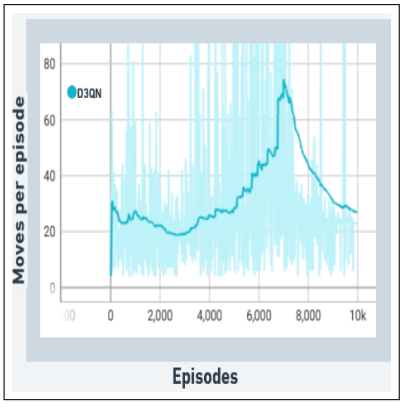
(b) Moves per Episode for Double DQN in 10x10 Map

Figure 10: Double DQN Performance in the 10x10 map: (A) Performance and (B) Moves per Episode.

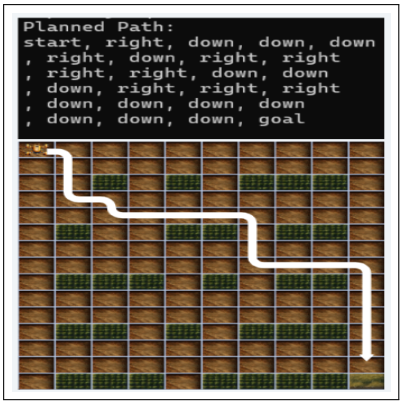
Dueling DQN: Observing Dueling DQN's performance demonstrates that continued optimization and the rectification of shortcomings observed in previous variants yield greater efficiency and reliability in path learning within a reduced time frame. The agent starts learning around the 5000th episode and is able to fully converge within 10000 episodes. According to fig. 11b *(*smoothened by a factor of 0.99)*, Dueling DQN also requires significantly lower amount of random moves to start learning the environment.



(a) Dueling DQN Performance in 10x10 Map



(b) Moves per Episode for Dueling DQN



(c) Path Planned by D3QN for 10X10 Grid in 10x10 Map

Figure 11: Dueling DQN Performance in the 10x10 map: (A) Performance and (B) Moves per Episode.

To further illustrate the impact of this efficiency, fig. 11c visualizes a path planned by the Dueling DQN in a 10x10 grid. The planned path demonstrates a well-optimized route with minimal detours, indicating a high level of learning stability. This structured path is the result of Dueling DQN's capacity to accurately distinguish between states of higher and lower values, reducing the computational load associated with unnecessary exploration.

Table 3
Comparison of DQN, D2QN, and D3QN across two scenarios (8x8 and 10x10)

Performance Metric	8x8			10x10		
	DQN	D2QN	D3QN	DQN	D2QN	D3QN
Convergence Starts	3000	3000	3000	-	7500	5000
Avg. Trailing Reward after 10k Episodes	100	100	100	0	25	100
Maximum Exploration (Unsmoothened)	84	70	64	130	50000	1750
Training Time (min)	5:32	4:21	3:16	-	16:20	8:01

5.4. Results & Analysis

The experimental results tabulated in table 3 indicate that in a relatively simple environment, such as the 8x8 grid, DQN, D2QN, and D3QN exhibit comparable performance in terms of convergence and average trailing reward. However, both D2QN and D3QN demonstrate noticeable improvements in training time, with D3QN achieving the fastest results. In contrast, within a more complex environment like the 10x10 grid with dense obstacle placement, the optimizations present in D2QN and D3QN result in significant advantages over the baseline DQN. In this scenario, D3QN successfully learns the environment within 10,000 steps, attaining an average trailing reward of 100, indicating complete mastery of the task. D2QN performs moderately, with a final trailing reward of 25, reflecting that the agent reached its goal in only 25 out of 100 episodes, corresponding to a 25% success rate. In comparison, DQN fails to learn the environment within the same timeframe.

The maximum number of moves taken per episode provides insights into the exploration behavior of each algorithm. D3QN consistently requires fewer moves than both DQN and D2QN, suggesting that the separation of advantage and value networks in D3QN facilitates more efficient exploration, reducing unnecessary actions while still achieving optimal outcomes.

6. Experimentation - Continuous Action Spaces

While the discrete action space model offers a simplified solution to the path planning problem, it is limited in its ability to fully capture the complexities of real-world agricultural environments. In practice, Unmanned Ground Vehicles (UGVs) operating in the field must have greater degrees of freedom to adapt to dynamic and unstructured terrains. Agricultural environments, with their irregular landscapes, obstacles like trees and crops, and the need for precise navigation, demand more flexible control over movement—something discrete actions cannot adequately provide.

6.1. Environment

In this experiment we have leveraged matplotlib to generate 3 scenarios (fig. 12) in which the agent will be trained.

The environment has the following entities:

1. Black spaces which indicate obstacles.
2. Green filled circle which indicates the starting point of the agent.
3. Red filled circle which indicates the goal of the agent.
4. Red translucent circle which indicates the goal area upon reaching which terminates the current episode.

Action Space: The agent can freely move in the environment with a continuous action space of [-1,1].
Observation Space: The agent can be present at any co-ordinates within the region and thus has an observation space of [0,20].

6.2. Reward Shaping

As demonstrated by Grzes (2017), potential-based reward shaping provides a framework for enhancing agent learning efficiency by shaping the reward structure without changing the underlying optimal policy.

In this implementation, the reward structure(fig. 13b) for the continuous environment is as follows:

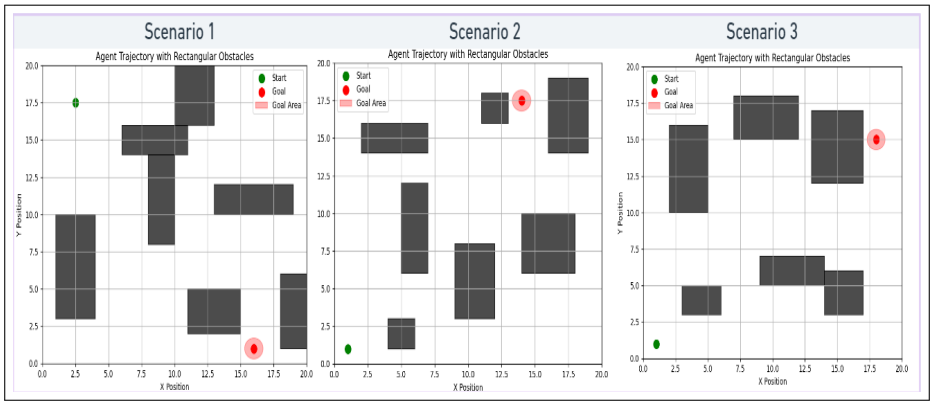


Figure 12: 2D Continuous Environment

- The agent incurs a penalty of -2 for colliding with or approaching obstacles.
- A penalty is applied for navigating too close to the boundary.
- Significant penalties are given for revisiting the same states to promote exploration.
- A small penalty is incurred for each step taken, encouraging quicker convergence.
- The agent receives a dynamic reward based on the Potential-Based Reward Shaping Function, defined as:

$$R_t = -\frac{\text{current distance from the goal}}{\text{distance from the start to the goal}},$$

thereby incentivizing the agent to minimize its distance to the goal with each action.

This approach is crucial for real-world applications, where unmanned ground vehicles (UGVs) must continuously adjust their actions to navigate dynamic environments effectively. By employing actor-critic algorithms like DDPG and TD3, specifically designed for continuous action spaces, we can enhance the UGV's performance in precision agriculture scenarios.

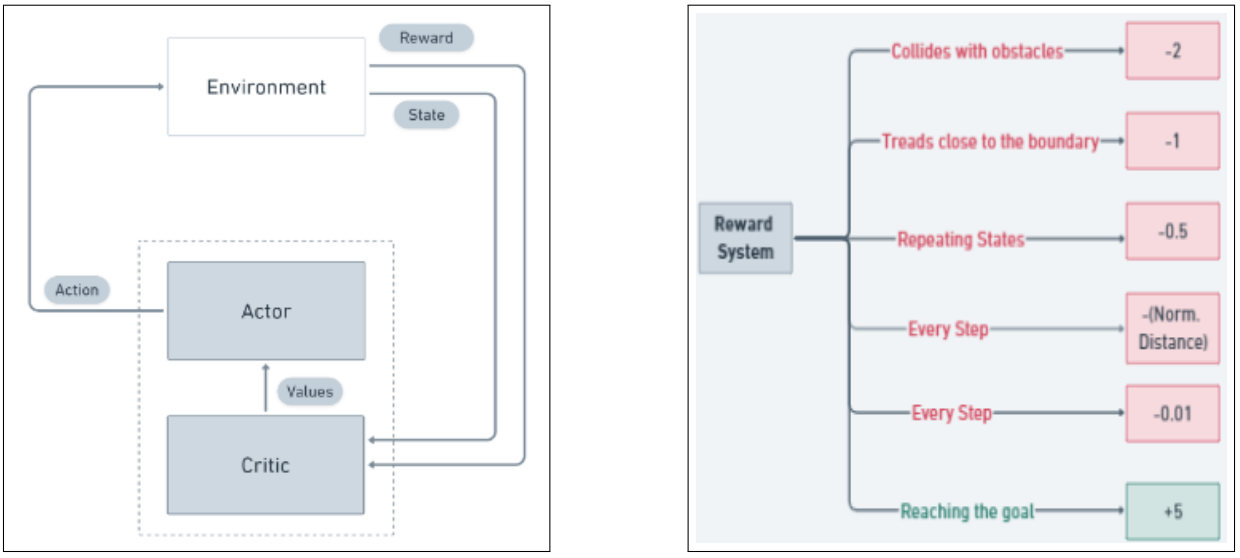
6.3. Actor-Critic Methods

Actor-Critic methods (Grondman et al., 2012) are reinforcement learning techniques that combine methodologies of policy gradient techniques and value based methods to optimize the agent's action in the environment. The Actor-Critic architecture as presented in fig. 13a is a foundational framework in reinforcement learning that combines two components: an actor and a critic. **Actor-Critic algorithms** combine two key components in reinforcement learning: the **actor** and the **critic**.

- **Actor**: Responsible for selecting actions. It learns a policy $\pi(s)$, which maps states s to actions a, and aims to maximize the expected cumulative reward.
- **Critic**: Evaluates the actions taken by the actor by estimating the value function $V(s)$ or the action-value function $Q(s, a)$. This feedback helps the actor improve its policy by learning which actions yield better outcomes.

Key elements:

1. **Policy-based (Actor)**: The actor updates the policy directly using gradients (policy gradient methods), determining which action to take in a given state.
2. **Value-based (Critic)**: The critic evaluates the actor's actions by estimating the expected reward, aiding in refining the policy over time.
3. **Advantage Estimation**: The critic computes the *advantage function* $A(s, a) = Q(s, a) - V(s)$, which helps the actor decide how much better or worse an action is compared to the expected value of the state.



(a) Actor-Critic Architecture

(b) Reward Structure

Figure 13: Visualization of Reward Structure and Actor-Critic Architecture

This synergy allows Actor-Critic algorithms to benefit from both policy optimization (actor) and value estimation (critic), leading to efficient learning in environments with continuous action spaces. The actor component is responsible for exploring and selecting actions based on the current policy, while the critic evaluates these actions by estimating the value function. This dual approach helps mitigate the variance often associated with policy gradient methods, allowing for more stable and reliable updates. Consequently, Actor-Critic algorithms can adaptively improve their policies based on the feedback from the critic, making them particularly well-suited for complex tasks that require nuanced decision-making and planning.

6.3.1. Deep Deterministic Policy Gradient

Building on the Actor-Critic framework, Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) is an off-policy reinforcement learning algorithm tailored for environments with continuous action spaces. Unlike stochastic policy gradient methods, which involve randomness in action selection, DDPG employs a deterministic policy to map states directly to specific actions. This deterministic approach not only improves sample efficiency but also makes it more effective for tasks that require precise control and fine-tuning, such as robotics and continuous control tasks. **Key Components of DDPG**:

1. **Actor Network**: The actor generates actions based on the current state s, using a deterministic policy $\mu(s|\theta^{\mu})$, where θ^{μ} are the parameters of the actor network.
2. **Critic Network**: The critic evaluates the action selected by the actor by estimating the action-value function $Q(s, a|\theta^{Q})$, where θ^{Q} are the parameters of the critic network.
3. **Target Networks and Soft Update**: DDPG employs two target networks (one for the actor and one for the critic), denoted as μ' and Q'. These target networks are updated gradually using the soft update rule to ensure stable learning and avoid sudden, destabilizing changes to the target values.
4. **Ornstein-Uhlenbeck (OU) Action Noise**: DDPG utilizes *OU Action Noise* (Hollenstein et al., 2022) to encourage exploration in continuous action spaces. The noise is added to the actions selected by the deterministic policy $\mu(s)$, allowing the agent to explore different parts of the environment by taking slightly varied actions. This is crucial for preventing the agent from getting stuck in local optima.
5. **Experience Replay**: DDPG uses a replay buffer to store transitions (s_t, a_t, r_t, s_{t+1}), which are sampled uniformly to break correlations between consecutive experiences and stabilize training.
6. **Deterministic Policy**: Unlike stochastic algorithms, DDPG uses a deterministic policy that allows more efficient exploration in continuous action spaces.

Algorithm 2 DDPG Algorithm

Initialize actor network $\mu(s|\theta^{\mu})$ and critic network $Q(s, a|\theta^{Q})$ with parameters θ^{μ} and θ^{Q}
Set target networks to initial parameters: $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay memory R
for each episode **do**
    Set a random exploration process $\mathcal{N}$ for action perturbation
    Observe and initialize the state s_0
    for each timestep t in the episode **do**
        Choose action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$
        Perform action a_t, receive reward r_t, and observe new state s_{t+1}
        Store the experience (s_t, a_t, r_t, s_{t+1}) in buffer R
        Sample a mini-batch of N experiences (s_i, a_i, r_i, s_{i+1}) from R
        Compute target $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update the critic by minimizing the loss:

$$L = \frac{1}{N} \sum_{i} \left(y_i - Q(s_i, a_i|\theta^{Q})\right)^2 \tag{2}$$

        Update the actor using the policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_{i} \nabla_a Q(s, a|\theta^{Q}) \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu}) \tag{3}$$

        Adjust target networks by soft updates:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1 - \tau)\theta^{Q'} \tag{4}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1 - \tau)\theta^{\mu'} \tag{5}$$

    end for
end for

6.3.2. Twin Delayed Deep Deterministic Policy Gradient

This algorithm (Fujimoto et al., 2018) is an enhancement of the DDPG algorithm, specifically designed to address the instability and overestimation issues commonly found in DDPG. By incorporating techniques such as target network updates and noise clipping, TD3 introduces several key improvements that significantly increase the stability and performance of the learning process in continuous action spaces.

Key Improvements Over DDPG:

1. **Clipped Double Q-Learning**: In DDPG, the critic network tends to overestimate Q-values, which can lead to suboptimal policies. TD3 addresses this by using two critic networks, denoted as Q'_1 and Q'_2, and taking the minimum of the two Q-values to calculate the target:

$$y_i = r_i + \gamma \min_{j=1,2} Q'_j(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}))$$

   This reduction in overestimation bias improves the stability of the learning process.
2. **Delayed Policy Updates**: TD3 updates the actor network (policy) less frequently than the critic networks. For every d updates to the critic, the actor is updated once. This delay allows the critic to better estimate the Q-values before updating the policy, reducing the likelihood of harmful updates:

$$\theta^{\mu} \leftarrow \theta^{\mu} + \alpha \nabla_{\theta^{\mu}} J(\theta^{\mu})$$

   where α is the learning rate of the actor network.

Table 4
Comparison of Hyperparameters - DDPG vs. TD3

Hyperparameter	Description	DDPG Value	TD3 Value
Alpha	Learning rate for the actor network determines the size of each update step	0.0001	0.0005
Beta	Learning rate for the critic network determines the size of each update step	0.001	0.005
Tau	Controls the update rate of the target network weights	0.001	0.001
Replay Buffer Size	Maximum capacity of stored experiences	100,000	100,000
Batch Size	Number of samples drawn per training update	64	64
Noise Clip	Clips the noise to this range to stabilize exploration	–	0.5
Policy Update Frequency	Number of steps to delay policy updates	–	2

3. **Target Policy Smoothing**: To prevent the policy from exploiting narrow peaks in Q-values, TD3 adds noise to the target policy during critic updates. This noise is clipped to ensure it remains within a reasonable range, which results in smoother policy updates:

$$a' = \mu'(s_{i+1}|\theta^{\mu'}) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$$

where ϵ is the added noise, and c is a constant used for clipping. This encourages more stable exploration and smoother learning.

These enhancements make TD3 more robust in handling continuous action spaces and provide significantly better performance compared to DDPG, particularly in environments that require fine control, such as autonomous navigation and robotics. The improvements in TD3, including the addition of noise clipping and a more frequent update mechanism for the target networks, address the challenges of overestimation bias and instability often encountered in reinforcement learning. Hyperparameters that are modified in TD3 are detailed in table 4. Notably, TD3 allows for a greater learning rate for both the Actor and Critic Networks, which contributes to more efficient learning and faster convergence in complex tasks.

6.4. Observations

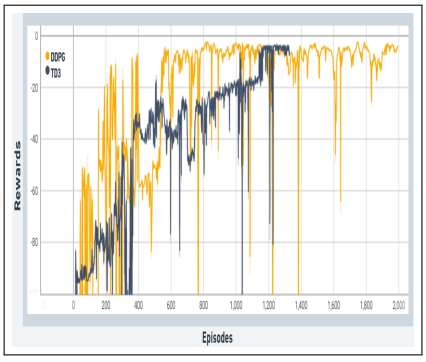
The successful implementation of DDPG and TD3 allows for training both algorithms within the 2D Continuous environment, facilitating a comparative analysis of their performance.

Convergence Criteria: The convergence is defined as the average reward collected per episode for the last trailing 100 episodes being less than -5, and the trailing average steps taken to reach the goal area being less than 30 steps.

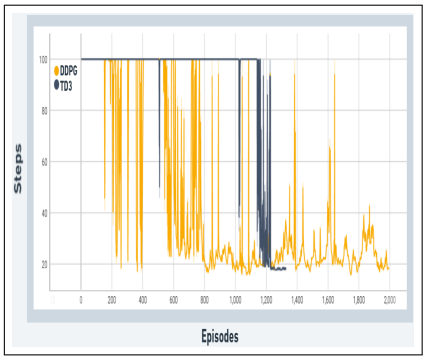
**In this experiment, stability is being quantified as the ability of the agent to be able to minimize the number of steps each episode after it has started converging.*

In Scenario 1, as illustrated in fig. 14a, DDPG began to converge early, around the 800th episode, primarily through its random exploration strategies. However, stability was not achieved until approximately the 2000th episode, indicating that while DDPG could identify rewarding actions, it struggled to maintain consistent performance during this period (see fig. 14b). In contrast, TD3 started converging later, around the 1100th episode, but exhibited a faster and more stable convergence process within the following 200 episodes. This suggests that TD3's enhancements effectively mitigated the instability issues seen in DDPG, allowing for more reliable learning. The paths planned by both TD3 and DDPG during this scenario can be observed in fig. 14c, highlighting their differing strategies for navigation and goal achievement.

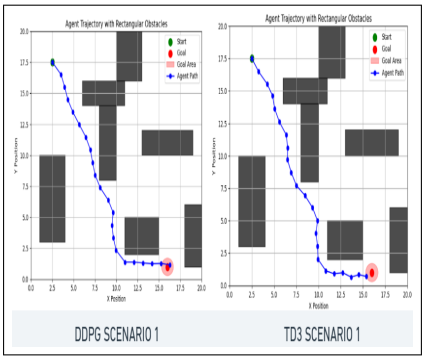
In Scenario 2, as seen in fig. 14d, TD3 again exhibited earlier convergence compared to DDPG. Although the difference was not as pronounced as in Scenario 1, it is noteworthy that TD3 maintained greater stability post-convergence, indicating its robustness in adapting to the task. This improved stability allowed TD3 to consistently achieve high rewards over multiple episodes, suggesting a more reliable learning process. Furthermore, the planned path generated by TD3 was observed to be more efficient and shorter than that produced by DDPG (see fig. 14f),



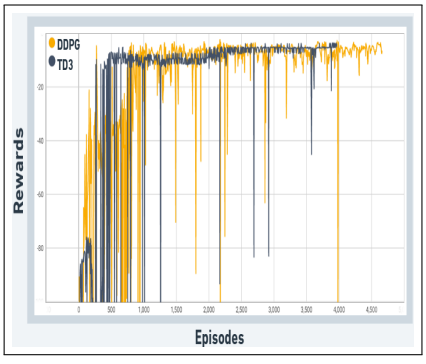
(a) Rewards collected vs episode for Scenario 1



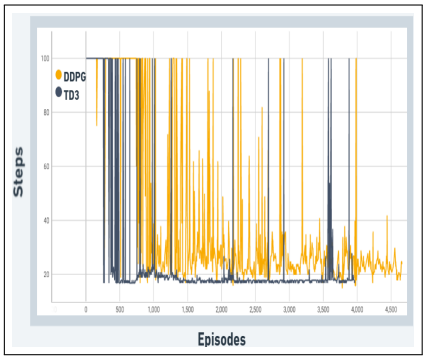
(b) Steps taken vs episode for Scenario 1



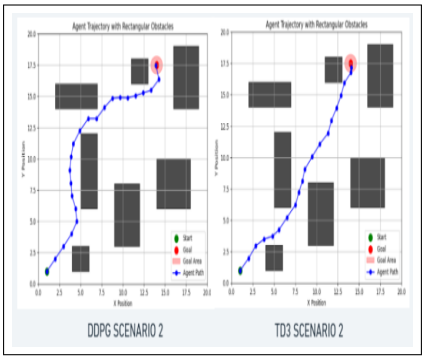
(c) Planned path for DDPG & TD3 for Scenario 1



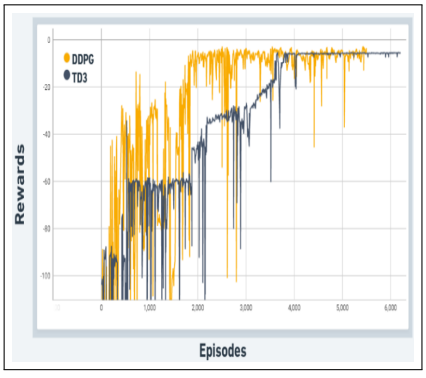
(d) Rewards collected vs episode for Scenario 2



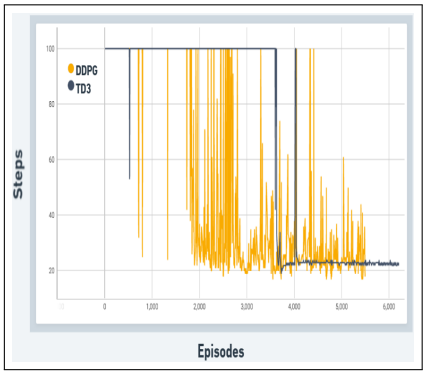
(e) Steps taken vs episodes for Scenario 2



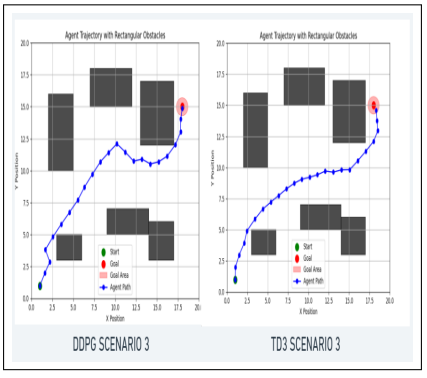
(f) Planned Path for Scenario 2



(g) Rewards collected vs episode for Scenario 3



(h) Steps taken vs episode for Scenario 3



(i) Planned Path for Scenario 3

Figure 14: Training performance of TD3 & DDPG across three scenarios

reflecting TD3's ability to optimize its navigation strategy effectively. TD3 also outperformed DDPG in terms of stability, as observed in fig. 14e, showcasing its advantages in maintaining performance even in challenging conditions.

In Scenario 3, as shown in fig. 14g, an inverse trend was observed: DDPG converged earlier, around the 6000th episode, while TD3 approached convergence after approximately 4000 episodes. However, TD3's average trailing rewards did not exceed -5, indicating incomplete convergence. Despite this, fig. 14h reveals that TD3 maintained excellent stability after initiating convergence, whereas DDPG exhibited significant inconsistency. Furthermore, the path planned by TD3 was more efficient and straightforward compared to that of DDPG (fig. 14i).

Table 5
Comparison of TD3 and DDPG across three scenarios

Performance Metric	Scenario 1		Scenario 2		Scenario 3	
	TD3	DDPG	TD3	DDPG	TD3	DDPG
Convergence starts	1200	800	780	340	3660	1880
Completely converged	1327	1994	3962	4659	-	5500
Path length	20.02	24.23	21.60	24.91	22.41	25.05
Stability Measure	95.05%	85.08%	96.65%	75.03%	96.24%	80.11%
Training time (mins)	18	25	15	20	30	22

6.5. Results and Analysis

The results presented in table 5 are obtained by averaging each metric across five training trials for robustness. The **stability measure** is calculated by setting a threshold for the number of steps and finding the fraction of episodes after convergence starts in which the agent requires fewer steps than the threshold to solve the environment. The formula for the stability measure is:

$$\text{Stability Measure} = \frac{\text{No. of Eps with Steps below threshold}}{\text{Total Eps After Convergence Starts}}$$

Key insights from the results include:

1. In **Scenario 1** and **Scenario 2**, TD3 starts converging later than DDPG but converges faster thereafter, indicating a slower warm-up but quicker learning rate for TD3.
2. In **Scenario 3**, TD3 plans a viable path but does not achieve full convergence, whereas DDPG does, albeit with more steps.
3. TD3 shows significant improvement in the **stability measure**, maintaining a consistent number of steps to solve the environment, with a **19.90% average stability improvement** over DDPG.
4. Regarding **training time**, TD3 generally completes training faster across most scenarios, except Scenario 3, where it takes longer due to environmental complexities. The overall **training time improvement is 5.96%**.
5. TD3 also demonstrates improvements in path length compared to DDPG, with a **13.72% average path length improvement**, which becomes crucial in larger environments, such as agricultural fields.

**Training time has been rounded off.*

In summary, TD3's stability, faster convergence, and efficient path planning (in Scenarios 1 and 2) make it well-suited for 3D simulation environments. Its consistent performance and reduced training time are advantageous for complex, dynamic settings like agricultural fields, where stable navigation and adaptability are essential.

7. Experimentation - Gazebo Env with ROS

7.1. 3D Agricultural Environment Design

The 3D agricultural environment used in this research was carefully designed to simulate a real-world farming field, incorporating common challenges faced by Unmanned Ground Vehicles (UGVs) in precision agriculture. The environment was built within the Gazebo simulator, which provides a physics-based platform for testing and validating autonomous navigation strategies. In this environment, the UGV must navigate autonomously, avoid obstacles, and reach dynamically placed goals, mimicking the tasks a UGV would perform in an actual agricultural setting. The dimensions of the robot and the agricultural environment are mentioned in table 6.

- **Environment Layout and Terrain**: The 3D environment mimics an agricultural field with structured and unstructured elements, featuring: terrain and boundaries (e.g., fences) enclose the environment, ensuring the UGV operates within a designated area, akin to real farm limits.

Table 6
Dimensions of the Robot and Environment

Specification	Dimensions
Robot Dimensions	14 cm x 14 cm x 14 cm
Environment Dimensions	10 m x 8 m

- **Obstacles and Hazards**: Static obstacles, including crops and stones, are modeled in Blender to create a realistic and cluttered layout that challenges the UGV's path-planning capabilities. Additionally, a dynamic obstacle, represented by a dummy human, is introduced to increase environmental complexity. This dynamic obstacle serves as a proxy for real-life agricultural challenges, such as moving livestock and humans, further simulating practical scenarios in precision agriculture.

(a) ROS-Gazebo Pipeline (b) Environment Layout-Static (c) Environment Layout-Dynamic

Figure 15: 3D Environment Structure & Pipeline

7.2. DRL Model Integration with ROS

7.2.1. System Architecture

The training architecture comprises three main components, illustrated in the diagram fig. 15a:

- **Gazebo/Physical Robot** The Gazebo simulator offers a realistic 3D environment for the UGV, simulating sensors like LiDAR (/scan) and odometry (/odom). In real-world applications, this is replaced by a physical robot with identical sensors. The UGV receives movement commands for linear and angular velocities through the /cmd_vel topic in ROS 2.
- **DRL Environment Node** This node acts as a bridge between the environment and the DRL agent, managing the flow of sensor data (e.g., obstacle distances) to the agent. It also relays actions (velocity and steering commands) from the DRL agent to the UGV and computes reward signals based on metrics (e.g., obstacle proximity, goal distance).
- **DRL Agent Node** Responsible for decision-making, this node contains the policy network trained with the TD3 algorithm. It determines actions (linear and angular velocities) based on observations from the environment. PyTorch facilitates training, where the agent maximizes cumulative rewards by refining its navigation policy. The Train() function updates model parameters based on environmental feedback.

7.2.2. System Workflow

The system workflow is divided into the following three sections:

1. **Training Pipeline for Navigation System**: The DRL-based navigation system's training pipeline integrates various components for the Unmanned Ground Vehicle (UGV) to learn efficient path-planning in a 3D environment. Utilizing the Gazebo simulator or a physical robot, it communicates with the Deep Reinforcement Learning (DRL) agent via ROS 2, allowing the UGV to discover optimal policies through trial and error, reinforced by reward feedback.

Table 7
Reward Structure in 3D Environment

Component	Reward/Penalty Formula	Description
Angular Deviation Penalty	$r_{yaw} = -1 \times \lvert goal\_angle \rvert$	Penalizes deviation of the agent's orientation (yaw angle) from the goal direction.
Angular Velocity Penalty	$r_{vangular} = -1 \times (action\_angular^2)$	Penalizes excessive angular motion based on squared angular velocity.
Distance-Based Reward	$r_{distance} = \frac{2 \times goal\_dist\_initial}{goal\_dist\_initial + goal\_dist} - 1$	Rewards the agent for minimizing the distance to the goal.
Obstacle Proximity Penalty	$r_{obstacle} = \begin{cases} -20 & \text{if } min\_obstacle\_dist < 0.22 \\ 0 & \text{otherwise} \end{cases}$	Penalizes the agent for moving too close to obstacles (within 0.22 units).
Linear Velocity Penalty	$r_{vlinear} = -1 \times \left((0.22 - action\_linear) \times 10\right)^2$	Penalizes deviation of the agent's linear velocity from a safe range.
Constant Step Penalty	-1	Penalizes every step to encourage faster goal-reaching behavior.
Success Reward	$+2500$	Reward for successfully reaching the goal.
Failure Penalty	-2000	Penalty for colliding with obstacles or the environment's boundary.

- **Observation and State Representation**: At each time step, the DRL environment node gathers sensor data (LiDAR, odometry) to provide the UGV's current state (position, orientation, obstacle proximity) to the DRL agent node, forming the observation space for decision-making.
- **Action Selection**: Based on observations, the DRL agent selects an action (linear and angular velocities) using the trained policy, which is sent to the DRL environment node to control the UGV in Gazebo or the physical robot via the /cmd_vel ROS 2 topic.
- **Reward Calculation**: In the 3D environment, the reward function is more complex than the 2D, taking into account both the agent's linear and angular movements. The reward structure is defined in table 7.
- **Policy Update (Training)**: Using the reward and new observation, the DRL agent updates its policy via reinforcement learning (e.g., TD3), aiming to maximize cumulative rewards by improving decision-making.
- **Iteration and Learning**: This cycle repeats across multiple episodes, with the UGV exploring the environment and enhancing its navigation strategy. Over time, the DRL agent minimizes collisions, optimizes path efficiency, and accelerates goal achievement.
- **Dynamic Goal Adaptation**: To prevent overfitting and encourage exploration, the position of the goal is randomly changed at the beginning of each episode. This approach ensures the agent does not learn only a single path but instead generalizes across varying scenarios.
- **Moving Obstacles**: The model trained in the static environment was retrained in the dynamic environment using transfer learning, introducing a moving obstacle (dummy human) to increase complexity and introduce external disturbance. The telemetry of the human is designed such that it will only collide with the agent and not the crops.

This training pipeline enables the DRL agent to effectively navigate complex agricultural fields, avoid obstacles, and reach dynamic goals, continually enhancing performance through experience and feedback.

7.3. Observations

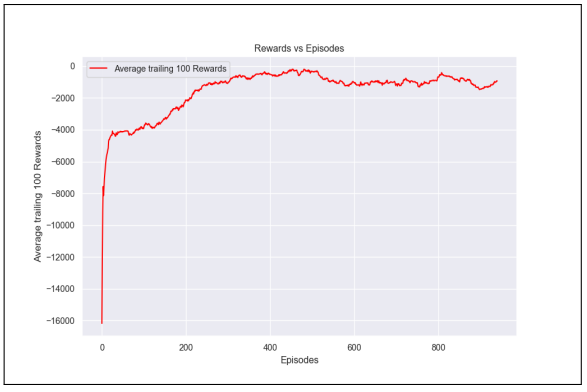
The TD3 algorithm was run over multiple episodes, allowing the UGV to learn navigation using the defined reward structure. Performance was tracked through average rewards and steps per episode.

- **Cumulative Rewards**: The cumulative rewards collected by the UGV during static training are presented in fig. 16a. Initial fluctuations reflect the UGV's exploration phase as it learns to navigate obstacles and

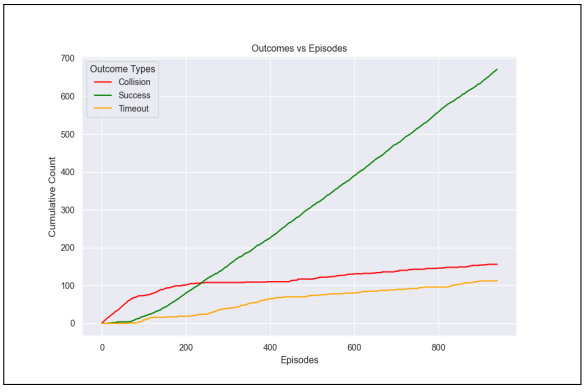
reach dynamically placed goals. Over time, the steady increase in average rewards indicates that the UGV is developing an effective navigation policy. In contrast to the smooth reward trajectory in the static environment, the environment with dynamic obstacles exhibits a more irregular trend (fig. 17a). However, by the 2500th episode, the agent is able to accumulate rewards of approximately 1000.

- **Successful Outcomes**: A steady increase in successful outcomes is observed in both static and dynamic environments (fig. 16b & fig. 17b). While the static environment minimizes collisions by the 700th episode, Timeouts increase due to the agent's strategy of waiting for the next goal to spawn when a goal is placed too close to the crops, ensuring no harm to the crops. Training is stopped at the 900th episode to facilitate the transfer of learning to the dynamic environment.

- **Dynamic Environment Training**: The dynamic environment demonstrates notable training efficiency, benefiting from the pre-trained static agent. Collisions with both dynamic and static obstacles stabilize by the 1700th episode, while successful outcomes increase rapidly. Training is continued until the 5000th episode to further enhance performance.

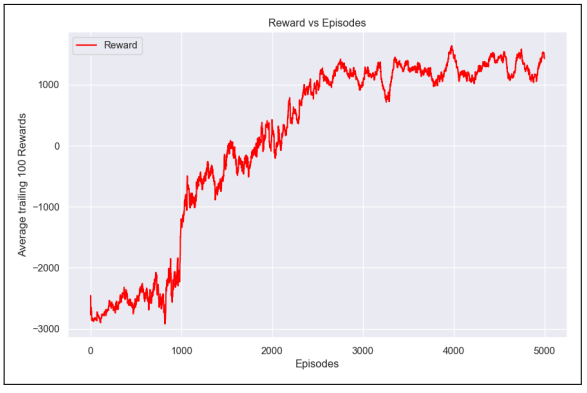


(a) Reward Graph

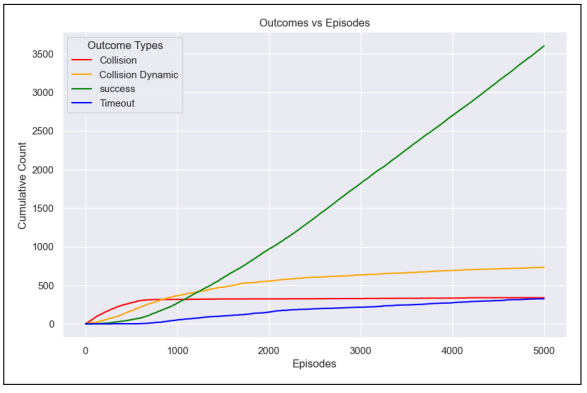


(b) Episode Outcomes

Figure 16: Training performance of TD3 in 3D Static Environment



(a) Reward Graph

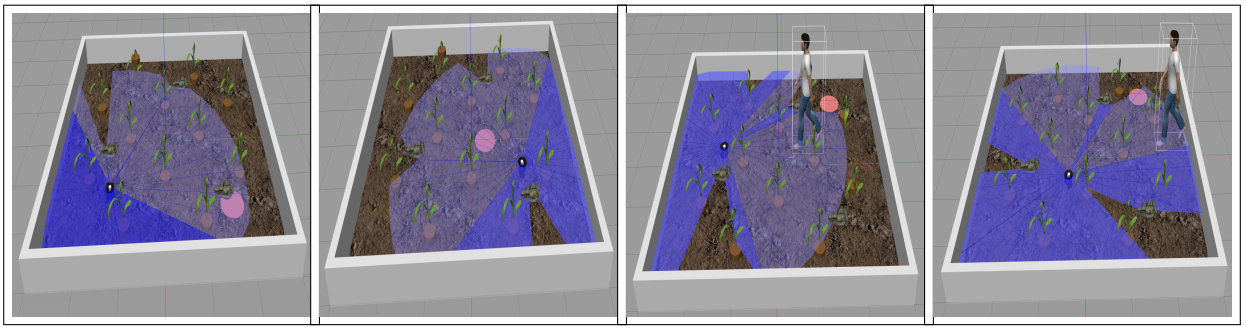


(b) Episode Outcomes

Figure 17: Training performance of TD3 in 3D Dynamic Environment

7.4. Results and Analysis

The performance of the static and dynamic agents is summarized in table 8, which includes key metrics such as trailing 100 successes, trailing 100 rewards, and the time taken for training. Below is a detailed analysis of the results:



(a) Goal 1 (b) Goal 2 (c) Goal 3 (d) Goal 4

Figure 18: Goals in static & dynamic environem

Table 8
Performance Metrics Grouped by Models and Agents

Model	Agent	Trailing 100 Successes	Trailing 100 Rewards	Timestamp
Static	td3_static_100	18%	-3912.403	12:16
	td3_static_300	72%	-1075.812	35:57
	td3_static_500	75%	-101.037	56:34
	td3_static_700	85%	-1031.634	01:13:05
	td3_static_900	76%	-1423.000	01:31:02
Dynamic	td3_dynamic_500	19%	-2595.982	01:35:17
	td3_dynamic_1000	60%	-1198.983	03:10:00
	td3_dynamic_2000	80%	204.507	05:45:54
	td3_dynamic_3000	86%	994.128	08:51:33
	td3_dynamic_5000	95%	1466.829	14:03:08

- **Accuracy of Static Agent** The success rate of static agents improves initially but decreases in later episodes. For example, **td3_static_700** achieves 85% success, which drops to 76% in **td3_static_900**. This decline is due to increased timeouts when goals are placed near crops, causing the agent to wait for the next goal to avoid crop damage.
- **Reward Increase in Dynamic Agents** Dynamic agents show a significant reward increase. For example, **td3_dynamic_500** has a reward of **-2595.982**, while **td3_dynamic_5000** achieves **1466.829**, a **156.5% increase**. This improvement reflects the agent's ability to navigate dynamic obstacles effectively.
- **Pretrained Static Agent on Dynamic Environment** When transferred to dynamic environments, pretrained static agents perform well. For instance, **td3_dynamic_5000** achieves a remarkable 95% success rate. Notably, the unsuccessful outcomes are timeouts rather than collisions, ensuring no harm to the crops or the robot. Transfer learning allows the agent to adapt quickly to moving obstacles, significantly improving performance in fewer episodes.
- **Training Time** Training time varies considerably between static and dynamic agents. The static agent requires upto 90 minutes to train upto 900 episodes while the dynamic agent takes upto 14 hours to train upto 5000 episodes!

8. Discussion

This research successfully achieved the following objectives:

1. **Demonstrated the Efficiency of Deep Reinforcement Learning:** The study proved that deep reinforcement learning algorithms are more efficient than traditional methods by showcasing their adaptability and learning capabilities.
2. **Enhanced Performance of Vanilla DQN:** The research improved the performance of the vanilla Deep Q-Network (DQN) by incorporating Double Q-Learning and Dueling Network architectures. This enhancement led to a 150% faster convergence in the Double DQN (D3QN) compared to Double DQN (D2QN) and resulted in a 98% reduction in training time within a complex 10x10 grid environment.
3. **Comparison of TD3 and DDPG:** A comparative analysis of Twin Delayed DDPG (TD3) and Deep Deterministic Policy Gradient (DDPG) revealed an average stability improvement of 19.9% in TD3. Additionally, TD3 outperformed DDPG in terms of mean path length and training time, making it the preferred choice for 3D simulations.
4. **Development of a 3D Agricultural Environment:** The research team designed a 3D agricultural environment from scratch and implemented a TurtleBot3 agent using TD3. The agent demonstrated seamless obstacle avoidance in both static and dynamic scenarios and yielded a 95% success rate in presence of moving obstacles.

This study successfully extends prior work by incorporating dynamic obstacles and 3D simulations, elements suggested by Zhao et al. (2023) but not addressed in their research . Our results demonstrate the robustness of DRL models in navigating complex, dynamic agricultural environments, highlighting the adaptability of these models.

Lipeng et al. (2024) and Gao et al. (2020) explored path planning in static environments using discrete action spaces . Gao's incremental training from 2D to 3D for indoor robots and Lipeng's DDQN with Prioritized Experience Replay (PER) are valuable for static environments but lack real-time adaptability. Our approach, focusing on continuous action spaces in dynamic agricultural fields, surpasses their methods by enabling smoother paths and more efficient decision-making in constantly changing settings.

Raajan et al. (2020) and Jiang et al. (2020) similarly employed discrete actions for collision avoidance in static environments. By introducing continuous action spaces, this study improves on their methods, yielding more adaptive solutions for real-time 3D scenarios, resulting in smoother navigation paths and enhanced decision-making in dynamic agricultural landscapes.

While Liu et al. (2024a) proposed TD3-DWA to optimize robotic navigation, focusing on velocity control in static settings, our work expands on this by implementing TD3 in dynamic environments . This shift allows for more adaptive, smoother paths, especially in agricultural scenarios, where obstacles are continuously changing.

Looking ahead, the next step involves implementing this solution on a real-life robot and testing it in actual agricultural fields. Additionally, this study opens the door for further enhancements, such as integrating the Dynamic Window Approach (DWA) with TD3 or experimenting with stochastic policy methods like SAC to improve decision-making under uncertainty.

9. Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

10. Conclusion

The experiments conducted in this study demonstrate that learning-based algorithms, particularly Deep Q-Networks (DQN), significantly outperform deterministic primitive algorithms for Unmanned Ground Vehicle (UGV) path planning in precision agriculture. While DQN provided substantial benefits in adaptability and performance, the research identified several areas for further optimization. This led to the exploration of advanced versions such as Double DQN and Dueling Double DQN, which offer improvements in training stability and convergence speed.

However, recognizing the necessity for continuous action evaluation in complex agricultural environments, the study transitioned to a continuous 2D simulation. In this context, the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm was found to outperform the Deep Deterministic Policy Gradient (DDPG), demonstrating enhanced performance in navigating the dynamic challenges of agricultural settings. Consequently, TD3 was selected as the optimal algorithm to enhance the agent's planning capabilities within a 3D environment utilizing Gazebo and ROS.

To effectively guide the agent's decision-making, a custom reward structure was implemented, taking into account various environmental factors. This adaptive reward system proved crucial for improving the agent's path planning

efficiency. The results underline the potential of continuous deep reinforcement learning techniques in advancing UGV path planning for precision agriculture.

Future work will focus on transitioning from simulation to a live robot platform, enabling real-world testing in agricultural settings. This next phase will further validate the effectiveness of continuous deep reinforcement learning approaches in practical applications, emphasizing their role in enhancing the efficiency and reliability of UGV operations in precision agriculture.

CRediT authorship contribution statement

Laukik Patade: Conceptualization, Programming, Methodology, Writing – original draft. **Rohan Rane:** Simulation, Programming, Validation, Designing, Software. **Sandeep Pillai:** Programming, Validation, Writing - original draft.. **Sujata Kulkarni:** Mentoring, Project Planning, Grant curation, Supervision.. **Kiran Talele:** Project Planning, Grant curation, Supervision. .

References

Adzhar, N., Yusof, Y., Ahmad, M.A., 2020. A review on autonomous mobile robot path planning algorithms. Advances in Science, Technology and Engineering Systems Journal 5, 236–240. doi:10.25046/aj050330.

Chen, P., Pei, J., Lu, W., Li, M., 2022. A deep reinforcement learning based method for real-time path planning and dynamic obstacle avoidance. Neurocomputing 497, 64–75. doi:https://doi.org/10.1016/j.neucom.2022.05.006.

Chu, Z., Wang, F., Lei, T., Luo, C., 2023. Path planning based on deep reinforcement learning for autonomous underwater vehicles under ocean current disturbance. IEEE Transactions on Intelligent Vehicles 8, 108–120. doi:10.1109/TIV.2022.3153352.

Deshpande, S.V., R, H., Ibrahim, B.S.K.K., Ponnuru, M.D.S., 2024. Mobile robot path planning using deep deterministic policy gradient with differential gaming (ddpg-dg) exploration. Cognitive Robotics 4, 156–173. URL: https://www.sciencedirect.com/science/article/pii/S2667241324000119, doi:https://doi.org/10.1016/j.cogr.2024.08.002.

Farama Foundation, . FrozenLake Environment. https://gymnasium.farama.org/environments/toy_text/frozen_lake/. Accessed: 2024-10-25.

Fujimoto, S., van Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. URL: https://arxiv.org/abs/1802.09477, arXiv:1802.09477.

Gadekar, A., Fulsundar, S., Deshmukh, P., Aher, J., Kataria, K., Patel, D.V., Barve, D.S., 2023. Rakshak: A modular unmanned ground vehicle for surveillance and logistics operations. Cognitive Robotics 3, 23–33. URL: https://www.sciencedirect.com/science/article/pii/S2667241323000083, doi:https://doi.org/10.1016/j.cogr.2023.02.001.

Gao, J., Ye, W., Guo, J., Li, Z., 2020. Deep reinforcement learning for indoor mobile robot path planning. Sensors 20. URL: https://www.mdpi.com/1424-8220/20/19/5493, doi:10.3390/s20195493.

Grondman, I., Busoniu, L., Lopes, G.A.D., Babuska, R., 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42, 1291–1307. doi:10.1109/TSMCC.2012.2218595.

Grzes, M., 2017. Reward shaping in episodic reinforcement learning, in: Sixteenth International Conference on Autonomous Agents and Multiagent Sytems (AAMAS 2017), ACM. pp. 565–573.

Guo, S., Zhang, X., Zheng, Y., Du, Y., 2020. An autonomous path planning model for unmanned ships based on deep reinforcement learning. Sensors 20. doi:10.3390/s20020426.

Hasselt, H.v., Guez, A., Silver, D., 2016. Deep reinforcement learning with double q-learning, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press. p. 2094–2100.

Hollenstein, J., Auddy, S., Saveriano, M., Renaudo, E., Piater, J., 2022. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. Transactions on Machine Learning Research URL: https://openreview.net/forum?id=NljBlZ6hmG. survey Certification.

Jiang, L., Huang, H., Ding, Z., 2020. Path planning for intelligent robots based on deep q-learning with experience replay and heuristic knowledge. IEEE/CAA Journal of Automatica Sinica 7, 1179–1189. doi:10.1109/JAS.2019.1911732.

Josef, S., Degani, A., 2020. Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain. IEEE Robotics and Automation Letters 5, 6748–6755. doi:10.1109/LRA.2020.3011912.

Li, Y., 2021. An rrt-based path planning strategy in a dynamic environment, in: 2021 7th International Conference on Automation, Robotics and Applications (ICARA), IEEE. pp. 1–5. doi:10.1109/ICARA51699.2021.9376472.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N.M.O., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. CoRR abs/1509.02971. URL: https://api.semanticscholar.org/CorpusID:16326763.

Lipeng, L., Xu, L., Liu, J., Zhao, H., Jiang, T., Zheng, T., 2024. Prioritized experience replay-based ddqn for unmanned vehicle path planning. URL: https://arxiv.org/abs/2406.17286, arXiv:2406.17286.

Liu, H., Shen, Y., Zhou, C., Zou, Y., Gao, Z., Wang, Q., 2024a. Td3 based collision free motion planning for robot navigation. URL: https://arxiv.org/abs/2405.15460, arXiv:2405.15460.

Liu, L., Chen, J., Zhang, Y., Chen, J., Liang, J., He, D., 2024b. Unmanned ground vehicle path planning based on improved drl algorithm. Electronics 13. doi:10.3390/electronics13132479.

Mammarella, M., Comba, L., Biglia, A., Dabbene, F., Gay, P., 2022. Cooperation of unmanned systems for agricultural applications: A theoretical framework. Biosystems Engineering 223, 61–80. doi:https://doi.org/10.1016/j.biosystemseng.2021.11.008.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015a. Human-level control through deep reinforcement learning. Nature 518, 529–533. doi:https://doi.org/10.1038/nature14236.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015b. Human-level control through deep reinforcement learning. Nature 518, 529–533. URL: https://api.semanticscholar.org/CorpusID:205242740.

OrtataŞ, F.N., UlutaŞ, H., Şahİn, M.E., Çİftcİ, F., 2023. Autonomous mapping and spraying in precision agriculture using unmanned ground vehicles, in: 2023 Innovations in Intelligent Systems and Applications Conference (ASYU), IEEE. pp. 1–5. doi:10.1109/ASYU58738.2023.10296767.

Raajan, J., Srihari, P.V., Satya, J.P., Bhikkaji, B., Pasumarthy, R., 2020. Real time path planning of robot using deep reinforcement learning. IFAC-PapersOnLine 53, 15602–15607. URL: https://www.sciencedirect.com/science/article/pii/S2405896320332171, doi:https://doi.org/10.1016/j.ifacol.2020.12.2494.

Talaviya, T., Shah, D., Patel, N., Yagnik, H., Shah, M., 2020. Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides. Artificial Intelligence in Agriculture 4, 58–73. doi:https://doi.org/10.1016/j.aiia.2020.04.002.

Tewari, V., Pareek, C., Lal, G., Dhruw, L., Singh, N., 2020. Image processing based real-time variable-rate chemical spraying system for disease control in paddy crop. Artificial Intelligence in Agriculture 4, 21–30. URL: https://www.sciencedirect.com/science/article/pii/S258972172030009X, doi:https://doi.org/10.1016/j.aiia.2020.01.002.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N., 2016. Dueling network architectures for deep reinforcement learning, in: Proceedings of The 33rd International Conference on Machine Learning, PMLR. pp. 1995–2003.

Yan, C., Xiang, X., Wang, C., 2020. Towards real-timepath planning through deep reinforcement learning for a uav in dynamic environments. Journal of Intelligent & Robotic Systems 98, 297–309. doi:10.1007/s10846-019-01073-3.

Yan, L., Wang, P., Yang, J., Hu, Y., Han, Y., Yao, J., 2021. Refined path planning for emergency rescue vehicles on congested urban arterial roads via reinforcement learning approach. Journal of Advanced Transportation 2021, 8772688. doi:https://doi.org/10.1155/2021/8772688.

Yang, Y., Li, J., Peng, L., 2020. Multi-robot path planning based on a deep reinforcement learning dqn algorithm. CAAI Transactions on Intelligence Technology 5, 177–183. doi:10.1049/trit.2020.0024.

Zhao, T., Wang, M., Zhao, Q., Zheng, X., Gao, H., 2023. A path-planning method based on improved soft actor-critic algorithm for mobile robots. Biomimetics 8. doi:10.3390/biomimetics8060481.

Zhi, C., Xiumin, S., 2019. Research on path planning of mobile robot based on a* algorithm. International Journal of Engineering Research and 8, 11. doi:10.17577/IJERTV8IS110186.