MDPI

Article

A Hybrid Planning–Learning Framework for Autonomous Navigation with Dynamic Obstacles

Hatice Arslan Öztürk [1], Sırma Yavuz [2], and Çetin Kaya Koç [3]

[1] Computer Engineering Department, Yıldız Technical University, Istanbul, Türkiye; haticearslan8154@gmail.com
[2] Computer Engineering Department, Atlas University, Istanbul, Türkiye; sirma.yavuz@atlas.edu.tr
[3] Nanjing University of Aeronautics and Astronautics, Nanjing, China; University of California Santa Barbara, USA; cetinkoc@ucsb.edu
* Correspondence: haticearslan8154@gmail.com

Abstract

Traditional navigation methods work well in known, static environments but degrade in real-world settings with dynamic and unpredictable obstacles. This paper presents **Double Deep Q-Network with A* guidance (DDQNA)**, a hybrid navigation algorithm that enables an agent to traverse mazes containing static and dynamic obstacles while maintaining a low probability of collision. DDQNA combines **A*** guidance with **Double Deep Q-Network (DDQN)** learning using an ϵ-greedy policy, and introduces a redesigned reward function and an improved action-selection mechanism to better exploit A*'s directional cues during training. We evaluate DDQNA in a custom **Pygame** simulation across **11** environments of increasing difficulty. Experimental results show that DDQNA consistently outperforms standard DDQN and other state-of-the-art reinforcement learning baselines, achieving higher goal-reaching rates, fewer visited cells, shorter computation times, and higher cumulative rewards. These results indicate that DDQNA provides both effective navigation and computational efficiency in complex environments with static and dynamic obstacles.

Keywords: autonomous navigation; deep reinforcement learning; dynamic obstacle avoidance; double deep q-networks.

1. Introduction

Autonomous vehicles (AVs) are at the forefront of technological innovation, significantly impacting our daily lives and reshaping various industries. Their remarkable advancements not only usher in profound changes in transportation but also transform business processes across sectors such as logistics, agriculture, emergency response, and national security. Equipped with sophisticated sensors and navigation systems, AVs enhance operational safety and efficiency. However, the successful and secure movement of these vehicles relies heavily on the effective functioning of these systems, including sensors, actuators, and the underlying decision-making algorithms.

Challenges such as weak or absent GPS signals and difficulties in location determination can introduce substantial risks to AV operations. To navigate these complexities, autonomous vehicles must be capable of making real-time, self-guided decisions based on dynamic environmental data.

Machine learning, particularly reinforcement learning (RL), is a powerful tool that equips AVs with the ability to learn from their surroundings and adapt swiftly to changing conditions. The integration of deep learning with RL has further advanced these capabilities,

Received:
Revised:
Accepted:
Published:

enabling exceptional performance in critical tasks like object detection [1] and semantic segmentation [2]. These advancements are crucial for AVs to accurately interpret and interact with their environment.

Our research contributes to this evolving field by developing an RL-based framework for an autonomous agent designed to navigate through a maze filled with both dynamic and static obstacles. In contrast to traditional navigation in static environments, dynamic environments include obstacles whose positions may change during navigation. Such dynamic obstacles create a non-stationary environment in which previously safe paths may become invalid, requiring the navigation system to continuously adapt its decisions. This work highlights the potential of these technologies in real-world applications, particularly focusing on the challenge of optimizing paths to targets while effectively maneuvering around obstacles in dynamic environments.

The main contributions of this paper can be summarized as follows:

- **A novel probabilistic integration of A* guidance into the DDQN action selection** via an epsilon-greedy policy enables the agent to dynamically choose between optimal deterministic actions and learned actions during runtime decision-making.
- **An experimentally optimized reward function** specifically designed to improve navigation performance in complex, dynamic, and randomly generated environments.
- **Extensive validation across eleven randomly generated dynamic environments** demonstrates statistically significant improvements over standalone DDQN, PPO, and A2C algorithms.
- **A modular simulation framework** that allows for reproducible experiments and benchmarking of hybrid reinforcement learning and classical planning algorithms.

This paper is organized as follows. A review of the relevant literature is introduced in Section 2. The definition of the problem and the proposed approach are explained in detail in Section 3. The experimental results are presented in Section 4, and finally, in Section 5, a discussion of the findings and their implications is provided.

2. Review of Relevant Literature

2.1. The First Approaches and DQN

As autonomous vehicle (AV) technology continues to evolve, emphasis is placed not only on developing advanced navigation and decision-making capabilities but also on optimizing these systems for specific applications, including delivery [3], indoor navigation for search and rescue operations [4,5], and warehouse management [6]. The integration of Deep Q-Networks (DQN) and other artificial intelligence techniques is central to this evolution, enabling AVs to learn from past experiences, adapt to new challenges, and carry out a diverse array of tasks with increasing proficiency.

Despite its broad range of applications, DQN presents opportunities for enhancement in real-world scenarios characterized by high dynamics and uncertainty. A primary challenge lies in the algorithm's tendency to overestimate the values of actions, which can lead to suboptimal policy decisions. This overestimation bias may result in significant inefficiencies, particularly in complex environments where precise decision-making is critical. Moreover, deep Q-learning faces difficulties in managing real-world scenarios that involve large state spaces and continuous action spaces, especially in the context of autonomous driving [7].

2.2. Google DeepMind's DDQN and its Applications

To address these challenges, Google DeepMind proposed the Double Deep Q Network (DDQN) algorithm to solve the overestimation problem caused by the DQN algorithm [8]. The DDQN selects the action of the agent that maximizes the Q-value, which is an

estimate of the future reward of an action executed in a given state. This approach has been employed in numerous studies, particularly in the field of autonomous vehicle movement, demonstrating its versatility and effectiveness in various applications. For instance, Munoz et al. [3] applied deep reinforcement learning for autonomous drone delivery, using a DDQN to determine the direction of movement from depth images. They created a realistic environment in AirSim for safer drone flight, recording neural network weights as checkpoints to improve training efficiency. Their work demonstrated the potential of using deep Q-learning to enhance decision making in complex environments. Similarly, Dorling et al. [9] validated a multirotor drone energy consumption model, which showed that consumption varies linearly with payload and battery weight. Their findings led to an improvement of more than 10% in drone delivery efficiency by optimizing battery weight. In another study, Mnih et al. [10,11] developed a deep Q network and a deep learning model for reinforcement learning (RL) in the Atari 2600 environment, achieving superior performance compared to other RL approaches and human gamers. Their work, along with Bellemare et al. [12], who discussed the Arcade Learning Environment (ALE) as a platform to develop general AI technology, highlighted the challenges and potential of using deep reinforcement learning in various applications. Kersandt et al. [13] examined drones capable of detecting obstacles and following visual traffic rules, achieving results similar to human performance in test flights. They suggested that improving the checkpoint strategy could further enhance the outcomes. Chowdhury et al. [4] developed an Unmanned Aerial Vehicle (UAV) model to perform indoor search and rescue missions without GPS, using Received Signal Strength (RSS) in a Q-learning algorithm for better convergence speed and trajectory accuracy. In [14], the authors proposed a dual-phase solution to DRL-related problems, merging offline and online learning using transfer learning and fine-tuning. This strategy involves training a neural network in a wide range of indoor environments and then applying this experience through transfer learning when training a smaller segment of the network in a similar, yet unfamiliar, testing environment. The network is divided into sections that can be trained and those that cannot, with updates only made to the weights of the trainable portion. This division is a compromise between achieving performance goals (such as avoiding obstacles) and limiting the number of training computations. The trainable section comprises the last few fully connected layers. Researchers used a collection of eight distinct environments for the training phase and subsequently validated the results using three unique environments for the testing process.

Reinforcement algorithms are based on the online learning paradigm. The agent gains experience by interacting with the environment and then updates the current policy using this experience. When discussing the autonomous movement of a vehicle, online interaction is a challenging task in most cases due to security reasons. In this case, offline training can be considered. Many commonly used reinforcement learning methods can learn from off-policy data; however, such methods often cannot learn effectively from entirely offline data without any additional on-policy interaction. The high-dimensional and expressive function approximation generally leaves algorithms vulnerable to distribution changes between training and test data sets [15]. To overcome this problem, policies are trained with reinforcement learning in simulation and then transferred to the real world [16–18]

2.3. Obstacle Detection and Avoidance

Collectively addressing these challenges is essential; however, our current focus is on the problem of obstacle detection and avoidance. These two aspects are critical for ensuring the safe movement of autonomous vehicles (AVs). AVs must accurately recognize and navigate around obstacles in their path to prevent collisions and guarantee safe operation.

Achieving this requires sophisticated sensor systems and complex algorithms capable of real-time processing.

Machine learning, particularly deep learning, has increasingly been employed to enhance obstacle detection algorithms, enabling agents to learn from past experiences and make improved predictions for future encounters. Obstacle detection primarily involves the application of sensors and image processing techniques aimed at accurately identifying and locating obstacles in the AV's trajectory. This process heavily relies on a variety of sensors, including cameras, LiDAR, and radar, which gather data about the AV's surroundings. The raw data collected from these sensors is then processed using advanced image processing algorithms to differentiate between obstacles and free space. At this stage, machine learning techniques, such as convolutional neural networks (CNNs), are often utilized to enhance the accuracy and reliability of obstacle detection [19,20].

On the other hand, obstacle avoidance entails making appropriate algorithmic decisions once obstacles have been detected. After an obstacle is identified and its position relative to the autonomous vehicle (AV) is established, the AV must determine how to adjust its path to circumvent the obstacle. This process typically involves complex path planning and control algorithms that take into account the current speed, direction, and maneuverability of the AV, as well as the size and location of the detected obstacle.

Reinforcement learning and other artificial intelligence techniques are frequently employed to optimize these decision-making processes, enabling AVs to learn from past experiences and refine their avoidance strategies over time.

A review of the relevant literature indicates that reinforcement learning (RL) techniques are commonly used for obstacle avoidance problems. Choi et al. [21] proposed a framework for collision avoidance in reinforcement learning, wherein mobile robot agents learned to efficiently navigate around dynamically moving obstacles while reaching target points. Additionally, they integrated the A* path planning algorithm to enhance the algorithm's pathfinding capabilities. An enhanced Q-learning algorithm that incorporates priority weights was introduced in [22] to address the challenges of dynamic obstacle avoidance path planning. However, in this paper, the word dynamic does not refer to the existence of a dynamically moving obstacle but to the random generation of the starting point, obstacles, and destination point in the agent travel process. These studies [23,24] demonstrate that autonomous mobile robots can be used effectively to avoid obstacles using Q-learning. The first study considers dynamic obstacle navigation, while the second demonstrates the effectiveness of Q learning in path planning and dynamic obstacle navigation through extensive hyperparameter tuning and iterative simulations. In addition to Q-learning, many reinforcement learning approaches have been commonly employed in obstacle avoidance and path planning challenges. In another paper [25], a reinforcement learning-based approach is proposed for safe and efficient drone navigation in static environments using a minimal set of low-cost sensors. The effectiveness of two autonomous navigation algorithms, PPO and A2C, is investigated and compared to the pathfinding performance of the A* algorithm. Tabakis and Dasygenis [26] employed the Double Deep Q-Network (DDQN) and Deep Deterministic Policy Gradient (DDPG) algorithms for path planning in both static and dynamic environments. Their findings indicate that DDQN is more effective in static environments, whereas DDPG excels in dynamic environments due to its adaptability. In [27], a novel reward mechanism is proposed utilizing Deep Q Learning (DRL) to enhance collision avoidance, enabling the robot to stop when faster-moving objects approach and resume once they pass. The Advantage Actor-Critic (A2C) algorithm is also one of the policy-based reinforcement learning methods utilized for dynamic obstacle avoidance problems in dynamic and complex environments. Zhou et al. [28] proposed an A2C model enhanced with an attention mechanism and prioritized experience replay for

mobile robots operating in crowded and dynamic indoor environments. This structure not only accelerated the learning process but also significantly improved obstacle avoidance performance by achieving high success rates. Similarly, Zhai [29] introduced a cooperative obstacle avoidance approach based on A2C in a connected vehicles environment. Their model enables vehicles to collectively share environmental information and avoid obstacles, resulting in a substantial reduction in collision rates compared to traditional individual vehicle approaches. In addition, hybrid strategies combining A2C with other optimization methods are frequently employed to enhance performance. For instance, Xing [30] developed a hybrid path planner by integrating the Seeker Optimization Algorithm (SOA) with A2C to accomplish both global and local planning tasks. This structure exhibited superior performance compared to conventional ROS-based planners, offering shorter paths and faster response times, particularly in dynamic environments. Sinha [31], demonstrated that by integrating A2C with quantum computing under the "Nav-Q" framework, autonomous vehicles could achieve more stable and safer navigation in dynamic urban settings. It was reported that Nav-Q converged faster and achieved higher cumulative rewards compared to classical A2C. Finally, Almazrouei [32] conducted a comprehensive survey evaluating the performance of A2C and other reinforcement learning methods for dynamic obstacle avoidance in autonomous ground vehicles. Their study highlighted A2C's advantages, particularly in continuous control problems, while also discussing challenges such as training difficulties and sim-to-real transfer issues. Another policy-based RL approach used in dynamic obstacle avoidance problems is the PPO algorithm. The PPO algorithm is preferred because it provides more stable and secure policy updates. In a literature review by Wu et al. [33], the increasing use of PPO in autonomous driving behavior planning and its performance advantages were examined in detail. To further improve performance, RL algorithms are often combined with additional techniques. Bilban and İnan [34] proposed a new structure called LFPPO by improving the classical PPO algorithm with a Lévy-flight based exploration strategy. This method enabled autonomous ground vehicles to perform safer and more stable navigation in real traffic conditions. In the experiments conducted in the CARLA simulation environment, LFPPO achieved much fewer collisions (1% vs. 19%) compared to the standard PPO with a 99% success rate. Nie et al. [35], with the method they developed for automatic guided vehicles (AGV), increased both the stability and overall performance of the learning process by combining sample editing and adaptive learning rate strategies with PPO. This hybrid structure enabled safe and collision-free route generation, especially in dynamic environments. Tang et al. [36] developed a PPO-based scene-specific training method for a land robot that avoids moving obstacles in unknown environments while also tracking targets. The PPO algorithm successfully performed both safe and target-oriented maneuvers by evaluating instantaneous information from the environment. The study showed how effectively PPO can be used in dynamic environments without prior map knowledge.

2.4. The Role of Traditional Methods

In addition to reinforcement learning methods, efforts to adapt traditional path-planning algorithms to dynamic and uncertain environments have increased with the need for real-time decision-making in autonomous systems. In this context, Maw et al. [37] proposed an improved anytime navigation algorithm called iADA*, which takes into account uncertainties in the real world and environmental changes. Compared to the classical ADA* algorithm, iADA* provides solutions that are 257% faster in partially known and unknown dynamic environments. With its real-time replanning capability, it enables rapid adaptation to dynamic conditions by generating collision-free and time-efficient paths. Similarly, the CBS-D* algorithm proposed by Jin et al. [38] extends the D* Lite

framework through a conflict-based strategy, allowing autonomous mobile robots to avoid collisions in unknown and continuously changing environments. By incorporating novel components such as third-order neighborhood scanning and wait and circuity strategies, CBS-D* achieves a 31% higher success rate compared to the traditional D* Lite algorithm. Yu and Wang [39] introduced a time-dependent multiobstacle avoidance algorithm (TD-MOA) designed in accordance with international regulations to prevent collisions at sea (COLREG). By integrating the D* Lite algorithm into a spatio-temporal risk model, this method produces safe and rule-compliant paths in environments with both dynamic and static obstacles. The simulation results confirmed the effectiveness of the method and its performance in real time. Although CBS-D* and iADA* introduce significant improvements over classical path planning methods such as A*, Anytime A*, and D* Lite by integrating conflict-based reasoning and anytime incremental search to enhance adaptability in dynamic environments, they remain fundamentally rule-based algorithms. Their reliance on predefined heuristics and deterministic strategies provides advantages such as reduced computational overhead and rapid decision-making in resource-constrained or time-sensitive scenarios. However, these approaches exhibit limited generalization capabilities and lack the ability to learn from experience. In contrast, reinforcement learning (RL) methods can autonomously develop adaptive navigation policies by interacting with the environment, enabling agents to process high-dimensional sensor data, predict obstacle behaviors, and refine decision-making through trial and error. Consequently, although traditional path planning algorithms designed to adapt to dynamic and uncertain environments are effective in replanning and quickly responding to changes, their scalability and robustness may be limited in highly unpredictable or complex scenarios, where learning-based approaches often provide superior performance. iADA-RL, an extended version of the iADA algorithm proposed by Maw et al. [40], integrates deep reinforcement learning to improve adaptability, combining a global graph-based planner with a local planner based on RL. This approach offers real-time path generation and replanning capabilities in dynamic environments, particularly for unmanned aerial vehicles (UAVs).

In addition to avoiding obstacles, it is essential to expand the current approach by incorporating complementary strategies to ensure the most effective and direct path to the target. For this purpose, the A* algorithm is one of the most commonly used path-planning algorithms in the literature [41–44]. The A* algorithm efficiently finds the shortest path between two points using a heuristic-based search. By integrating A* with reinforcement learning, the hybrid model combines the strengths of both approaches: A* provides precise path planning, ensuring the quick identification of the shortest viable route, while reinforcement learning offers the flexibility to adapt to changing conditions in real time [25]. Also, a hybrid approach enhances time performance [43–45], as the algorithm can plan paths more quickly by adapting the weight coefficients of the heuristic function, a crucial factor in real-time navigation. The combination ensures optimal path planning, as it uses state-action and reward functions to guide the algorithm toward the most efficient paths. Additionally, the hybrid approach is capable of handling dynamic environments effectively, making it well-suited for real-world applications where conditions can change rapidly. These developments illustrate how autonomous navigation research has evolved from static path planning toward adaptive navigation strategies designed for dynamic environments. In particular, real-time replanning algorithms, reactive obstacle avoidance methods, and learning-based approaches have become key technologies for handling moving obstacles and environmental uncertainty.

2.5. Our Final Analysis

Upon examination of the studies conducted, we believe that highlighting a few points would be beneficial to facilitate significant development and improvement. Approaches based solely on reinforcement learning can face limitations in real-time path planning and navigation efficiency, especially in dynamic environments where quick decision-making is crucial. Determining the optimal path in the presence of dynamic obstacles can be computationally costly. Additionally, studies that use reinforcement learning and compare its performance with path-planning algorithms generally conduct their analyzes in static environments. There are relatively few studies where the environment is dynamic or includes a dynamically moving obstacle. Furthermore, the number of environments used for training is limited. Creating different environments in 3D simulation platforms comes with a high time cost. In addition, there is no universally accepted performance metric for evaluating the performance of algorithms, which makes comparing them challenging. In these studies, the performance of the algorithms was evaluated only based on their alignment with the specified objectives. Even the most advanced simulation environments fail to model the real world. Encoding the chaos and uncertainty contained in the world is quite challenging, which limits the application of theory to practice.

Table 1 presents a comparative analysis that clarifies the differences between classical path planning algorithms and reinforcement learning (RL) methods. The table contrasts their fundamental properties concerning environmental knowledge, learning capability, dynamic adaptation, stability, and responsiveness to path changes.

Table 1. Comparison of Path Planning and RL Algorithms in Terms of Environment Knowledge, Learning Capability, and Adaptability

Algorithm	Method Type	Env. Knowledge	Learning	Dynamic Adaptation	Stability	Path Change Response
A*	Deterministic Planning	✓	✗	✗	Deterministic	Restarts from beginning
Anytime A*	Deterministic Planning	✓	✗	✗	Improves over time	Restarts from beginning
D* Lite	Replanning	✗	✗	✓	Deterministic	Replans from current position
PPO	Reinforcement Learning (RL)	✗	✓	✓	High	Policy-based replanning
A2C	Reinforcement Learning (RL)	✗	✓	✓	Moderate (may fluctuate)	Policy-based replanning
DDQN	Reinforcement Learning (RL)	✗	✓	✗	Low (inconsistent)	Learns slowly
DDQNA	Hybrid (Planning + RL)	✗	✓	✓	High	Adaptive via planning and learning

Classical algorithms such as A*, Anytime A*, and D* Lite are rule-based and deterministic. While they provide reliable and predictable behavior, they generally require complete or partial prior knowledge of the environment and offer limited flexibility in highly dynamic settings, as they lack intrinsic learning capabilities.

In contrast, RL-based methods such as PPO and A2C learn policies from interaction, enabling greater adaptability to changing conditions; however, though their stability is sensitive to training quality and hyperparameter choices. DDQN also possesses learning capabilities, but when used in isolation, it may struggle to maintain consistent performance and adaptability in more complex tasks.

The proposed hybrid method, DDQNA, combines the strengths of classical planning with learning-based adaptation. By leveraging both deterministic planning and experience-driven policy refinement, DDQNA achieves competitive performance in complex, unpredictable environments without requiring prior knowledge of the environment.

2.6. Our Proposal

By proposing a hybrid algorithm that combines the Double Deep Q Network and the A* algorithm for autonomous agent navigation, our work aims to address the mentioned shortcomings. In addition, the proposed approach not only leverages the robustness of DDQN in decision making but also integrates the efficiency of A* in finding the shortest

path, offering a comprehensive solution to autonomous navigation in complex environments with static and dynamic obstacles. Although A* and DDQN are well-established methods when applied independently, the integration proposed in this study, to the best of our knowledge, has not been addressed in prior research. Specifically, it incorporates several novel components that distinguish it from existing approaches:

- A **hybrid integration mechanism** in which A* and DDQN are combined via a probabilistic epsilon-greedy policy during runtime action selection. Unlike previous hybrid approaches, our method dynamically selects between optimal A*-guided actions and learned DDQN actions at each decision step.
- In addition to hybrid action selection, we propose an **experimentally optimized reward function** that penalizes repeated cell visits, proximity to dynamic obstacles, and inefficient trajectories while rewarding shorter paths to the goal.
- To support a more comprehensive evaluation, we tested the proposed method in **eleven randomly generated environments** rather than in a limited set of predefined scenarios. This setup offers several advantages:
  - When the results are analyzed in detail, it can be observed that reinforcement learning algorithms do not consistently succeed in all environments. Relying on only a few environments risks focusing on favorable scenarios and introducing bias. We specifically mitigated this by using a broader evaluation.
  - The combination of **dynamic obstacles** and the requirement to reach the goal via the shortest possible path represents a significant challenge. The presence of a dynamically moving obstacle increases the problem complexity considerably, as evidenced by the lower success rates of baseline RL algorithms in our experiments.
  - We would also like to emphasize that generating the environments *sequentially and randomly* is an unbiased method to test algorithm robustness. The positions of obstacles and goals are not manually selected to facilitate the task but are generated completely randomly by being executed sequentially.

3. Materials and Methods

Problem Definition: As illustrated in Figure 1, our objective is to enable our autonomous vehicle (AV) to reach the red target point from the green starting point via the shortest path without colliding with any static or dynamic obstacles. In this study, a dynamic obstacle refers to an obstacle whose position changes during the navigation process, creating a time-varying environment that may invalidate previously planned paths. The environment is randomly generated. The autonomous vehicle (AV) does not have prior knowledge about the environment, such as obstacle locations or the maze structure. The agent only observes its current location and learns to navigate through interaction with the environment by receiving reward signals based on its actions. Although the environment itself has full knowledge of the map for purposes such as reference path planning or performance evaluation, the agent makes decisions based only on its local observations. We note that the environment refers to the simulation system that controls the agent's movement, rewards, and episode progression. Additionally, the obstacles are of fixed size and are not larger than the AV itself. Furthermore, the agent moves with a constant step size throughout the grid, which corresponds to a fixed velocity in this discrete environment.

Since we aim for the agent (AV) to reach the target via the shortest path, we incorporated the widely used A* algorithm into our methodology. The A* algorithm is typically an effective path planning algorithm for static environments, as it relies on a pre-known, fixed representation of the map or environment for planning the path. However, in a dynamic environment where obstacles may change positions over time or new obstacles may appear,

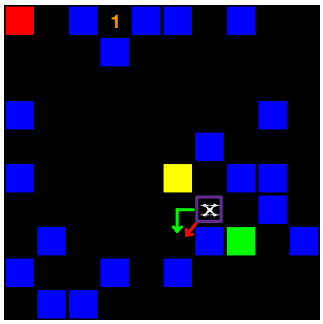

Figure 1. The red arrow indicates a diagonal move the agent cannot perform; the green arrow shows the feasible path. The agent is not larger than obstacles. If an obstacle lies orthogonally adjacent to the agent, it cannot move diagonally.

the effectiveness of the A* algorithm can diminish. This is because the algorithm, once the path is planned, cannot update its path or respond immediately to dynamic changes. For this reason, in our experiments, we did not rely solely on the A* algorithm. To analyze the contribution of the A* algorithm to our objective, we compared the performance of the DDQNA (our hybrid approach) with the standard DDQN algorithm and other commonly used reinforcement learning methods.

3.1. Environment

OpenAI Gym [46], DeepMind Lab [47], Mujoco (Multi-Joint dynamics with contact) [48], CoppeliaSim [49], and AirSim [50] are just a few of the numerous simulation environments available for RL experimentation, each offering a unique set of challenges and opportunities for agent development. These environments offer photorealistic and complex 3D settings, enabling the testing of Reinforcement Learning (RL) algorithms under conditions that closely approximate real-world scenarios. As a result, various challenging scenarios and environments can be explored to enhance the capabilities of RL agents. To alleviate the computational burden associated with 3D simulation environments, we use the Pygame library [51], which is more suitable for simpler 2D simulations. We designed an environment with a starting point and a target point, containing a single dynamic obstacle and capable of being randomly regenerated. This approach allows us to automatically generate as many problem-specific environments as desired without the need for additional design. The key differences between the designed environment and the existing simulations [46–50] are summarized below.

- Random environments are automatically generated.
- The dimensions of the environment can be adjusted as desired.
- Dynamic obstacles can be added.
- The 2D simulation reduces the system requirements and facilitates the testing of various scenarios to challenge and improve the capabilities of RL agents.

The illustrated example demonstrates a 10x10 maze comprising a starting point (green), a destination point (red), and a dynamically moving obstacle (yellow). The agent is terminated after colliding with walls or obstacles. The primary goal is not only to arrive at the destination in the shortest possible time but also to navigate effectively through the environment, minimizing the number of steps taken. Beyond simply reaching the destination, our objectives include minimizing repeat visits to cells and finding the most direct path to the target. To ensure that the starting and destination points in the randomly generated maze are far apart, we use the Euclidean metric to strategically set their locations. The agent's possession of eight distinct actions logically leads to the selection of the Euclidean

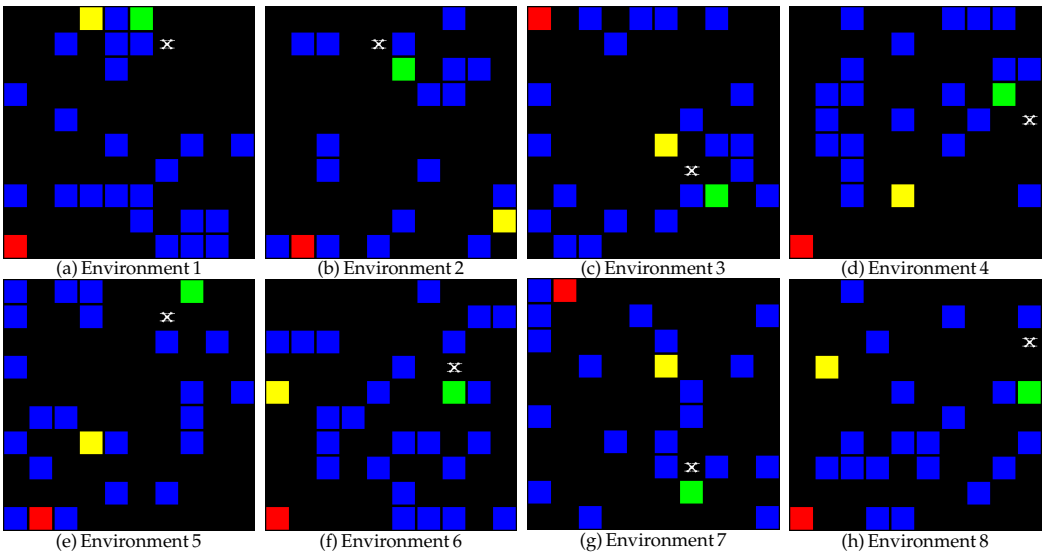

Figure 2. Randomly generated environments ordered left to right. Each panel shows a 10×10 maze with a start (green), goal (red), one dynamically moving obstacle (yellow), and the agent (white cross). The obstacle moves randomly during the episode; the position shown is a snapshot.

metric; otherwise, the Manhattan metric would be more suitable. Examples of randomly generated environments are given in Figure 2.

3.2. Action

The simulation agent is designed to perform eight distinct actions, as shown in Figure 3. The agent's movements encompass rightward, leftward, forward, backward, and diagonal directions, providing a comprehensive range of motion.

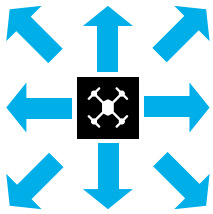

Figure 3. Actions of the agent.

Due to the structure of the environment and the placement of obstacles, the agent's diagonal movement is restricted under certain conditions. Specifically, if there is an obstacle directly at a 90-degree angle to the agent, it cannot move diagonally. As seen in Figure 1, to move to the bottom-left cell, the agent must first move left and then down. Additionally, the agent can never enter cell number 1 because of this restriction. This constraint ensures that the agent navigates around obstacles by first moving horizontally or vertically before attempting a diagonal move.

3.3. Reward

In Reinforcement Learning (RL) algorithms, the reward function represents feedback that an agent receives from its environment as a consequence of its actions. Based on this feedback, the agent strives to optimize its *policy*, which is a set of rules that dictate which action it will select in any given situation. Positive rewards signal to the agent that it has executed a proper action, encouraging it to repeat such actions. Negative rewards, on the other hand, indicate that the agent should avoid certain actions. Currently, our goal is to enable the agent to reach its target with minimal effort, without colliding with obstacles. Given our preference for the agent taking fewer steps towards the target, we

have attributed a cost to each step. For every step that does not result in reaching the target, encountering obstacles, or increasing the distance from the target, the agent receives a negative reward. When the target is reached or the agent gets closer to it, the agent receives a positive reward. As we have observed the positive impact of improving the reward mechanism on performance, we continue to enhance the reward function based on experimental results. The final reward function that we have selected is given below.

$$R(s,a) = \begin{cases} -100 & \text{obstacle collision, episode ends} \\ +100 & \text{target reached, episode ends} \\ -5 & \text{per movement action } a \\ +10 & \text{if } a \text{ decreases target distance} \\ -20 & \text{if } a \text{ decreases obstacle distance} \\ -20 \times n & \text{revisiting a cell the } n\text{th time} \end{cases} \tag{1}$$

This reward mechanism is designed to encourage the agent to reach the target through the shortest path without hitting obstacles. The agent gains rewards as it approaches the target, but it should avoid getting close to obstacles, taking unnecessary steps, and repeatedly visiting the same cells. This balanced approach ensures that the agent progresses effectively while avoiding risky situations. The reward function significantly impacts the learning process and the resultant performance of an RL algorithm. A properly defined reward function helps the agent learn more rapidly and efficiently, enabling it to achieve superior results.

3.4. Epsilon Greedy Strategy

The epsilon-greedy strategy is a popular method used in reinforcement learning to balance exploration and exploitation. In this approach, the agent primarily selects actions that it believes will yield the highest reward (exploitation). However, with a probability of epsilon (ϵ), the agent chooses a random action instead (exploration). This allows the agent to discover new strategies and potentially better actions that it might not have considered otherwise [52].

Mathematically, the action selection can be described as:

$$\text{action} = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \text{optimal action} & \text{with probability } 1-\epsilon \end{cases} \tag{2}$$

Additionally, epsilon is often decreased over time to allow the agent to exploit more as it learns. This can be expressed as:

$$\epsilon = \max(\epsilon_{\text{end}}, \epsilon \times \epsilon_{\text{decay}}) \tag{3}$$

If the agent chooses not to explore (i.e., when a random number is greater than ϵ), it then decides how to exploit. If an optimal action (i.e., a recommended action from A*) is provided, the agent selects that action with a fixed probability of 0.5. Otherwise (i.e., with the remaining 50% chance), the agent chooses the action predicted by the DDQN model. By gradually decreasing the value of epsilon over time, the agent can shift from exploring the environment to exploiting the knowledge it has gained, thereby optimizing its performance. This approach helps the agent balance the need to explore new possibilities with the goal of maximizing rewards based on what it has learned.

The epsilon-greedy part ensures that the agent sometimes explores by picking random actions. When not exploring, there is a 50% chance that the agent will follow the path

recommended by the A* algorithm. The remaining 50% of the time, the agent will use the DDQN model to select the optimal action. Furthermore, the 50% probability of choosing between A and DDQN* after deciding not to explore is independent of the epsilon-greedy strategy. This selection mechanism ensures that the agent balances between model-based pathfinding and the learned policy to improve navigation performance.

3.5. Network Architecture

The network structure consists of four fully connected layers interconnected by Leaky ReLU and dropout activation functions [53]. This architecture was chosen due to the benefits provided by these activation functions and techniques. Leaky ReLU helps mitigate the vanishing gradient problem by allowing a small, non-zero gradient when the unit is not active, facilitating faster and more stable training of deep networks. Dropout, on the other hand, is a regularization technique that prevents overfitting by randomly dropping units during training, thereby improving the generalization of the model. Generally, in DDQN, it is expected that a Q-value output is produced for every action, and thus no activation function is utilized in the output layer. Here, the input layer is set to 2 because the DDQN agent receives a two-dimensional state input from the environment, corresponding to the agent's current position (x,y) on the 2D grid. The output layer is set to 8 to represent the eight possible actions available to the agent, as illustrated in Figure 4. The agent uses *Adam* as the optimization algorithm and *Mean Squared Error (MSE)* as the loss function [53]. Adam is an optimization algorithm that adapts the learning rate, facilitating quick and efficient learning. The MSE measures the discrepancy between the Q-values estimated by the agent and the target Q-values.

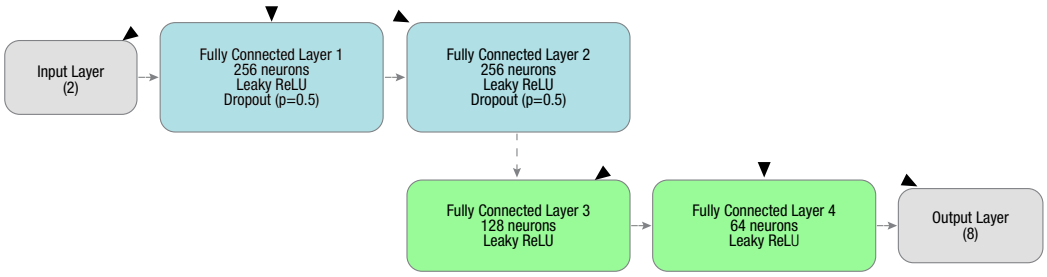

Figure 4. Network architectures of DDQN and DDQNA.

3.6. Double Deep Q-Network (DDQN) Algorithm

Double Deep Q-Networks (Double DQN) is an improved reinforcement learning method that reduces the overestimation bias found in standard Deep Q-Networks (DQN). In DQN, a neural network that predicts the value of performing a particular action in a given state, called the Q network, approximates the Q-value function. However, because this paradigm relies on a single network for both action selection and evaluation, it is prone to overestimating Q-values. Double DQN improves this by introducing a dual-network architecture. It retains the original Q-network for action selection, but it employs a secondary network, termed the target network, for action evaluation. While the original Q-network is still responsible for selecting the best action to perform in the next state, the target network is used to calculate the Q-value of that action. This decoupling reduces the bias of overestimation, leading to more stable learning [8,54].

$$Y_{\text{DoubleDQN}} = R_{t+1} + \gamma Q\left(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t), \theta_t^-\right), \quad (4)$$

where $Y_{\text{DoubleDQN}}$ denotes the updated target Q-value, R_{t+1} is the reward for the next state, γ represents the discount factor, and Q is the Q-value function. The function $Q(S_{t+1}, a; \theta_t)$ determines the action a that maximizes the Q-value in the subsequent state S_{t+1}, with θ_t as the parameters of the current Q-network, and θ_t^- as those of the target network [55]. This strategic separation of action selection and evaluation through the target network mitigates overestimation bias and fosters more reliable learning outcomes.

3.7. A and DDQNA Algorithm*

The A* algorithm is a path-finding and graph traversal algorithm renowned for its efficiency and accuracy in determining the shortest path between two points. It combines features of Dijkstra's algorithm and Best-First Search, making it optimal and complete [56]. The heuristic aspect of A* estimates the cost of reaching the goal from a node, guiding the search in a direction that is likely to lead to the shortest path.

In this study, we propose the **DDQNA algorithm**, which combines A* with Double Deep Q-Network (DDQN) to guide an autonomous agent through a maze environment. The purpose is to enable the A* algorithm to guide the agent during training, with the aim of helping the agent find the shortest path to the target. The effectiveness of the algorithm in this context lies in its ability to calculate the most efficient route, considering various constraints and obstacles within the maze. Unlike the algorithm proposed in [41], our mechanism integrates the A* algorithm with the epsilon greedy strategy for action selection, and it selects the action recommended by the A* algorithm with a probability of 50%, regardless of any learning rate. This approach allows for random action selection based on the epsilon greedy strategy to promote exploration while also utilizing the neural network to identify actions with the highest expected reward for exploitation. This combination enhances the learning efficiency and performance of agents in complex environments by blending the stochastic nature of the epsilon greedy strategy with the precision of A*'s pathfinding capabilities. The only difference between the DDQN algorithm with the improved reward mechanism and the DDQNA algorithm is that the DDQNA algorithm benefits from the guidance of the A* algorithm. The pseudocode for the DDQNA algorithm is given in Algorithm 1.

3.8. Theoretical Formulation of the Action Selection Policy

The proposed DDQNA algorithm utilizes a hybrid decision-making strategy that combines model-based A* path guidance with model-free learning via Double Deep Q-Networks (DDQN). This strategy is embedded within an ϵ-greedy exploration framework.

Let $\pi(s)$ denote the action-selection policy at state s. The policy can be formalized as a probabilistic mixture of two sub-policies:

$$\pi(s) = \begin{cases} \text{random action} & \text{with probability } \epsilon, \\ \pi_{\text{A}^*}(s) & \text{with probability } (1-\epsilon)\cdot p, \\ \pi_{\text{DDQN}}(s) = \arg\max_a Q(s,a;\theta) & \text{with probability } (1-\epsilon)\cdot(1-p), \end{cases} \tag{5}$$

where ϵ is the exploration probability and $p \in [0,1]$ is the fixed probability of selecting the A* action when exploiting.

In our implementation, we set $p = 0.5$ to ensure equal contribution from both sub-policies during the exploitation phase. This symmetric setup encourages both structured search (via A*) and adaptive learning (via DDQN) without biasing the learning trajectory prematurely.

Algorithm 1 DDQNA Algorithm: Hybrid Action Selection with A* and DDQN

Initialize maze environment and DDQNA agent
Initialize neural network parameters θ, target network $\theta^- \leftarrow \theta$
Set hyperparameters: ϵ, γ, update frequency, number of episodes, learning rate
for each environment E in training set **do**
  for each episode $e = 1$ to N **do**
    Initialize state s_0
    for each time step t **do**
      Sample random number $r \in [0, 1]$
      if $r < \epsilon$ **then** ▷ Exploration
        Choose random action $a_t \in \mathcal{A}$
      else ▷ Exploitation via hybrid strategy
        Sample $r' \in [0, 1]$
        if $r' < p$ **then**
          $a_t \leftarrow \pi_{A^*}(s_t)$ ▷ Use A* path guidance
        else
          $a_t \leftarrow \arg\max_a Q(s_t, a; \theta)$ ▷ Use DDQN policy
        end if
      end if
      Execute action a_t, observe reward r_t and next state s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
      Sample mini-batch from $\mathcal{D}$ and update θ using:

$$y_t = r_t + \gamma \cdot Q\left(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-\right)$$

$$\mathcal{L}(\theta) = \left(y_t - Q(s_t, a_t; \theta)\right)^2$$

      if step mod target_update_frequency == 0 **then**
        Update target network: $\theta^- \leftarrow \theta$
      end if
      if episode terminates **then**
        break
      end if
    end for
    Decay exploration rate: $\epsilon \leftarrow \max(\epsilon_{\text{end}}, \epsilon \cdot \epsilon_{\text{decay}})$
  end for
end for

3.9. Ablation Study and Effect of the A Weight Parameter p*

To assess the impact of the A* weighting parameter p, which defines the probability of selecting the A* action during exploitation, we conducted a comparative evaluation across eight test environments with $p = 0.25$, $p = 0.50$, and $p = 0.75$.

The results indicate that while increasing the A* weight to 75% occasionally yields high success in specific environments (e.g., Environment 6), it also produces pronounced instability, as evidenced by 0% success rates in Environments 2, 5, and 7. Consequently, the standard deviation of success with $p = 0.75$ is approximately 42.5%, the highest among all configurations.

By contrast, the 50%-50% hybrid configuration yields the highest mean success rate (85.4%) and the lowest variance (Std 3.6%), demonstrating both robust performance and consistency across diverse maze topologies.

Table 2 summarizes the per-environment success rates, while Table 3 aggregates the means and standard deviations across environments.

A closer examination of Table 2 shows that excessive reliance on the A* guidance (p = 0.75) leads to inconsistent performance across environments. Although the algorithm

Table 2. Per-environment reached goal (%) for different p values.

Environment	$p = 0.25$	$p = 0.50$	$p = 0.75$
1	82.0	84.0	78.0
2	81.0	92.0	0.0
3	0.0	83.0	76.0
4	90.0	86.0	81.0
5	86.0	87.0	0.0
6	82.0	81.0	100.0
7	79.0	88.0	0.0
8	85.0	82.0	87.0

occasionally achieves very high success rates (e.g., Environment 6), it completely fails in several environments where dynamic obstacles significantly alter the optimal path. This behavior indicates that over-dependence on deterministic planning limits the agent's ability to adapt through learning. Conversely, when the A* guidance probability is reduced (p = 0.25), the algorithm relies predominantly on learned policies. While this improves adaptability, the agent sometimes lacks sufficient structured guidance, leading to inefficient exploration and occasional failures. The balanced configuration (p = 0.5) provides the most stable performance by combining the strengths of both mechanisms. As shown in Table 3, it achieves the highest mean success rate and the lowest variance across environments.

Table 3. Mean and standard deviation of success rates (%) over all environments.

Metric	$p = 0.25$	$p = 0.50$	$p = 0.75$
Reached Goal Mean	73.1	**85.3**	52.7
Reached Goal Std	∼29.7	∼**3.6**	∼42.6

Interestingly, the 25% A* configuration shows moderate average success but also significant inconsistency, including a complete failure to reach the goal in Environment 3.

These observations confirm that excessive reliance on A* planning can compromise adaptability and prevent the agent from learning effective policies in environments with dynamic obstacles or unpredictable configurations. Conversely, insufficient guidance reduces efficiency and reliability.

Overall, the ablation results validate the choice of $p = 0.5$ as the most balanced configuration, offering both high success rates and low variance across all tested environments. This balance supports better generalization and robust navigation performance.

3.10. Computational Complexity Analysis of the DDQNA Algorithm

The computational complexity of the proposed DDQNA training procedure can be expressed in terms of the following parameters: the number of episodes e, the maximum number of steps per episode s, the batch size b used for training, the number of neurons per layer in the Q-network f, and the number of cells in the maze $n = w \times h$.

Each training step consists of several components. First, A* path planning is performed to compute an optimal action, which requires $\mathcal{O}(n \log n)$ time in a grid environment. Next, a forward pass through the Q-network evaluates the action values, incurring $\mathcal{O}(f^2)$ computational cost. The current transition is appended to the replay buffer in constant time. If enough samples are available, a minibatch gradient update is applied, requiring $\mathcal{O}(b \cdot f^2)$ operations. Finally, the environment state, including agent position, reward computation, and obstacle dynamics, is updated in constant time.

Summing these components, the per-step time complexity would be:

$$\mathcal{O}\left(n \log n + f^2 + b \cdot f^2\right).$$

Since each episode consists of up to s steps, the per-episode time complexity grows proportionally to s. Therefore, over all training episodes, the total time complexity is:

$$\mathcal{O}\big(e \cdot s \cdot (n \log n + b \cdot f^2)\big).$$

In practice, the neural network computations are highly parallelizable on modern GPUs, and A* search remains efficient for grids of moderate size.

Regarding memory requirements, the space complexity is determined by four components: the weights of the main and target Q-networks, the replay buffer, the visited cell dictionary, and the maze representation itself. Each network requires $\mathcal{O}(f^2)$ space to store weights. The replay buffer stores up to m transitions in $\mathcal{O}(m)$ memory. Tracking cell visit counts and maintaining the maze matrix each require $\mathcal{O}(n)$ storage. Thus, the overall space complexity is:

$$\mathcal{O}\big(f^2 + m + n\big),$$

which is moderate and suitable for conventional hardware configurations. Table 4 summarizes the time and space complexity of the main computational components involved in the training process.

Table 4. Summary of time and space complexity components.

Component	Time Complexity	Space Complexity
A* Path Planning	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
Forward Pass	$\mathcal{O}(f^2)$	$\mathcal{O}(f^2)$
Replay Buffer Update	$\mathcal{O}(1)$	$\mathcal{O}(m)$
Training Step	$\mathcal{O}(b \cdot f^2)$	$\mathcal{O}(m)$
Environment Update	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Total Training	$\mathcal{O}\big(e \cdot s \cdot (n \log n + b \cdot f^2)\big)$	$\mathcal{O}(f^2 + m + n)$

The results indicate that the dominant computational cost arises from the neural network training step, which scales with the batch size and the size of the network. The additional overhead introduced by the A* planner is relatively moderate and scales with O(n log n), which remains efficient for grid-based environments.

This analysis suggests that the hybrid approach does not introduce prohibitive computational costs compared to standard deep reinforcement learning algorithms. Instead, it provides structured guidance with manageable computational overhead, making the approach suitable for real-time navigation scenarios with moderate environment sizes. As a result, the proposed hybrid approach, which combines A* search with deep Q-learning, achieves scalable performance in dynamic navigation tasks without imposing prohibitive computational or memory demands.

3.11. Evaluation Complexity

During the test phase, the algorithm executes the learned policy without performing any parameter updates. At each step of a test episode, the pre-trained Q-network selects an action by computing a forward pass through the network. In this phase, neither replay buffer sampling nor gradient backpropagation is required.

Specifically, the computational cost per step consists of two main components. First, the forward pass incurs a time complexity proportional to $\mathcal{O}(f^2)$, where f denotes the number of neurons per layer in the network. Second, updating the state of the environment, including the movement of the agent, the calculation of rewards, and the motion of the obstacle, requires constant time $\mathcal{O}(1)$. When visualization is enabled through the PyGame interface, additional rendering operations introduce a further cost of $\mathcal{O}(n)$, where n represents the total number of grid cells. In particular, this rendering overhead is independent of the decision-making process.

Accordingly, the per-step time complexity during evaluation is given by:

$$\mathcal{O}(f^2 + n).$$

Since each evaluation episode comprises up to s steps, the total time complexity per episode scales proportionally as follows:

$$\mathcal{O}(s \cdot (f^2 + n)).$$

Regarding memory requirements, evaluation retains only the pre-trained Q-network parameters and the environment state. This results in a space complexity of:

$$\mathcal{O}(f^2 + n).$$

Overall, this analysis demonstrates that the computational demands of evaluation are substantially lower than those of training. Consequently, the approach remains efficient and well-suited for repeated testing and large-scale benchmarking scenarios.

4. Results

In this section, we evaluate the performance and effectiveness of the proposed approach during both the training and testing phases across a range of environments. All experiments were conducted under consistent conditions on a computer equipped with an Intel Core i7 2.4 GHz processor, 32 GB of RAM, and an Nvidia GeForce RTX 4050 GPU.

In this paper, we compare our proposed hybrid method (DDQNA), which integrates classical path planning with a reinforcement learning method, with standard RL baselines, including A2C, PPO, and DDQN. The selected baselines represent different families of deep reinforcement learning methods, including value-based and actor–critic approaches, which are widely adopted in autonomous navigation research. Since the proposed DDQNA framework is designed as a reinforcement learning–based decision-making strategy, evaluating it against other RL algorithms enables a more appropriate assessment of improvements in learning stability, convergence behavior, and navigation success rate. Classical navigation algorithms such as A*, D*, and D* Lite follow deterministic planning paradigms that typically rely on known environment representations and explicit path computation. In contrast, reinforcement learning methods focus on learning navigation policies through interaction with the environment. Because these two paradigms address different aspects of the navigation problem and operate under different assumptions, comparing DDQNA with other RL baselines provides a clearer evaluation of its learning-driven navigation capabilities. This selection ensures that the evaluation covers representative reinforcement learning paradigms and allows the proposed hybrid framework to be assessed against widely used learning-based navigation strategies under consistent experimental conditions. To ensure a fair and controlled comparison, all algorithms were trained under identical experimental conditions, including the same environments, reward structure, training episodes, and evaluation metrics. The hyperparameters were selected based on commonly adopted configurations reported in reinforcement learning literature and were kept consistent across algorithms. In addition to the algorithmic structure, we specifically investigate the effect of the reward function on agent performance. To this end, the baseline DDQN, A2C, and PPO algorithms were implemented in their standard forms using a conventional reward structure that penalizes undesirable actions and rewards goal-directed behavior. A reward of +100 was assigned for encouraged actions, while a penalty of -100 was applied for undesired behaviors. To better evaluate the contribution of the proposed reward mechanism, we also applied the reward function defined in Equation 1 to the standard DDQN algorithm. This modified version is referred to as DDQN* throughout the experiments. Except for the

change in reward formulation, all training and testing conditions, including environment configurations, hyperparameters, and computational settings, were kept identical in all algorithms to ensure a fair and controlled comparison.

4.1. Training Process

The training process primarily consists of a loop that runs over a specified number of training environments, each of which consists of 2000 episodes. For each training environment, a new environment is initialized, and the agent's current environment is set to this new environment. The epsilon value, which controls the trade-off between exploration and exploitation, is initially set to the starting value (1.0). For each episode within the current environment, the environment is reset, and the agent performs actions until the episode ends. These actions are selected based on the current state and the epsilon value. At each step, the agent trains by updating its Q-network based on the state, action taken, reward received, next state, and whether the episode has ended. The epsilon value decays at the end of each episode to slowly shift the balance from exploration towards exploitation as the agent learns more about the environment. At a certain frequency, the target Q-network used for calculating the target Q-values is updated with the weights of the online Q-network that is actively learning. Once all episodes of the current environment are completed, the weights of the trained agent are saved. Table 5 concisely summarizes the key hyperparameters used in the algorithms, along with their respective values and brief descriptions.

Table 5. Simulation and hyperparameter settings for the algorithms.

Hyperparameter	Description	Value
`width`	Width of the maze environment.	10
`height`	Height of the maze environment.	10
`num_train_environments`	Total number of different training environments to use.	11
`num_episodes_per_environment`	Number of episodes to run in each training environment.	2000
`max_steps_per_episode`	Maximum number of steps allowed in each episode.	1000
`epsilon_start`	Initial value of epsilon for the epsilon-greedy policy.	1.0
`epsilon_end`	Final value of epsilon after decay.	0.01
`epsilon_decay`	Factor by which epsilon is decayed after each episode.	0.9977
`gamma` (`DDQNAgent`)	Discount factor for future rewards in the Q-learning algorithm.	0.99
`update_frequency` (`DDQNAgent`)	Frequency of updating the target model in the DDQNA agent.	10

A significant enhancement in the training phase is the integration of the A* algorithm, which serves as a guide for the agent, providing insights on the shortest path to the goal during each episode. This strategic guidance is pivotal, especially in the initial phases of training, augmenting the agent's capacity to navigate the maze efficiently while still allowing for exploratory behavior as dictated by the epsilon greedy policy.

In the maze environment, we aim for the agent to reach the destination with the minimum number of steps without colliding with any static or dynamic obstacles. The reward function, elaborated in detail in Section 3.3, was designed to align with this objective. Consequently, when evaluating the results of the experiment, we incorporate metrics relevant to our goal. To assess the agent's performance over 2000 training episodes, we considered several key metrics: the percentage of episodes in which the agent successfully reached the goal, the average number of steps taken, the average computation time, the total reward, and the number of distinct cells visited. The optimal result for the agent would be to reach the goal directly via the shortest possible route. However, there are instances in which the agent may visit a particular cell several times before reaching the goal. As long as the agent avoids obstacles and successfully reaches the goal, this behavior can also be deemed successful. The training results for Environments 1-8 can be reviewed in Table 6. A detailed analysis of table reveals several important performance patterns. First, the

DDQNA algorithm consistently achieves the highest percentages of reaching goals in almost all environments. This result indicates that the integration of A* guidance significantly accelerates the learning process by providing structured direction information during the early stages of training. Second, the average step count of DDQNA is substantially lower than that of the baseline algorithms in most environments. This suggests that the hybrid framework not only reaches the goal more frequently but also learns more efficient navigation policies that approximate the shortest paths. Third, although DDQNA introduces additional computational overhead compared to standard DDQN due to the A* planning component, the overall computation time remains significantly lower than that of PPO and DDQN. This shows that the proposed hybrid approach achieves a favorable balance between navigation efficiency and computational cost.

When evaluating the average results over 2000 episodes during the training process, it is important to note that at the beginning of the training, the agent is just starting to learn and is entirely unfamiliar with the environment. We emphasize this to highlight the agent's performance even when learning in an entirely new environment. To thoroughly assess this, we provide detailed training results. Consequently, the training outcomes for the proposed algorithm are noteworthy.

Table 6. Comparison of A2C, PPO, DDQN, DDQN* and DDQNA Algorithms' training results for 8 environments. The percentage of visited cells, average step count, and computation time were calculated by averaging over 2000 episodes. The reached goal percentage indicates the proportion of episodes in which the agent successfully reached the goal out of the 2000 episodes. The improvement column represents the ratio of DDQNA over DDQN*.

Env	Evaluation Metric	A2C	PPO	DDQN	DDQN*	DDQNA	Imp. Ratio
1	Visited Cells Percentage	2.86	17.01	13.25	13.44	17.56	1.31
	Average Step Count	609.14	112.97	333.31	127.91	55.29	2.31
	Calculation Time (sec)	1.54	11.36	0.70	8.46	3.63	2.33
	Reached Goal Percentage	0.00	60.90	6.60	4.12	**55.17**	13.39
2	Visited Cells Percentage	3.48	16.27	14.43	15.79	11.90	0.75
	Average Step Count	250.54	103.08	167.15	102.57	20.66	4.96
	Calculation Time (sec)	0.63	10.37	0.35	6.78	1.33	5.10
	Reached Goal Percentage	0.00	52.15	42.60	35.28	**80.80**	2.29
3	Visited Cells Percentage	2.04	12.77	10.84	14.68	11.14	0.76
	Average Step Count	204.07	56.09	236.80	44.29	20.84	2.13
	Calculation Time (sec)	0.51	5.64	0.51	2.90	1.35	2.15
	Reached Goal Percentage	0.00	71.10	28.50	70.81	**75.95**	1.07
4	Visited Cells Percentage	2.14	14.15	14.34	16.26	14.96	0.92
	Average Step Count	312.23	50.85	173.86	52.63	24.77	2.12
	Calculation Time (sec)	0.78	5.11	0.37	3.46	1.60	2.16
	Reached Goal Percentage	0.05	73.55	38.20	57.48	**77.80**	1.35
5	Visited Cells Percentage	4.15	7.57	11.38	14.12	16.80	1.19
	Average Step Count	145.50	226.53	234.07	136.55	60.27	2.27
	Calculation Time (sec)	0.36	22.78	0.49	9.04	3.97	2.28
	Reached Goal Percentage	0.10	0.15	4.05	2.73	**53.81**	19.71
6	Visited Cells Percentage	2.07	12.25	10.39	10.01	11.33	1.13
	Average Step Count	69.58	108.23	229.62	137.93	27.50	5.02
	Calculation Time (sec)	0.18	10.88	0.48	9.13	1.79	5.10
	Reached Goal Percentage	0.00	41.65	21.10	9.05	**74.37**	8.22
7	Visited Cells Percentage	2.16	16.84	11.66	13.24	13.50	1.02
	Average Step Count	76.51	62.84	207.57	67.85	28.40	2.39
	Calculation Time (sec)	0.20	6.32	0.44	4.47	1.85	2.42
	Reached Goal Percentage	0.10	67.80	23.70	48.67	**77.75**	1.60
8	Visited Cells Percentage	2.28	13.85	13.80	15.71	16.65	1.06
	Average Step Count	255.88	181.32	220.45	98.00	54.24	1.81
	Calculation Time (sec)	0.68	18.24	0.46	6.47	3.56	1.82
	Reached Goal Percentage	0.60	29.90	29.55	34.77	**58.78**	1.69

4.2. Test Process

The testing process begins with loading pre-trained weights into the agent for those environments. Each environment is tested 100 times consecutively, and the results are averaged. **The A* algorithm**, which is utilized for guidance during the training process, **is not used during the testing phase**. The agent interacts with the environment by choosing actions based on the current state. Note that the epsilon value is set to 0.0 for testing, meaning the agent will always exploit its learned knowledge and will not explore random actions. The agent continues to interact with the environment until the episode ends. At each step, the agent performs the chosen action, observes the reward, and the next state. After the episode ends, the performance of the agent is evaluated. Test results for Environments 1–8 can be reviewed in Table 7. In all environments, DDQNA consistently demonstrates better efficiency (lower step count and calculation time) and effectiveness (higher goal achievement rate) compared to other algorithms. The DDQNA algorithm appears to be less affected by changes in the environment, whereas the DDQN* algorithm exhibits significant performance variations depending on the difficulty level of the environment. In the third environment, characterized by the presence of a direct and unobstructed path from the initial point to the goal, both algorithms are observed to perform at a similar level. The fifth environment, characterized by a longer distance, more obstacles, and additional turning points along the path from the start to the goal, poses significant challenges for A2C, PPO, DDQN and DDQN* algorithms. Although the DDQNA algorithm faced difficulties, its performance remained at a relatively stable level. In contrast, other algorithms experienced a substantial decrease in performance, indicating an increased sensitivity to complex navigational challenges compared to the DDQNA.

The standard deviation of the goal achievement rate for the DDQN* algorithm is approximately 38.59, while for the DDQNA algorithm it is significantly lower at approximately 3.39. In comparison, the standard deviations for the other algorithms are approximately 0.0 for A2C, 27.43 for PPO, and 16.84 for the standard DDQN algorithm. This large standard deviation indicates that the performance of the A2C, PPO, DDQN and DDQN* algorithms is highly variable across different environments, suggesting that its effectiveness is greatly influenced by the specific characteristics or challenges of each environment. In contrast, the low standard deviation of the DDQNA goal achievement rate underscores its resilience in maintaining performance under difficult conditions, highlighting a potential advantage in adaptability and robustness.

A comparative summary of the main characteristics of the algorithms is given in Table 8. Despite moderate computational times, the A2C algorithm consistently fails to reach the goal in all cases, showing poor exploration capabilities and limited learning efficiency. However, PPO achieves high success rates in most environments, showing strong policy optimization and generalization. However, the increase in computational time and the variation in performance between environments highlight the challenges in terms of real-time applicability and consistency. The standard DDQN algorithm, despite having the fastest computational time, shows poor performance in terms of goal achievement. With the integration of the proposed reward mechanism, the modified DDQN * algorithm shows a significant improvement over DDQN, especially in environments 2, 3, and 7, but still lacks consistent reliability.

Table 7. Comparison of A2C, PPO, DDQN, DDQN* and DDQNA Algorithms' test results for 8 environments. The percentage of visited cells, average step count, and computation time were calculated by averaging over 2000 episodes. The reached goal percentage indicates the proportion of episodes in which the agent successfully reached the goal out of the 2000 episodes. The improvement column represents the ratio of DDQNA over DDQN*.

Env	Evaluation Metric	A2C	PPO	DDQN	DDQN*	DDQNA	Imp. Ratio
1	Visited Cells Percentage	4.16	15.23	1.0	3.96	16.19	4.09
	Average Step Count	115.92	72.08	1000.0	164.03	19.52	8.4
	Calculation Time (sec)	11.66	7.25	3.45	10.91	1.3	8.39
	Reached Goal Percentage	0	64.0	0	0	**84**	-
2	Visited Cells Percentage	2.0	13.42	2.12	13.28	9.55	0.72
	Average Step Count	234.26	65.13	65.28	15.09	9.61	1.57
	Calculation Time (sec)	23.55	6.55	0.02	1.0	0.64	1.56
	Reached Goal Percentage	0	76.0	0	89	**92**	1.03
3	Visited Cells Percentage	2.0	12.09	1.22	16.48	7.37	0.45
	Average Step Count	179.2	89.2	125.95	49.28	7.96	6.19
	Calculation Time (sec)	18.02	8.97	0.04	3.28	0.53	6.19
	Reached Goal Percentage	0	61.0	0	81	**83**	1.02
4	Visited Cells Percentage	2.0	12.46	9.89	14.98	13.11	0.88
	Average Step Count	191.06	36.19	170.83	96.82	13.37	7.24
	Calculation Time (sec)	19.21	3.64	0.06	6.44	0.89	7.24
	Reached Goal Percentage	0	88.0	47	23	**86**	3.74
5	Visited Cells Percentage	4.0	5.13	2.59	2.83	14.37	5.08
	Average Step Count	111.78	300.41	446.13	107.63	17.16	6.27
	Calculation Time (sec)	11.24	30.21	0.15	7.15	1.14	6.27
	Reached Goal Percentage	0	0	0	0	**87**	-
6	Visited Cells Percentage	2.0	12.1	1.71	2.31	10.46	4.53
	Average Step Count	39.48	37.97	64.66	76.58	11.56	6.62
	Calculation Time (sec)	3.97	3.82	0.02	5.09	0.77	6.61
	Reached Goal Percentage	0	75.0	0	0	**81**	-
7	Visited Cells Percentage	2.0	13.65	5.25	12.08	12.05	1.0
	Average Step Count	75.18	35.06	140.17	15.29	13.59	1.13
	Calculation Time (sec)	7.56	3.52	0.05	1.02	0.9	1.13
	Reached Goal Percentage	0	68.0	0	84	**88**	1.05
8	Visited Cells Percentage	2.0	13.69	3.82	17.03	13.53	0.79
	Average Step Count	194.58	57.19	156.23	301.65	13.85	21.78
	Calculation Time (sec)	19.57	5.75	0.05	20.06	0.92	21.8
	Reached Goal Percentage	0	78.0	1	70	**82**	1.17

Table 8. Comparison of key characteristics across evaluated algorithms.

Algorithm	Goal Achievement	Stability	Computation Time	Notable Characteristics
A2C	Very Low	Stable but Ineffective	Medium	Poor exploration, low success
PPO	High (variable)	Moderate (Inconsistent)	High	Strong learning, costly execution
DDQN	Very Low	Inconsistent	Very Low	Fast but fails to learn effectively
DDQN*	Moderate	Moderate	Medium	Improved with reward design
DDQNA	Very High	Very Stable	Very Low	Best overall, hybrid planning-learning approach

The best overall performance is achieved by the proposed DDQNA algorithm, which combines the strengths of classical A* planning with reinforcement learning. DDQNA consistently achieves the highest goal achievement rates, the lowest average step counts, and the shortest computational times in all environments. In addition, it exhibits the lowest standard deviation in success rates, indicating robust and stable behavior. These findings confirm that the integration of heuristic planning with learning-based adaptation provides a significant advantage for autonomous navigation in dynamic environments when supported by an enhanced reward structure.

In Figures 5 and 6, the path navigated by the agent to reach the goal in a test episode is illustrated for each environment. Test episodes in which the agent reached the goal without collisions are visualized for the DDQN* and DDQNA algorithms. However, it

should be noted that there were three episodes in which the DDQN* algorithm failed to reach the goal. The relative stability observed for DDQNA in the figures reflects its robustness in performance, which corroborates the statistical findings. Furthermore, the significant difference between the total reward values obtained by the DDQN* and DDQNA algorithms is seen in Tables 9 and 10.

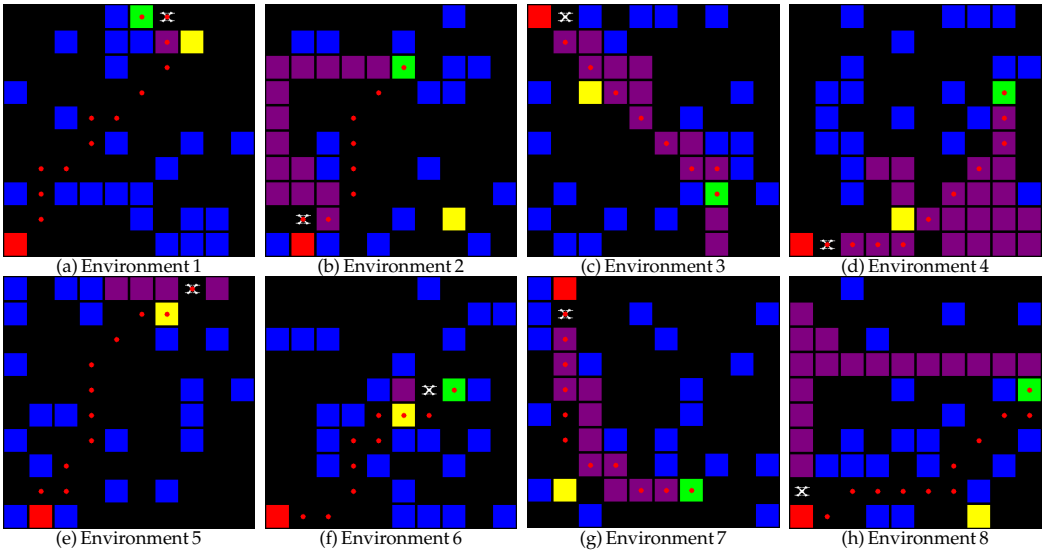

Figure 5. The images show the path (purple) navigated by the agent from the start (green) to the goal (red). The path found by the DDQN* algorithm is compared with the shortest path (red). The shortest path is calculated using the A* algorithm according to the current action rules without considering the dynamic obstacle.

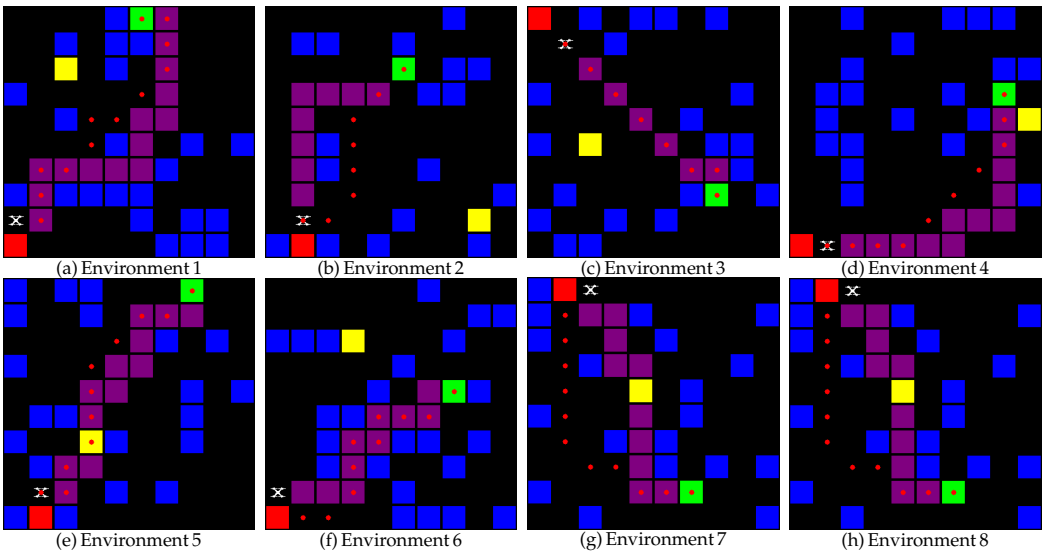

Figure 6. The images show the path (purple) navigated by the agent from the start (green) to the goal (red). The path found by the DDQNA algorithm is compared with the shortest path (red). The shortest path is calculated using the A* algorithm according to the current action rules without considering the dynamic obstacle.

The results clearly demonstrate that DDQNA significantly outperforms DDQN* across all scenarios, achieving higher reward values. During the testing phase, improvements in reward values reached or approached 100%, reflecting substantial gains in the agent's policy efficiency and overall performance. However, considering the goal achievement rate in Environment 2 as shown in Table 7, it becomes evident that a substantial improvement in the reward function does not always provide a comprehensive indication of algorithmic

Table 9. Average of total reward values obtained after training DDQN* and DDQNA algorithms over 2000 episodes for each environment.

Environment	DDQN*	DDQNA	Improvement (%)
1	-41564.46	**-8287.95**	80.06
2	-27386.35	**-744.00**	97.28
3	-8032.47	**-1249.97**	84.44
4	-7913.46	**-933.82**	88.20
5	-42828.64	**-8922.66**	79.17
6	-63962.71	**-3208.42**	94.98
7	-18677.85	**-2294.12**	87.72
8	-40112.40	**-2289.68**	94.29

performance. This highlights the importance of evaluating multiple performance metrics to obtain a more reliable and robust assessment.

These findings indicate that the integration of A* guidance in the DDQNA algorithm not only accelerates learning but also enhances policy stability and effectiveness during generalization. The performance gap between algorithms becomes more pronounced in complex environments, highlighting the advantage of incorporating planning-based approaches into reinforcement learning frameworks.

Table 10. Average of total reward values obtained after training DDQN* and DDQNA algorithms over 100 episodes for each environment.

Environment	DDQN*	DDQNA	Improvement (%)
1	-340570.55	**-60.60**	99.98
2	-58.65	**40.95**	169.82
3	-478389.00	**21.60**	100.00
4	-46191.90	**9.35**	100.02
5	-252324.95	**-83.80**	99.97
6	-238574.50	**4.00**	100.00
7	-495.85	**-13.15**	97.35
8	-1131682.05	**-8.25**	100.00

4.3. Scalability and Statistical Evaluation under Varying Maze Sizes and Obstacle Densities

To address the scalability concerns and assess the generalization ability of the proposed DDQNA algorithm, additional experiments were conducted across three maze sizes: 12×12, 15×15, and 18×18. The corresponding environment configurations are illustrated in Figure 7. Each size was evaluated under varying dynamic obstacle conditions—specifically with 1, 2, and 3 moving obstacles.

For each configuration, the agent was tested in 100 episodes using fixed policies. Statistical significance was evaluated using independent t-tests for continuous variables (e.g., steps, time, visited cells) and the chi-square test for categorical outcomes (e.g., goal success rates). Furthermore, mean ± standard deviation and 95% confidence intervals (CI) were reported to provide a more robust understanding of variability and confidence in the results. Table 11 presents the statistical summary of DDQNA's performance across these variations. The results include the percentage of visited cells, average step count, computation time, and goal success rate, each reported as mean ± standard deviation and accompanied by a 95% confidence interval (CI). For goal success rates, Wilson score intervals were used.

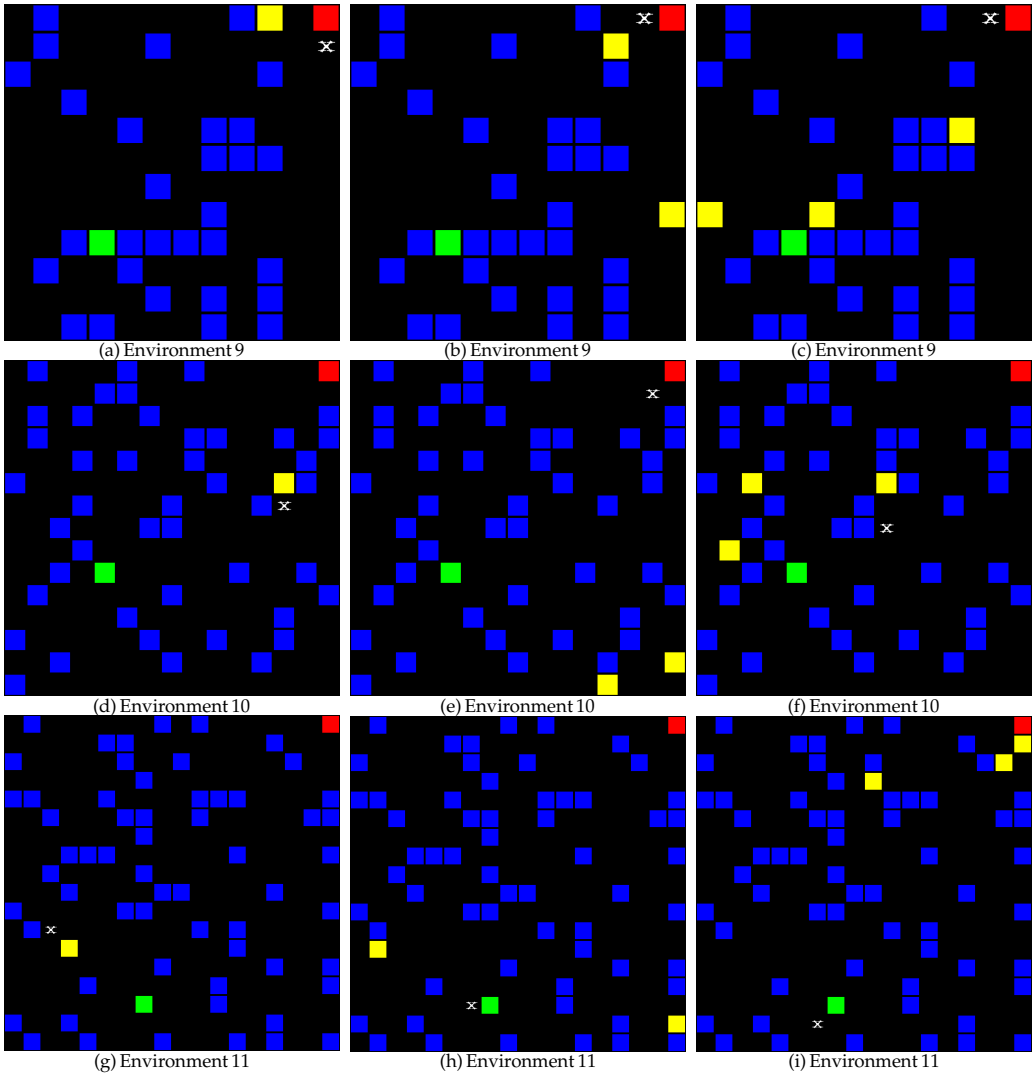

Figure 7. Visualization of DDQNA navigation results under increasing dynamic obstacle density across different maze sizes. Each row corresponds to a different maze dimension (12×12, 15×15, and 18×18), while columns compare 1, 2, and 3 dynamic obstacle configurations. The agent is marked with a white cross, the goal in red, the start in green, static obstacles in blue, and dynamic obstacles in yellow.

An analysis of Table 11 reveals that the DDQNA algorithm achieves higher goal success rates in larger maze environments (15×15 and 18×18) compared to the smaller 12×12 maze, particularly in the presence of three dynamic obstacles. As the maze size increases, the algorithm consistently delivers better performance than in 10×10-like scenarios, highlighting the advantages of additional navigational flexibility. Notably, in the most challenging case with three dynamic obstacles, the 18×18 maze yields significantly superior results, indicating that larger environments mitigate the impact of obstacle-induced constraints more effectively.

The impact of increasing obstacle density on performance metrics is statistically analyzed in Table 12. To test whether the increase from one to three dynamic obstacles produces a genuine performance change, we compared per-episode metrics using two-sided tests—Student's t-test for continuous variables and the χ^2 test for goal achievement rates. A threshold of $p < 0.05$ was adopted for significance.

- **Environment 9 (12×12).** The goal-reached rate dropped from 96% to 52% ($p < 0.001$), and the visited-cell ratio decreased significantly ($p < 0.001$), confirming that additional

Table 11. Statistical summary of DDQNA performance across varying maze sizes and obstacle densities. Values are Mean ± SD with 95% confidence intervals (CI); Goal success shows Wilson 95% CI.

Env	Obstacles	Visited Cells (%)	Average Steps	Computational Time (s)	Reached Goal (%)
9	1	7.83 ± 0.71 [7.69, 7.97]	12.18 ± 1.53 [11.88, 12.48]	0.81 ± 0.10 [0.79, 0.83]	96.0 [90.2, 98.4]
	2	7.76 ± 1.07 [7.54, 7.97]	12.18 ± 2.28 [11.73, 12.63]	0.81 ± 0.15 [0.78, 0.84]	90.0 [82.6, 94.5]
	3	5.63 ± 2.75 [5.08, 6.17]	10.27 ± 18.05 [6.69, 13.85]	0.68 ± 1.20 [0.44, 0.92]	52.0 [42.3, 61.5]
10	1	6.70 ± 1.10 [6.48, 6.92]	17.62 ± 6.41 [16.35, 18.89]	1.17 ± 0.43 [1.08, 1.25]	94.0 [87.5, 97.2]
	2	6.76 ± 0.83 [6.59, 6.92]	33.08 ± 117.42 [9.78, 56.38]	2.20 ± 7.81 [0.65, 3.75]	90.0 [82.6, 94.5]
	3	7.51 ± 1.46 [7.23, 7.79]	43.88 ± 19.15 [40.18, 47.58]	2.91 ± 0.74 [2.77, 3.05]	83.0 [74.9, 88.9]
11	1	6.57 ± 0.98 [6.39, 6.76]	28.67 ± 9.02 [26.88, 30.46]	1.97 ± 0.59 [1.86, 2.08]	91.0 [84.0, 95.2]
	2	7.10 ± 1.20 [6.88, 7.32]	55.96 ± 20.85 [51.79, 60.12]	3.72 ± 1.21 [3.47, 3.97]	87.0 [79.3, 92.4]
	3	8.07 ± 1.35 [7.82, 8.32]	75.46 ± 22.13 [70.92, 80.01]	5.25 ± 1.48 [4.96, 5.54]	82.0 [73.7, 88.6]

Table 12. Comparison of 1 vs 3 dynamic obstacles for DDQNA algorithm across different maze environments. Values are presented as Mean ± SD. Statistical significance is evaluated using t-tests for continuous variables and chi-square test for goal percentages.

Env	Evaluation Metric	1 Obstacle (Mean ± SD)	3 Obstacles (Mean ± SD)	p-value
9	Visited Cells (%)	7.83 ± 0.71	5.63 ± 2.75	0.0000
	Average Steps	12.18 ± 1.53	10.27 ± 18.05	0.2941
	Computational Time (s)	0.81 ± 0.10	0.68 ± 1.20	0.2963
	Reached Goal (%)	96.00 ± 1.96	52.00 ± 5.00	0.0000
10	Visited Cells (%)	6.70 ± 1.10	5.82 ± 2.18	0.0004
	Average Steps	17.62 ± 6.41	43.88 ± 19.15	0.0000
	Computational Time (s)	1.17 ± 0.43	2.91 ± 0.74	0.0000
	Reached Goal (%)	94.00 ± 2.41	83.00 ± 3.64	0.0130
11	Visited Cells (%)	6.57 ± 0.98	8.07 ± 1.35	0.0000
	Average Steps	28.67 ± 9.02	75.46 ± 22.13	0.0000
	Computational Time (s)	1.97 ± 0.59	5.25 ± 1.48	0.0000
	Reached Goal (%)	91.00 ± 2.89	82.00 ± 3.79	0.0471

obstacles measurably hinder navigation efficiency. Differences in average steps and computation time were not significant ($p \approx 0.29$), likely due to high variance caused by a few difficult episodes.

- **Environment 10 (**15×15**).** All four metrics showed statistically significant degradation as the number of obstacles increased. Notably, the goal-reached rate fell from 94% to 83% ($p = 0.013$), and the mean step count more than doubled ($p < 0.001$), indicating penalties in both safety and path length under higher obstacle density.
- **Environment 11 (**18×18**).** The larger maze size amplified this trend: the success of the goal decreased from 91% to 82% ($p = 0.047$), while the percentage of cells visited, the mean steps, and the computation time increased significantly ($p < 0.001$).

In general, the DDQNA algorithm remains robust, maintaining the success of $\geq 82\%$ goals even in the most challenging scenario 18×18 maze. In conclusion, these statistical tests confirm that obstacle density has a measurable and nonrandom impact on both path efficiency and computation time.

4.4. Statistical Evaluation of Reward Shaping

This section presents a statistical comparison of standard DDQN and its reward-shaped variant (DDQN*) to evaluate the isolated impact of the reward function design on

navigation efficiency. The comparative results summarized in Table 13 demonstrate that the DDQN* algorithm, which adopts a reward-shaping strategy while maintaining the same architectural structure as standard DDQN, achieves statistically significant performance gains across a range of dynamic environments. In particular, DDQN* consistently exhibits better exploration, as reflected by significantly higher visited cell ratios in all test cases ($p < 0.001$). Although step count and total reward outcomes vary with environmental complexity, DDQN* yields substantial benefits in scenarios where the baseline DDQN fails to reach the goal. These results indicate that the enhancements to the reward function effectively encourage safer and more goal-directed behavior.

Table 13. Comparison of DDQN and DDQN* across Environments 1–8. Values are presented as Mean ± SD. Statistical significance was assessed using Welch's t-test for continuous metrics and the chi-square test for categorical outcome distributions. Bold p-values indicate statistical significance at $\alpha = 0.05$.

Env	Metric	DDQN	DDQN*	p-value
1	Visited Cells (%)	1.00 ± 0.00	3.96 ± 1.89	< 0.001
	Step Count	10000.00 ± 0.00	164.03 ± 298.58	< 0.001
	Total Reward	-10000.00 ± 0.00	$-340570.55 \pm 1097449.98$	**0.0033**
	Calculation Time (s)	3.45 ± 0.04	10.91 ± 19.87	< 0.001
	End Reason	Max steps (100)	Hit obstacle (100)	1.000
2	Visited Cells (%)	2.12 ± 0.92	13.28 ± 2.52	< 0.001
	Step Count	65.28 ± 100.41	15.09 ± 4.99	< 0.001
	Total Reward	-164.28 ± 100.41	-58.65 ± 312.59	**0.0017**
	Calculation Time (s)	0.02 ± 0.03	1.00 ± 0.33	< 0.001
	End Reason	Hit (100)	Goal (89), Hit (11)	1.000
3	Visited Cells (%)	1.22 ± 0.48	16.48 ± 4.64	< 0.001
	Step Count	125.95 ± 382.31	49.28 ± 217.51	0.083
	Total Reward	-224.95 ± 382.31	$-478389.00 \pm 4764219.39$	0.318
	Calculation Time (s)	0.04 ± 0.13	3.28 ± 14.47	**0.028**
	End Reason	Hit (100)	Goal (81), Hit (19)	1.000
4	Visited Cells (%)	9.89 ± 9.25	14.98 ± 9.21	< 0.001
	Step Count	170.83 ± 447.82	96.82 ± 146.57	0.119
	Total Reward	-175.83 ± 487.12	-46191.90 ± 140763.23	**0.0015**
	Calculation Time (s)	0.06 ± 0.15	6.44 ± 9.75	< 0.001
	End Reason	Hit (53), Goal (47)	Hit (77), Goal (23)	0.533
5	Visited Cells (%)	2.59 ± 0.57	2.83 ± 1.78	0.201
	Step Count	446.13 ± 569.41	107.63 ± 258.87	< 0.001
	Total Reward	-545.13 ± 569.41	-252324.95 ± 911210.05	**0.0068**
	Calculation Time (s)	0.15 ± 0.19	7.15 ± 17.21	< 0.001
	End Reason	Hit (100)	Hit (100)	1.000
6	Visited Cells (%)	1.71 ± 0.46	2.31 ± 0.84	< 0.001
	Step Count	64.66 ± 191.92	76.58 ± 237.25	0.697
	Total Reward	-163.66 ± 191.92	$-238574.50 \pm 1527199.24$	0.122
	Calculation Time (s)	0.02 ± 0.06	5.09 ± 15.78	**0.0018**
	End Reason	Hit (100)	Hit (100)	1.000
7	Visited Cells (%)	5.25 ± 1.44	12.08 ± 3.13	< 0.001
	Step Count	140.17 ± 360.17	15.29 ± 7.81	**0.0008**
	Total Reward	-239.17 ± 360.17	-495.85 ± 4182.59	0.542
	Calculation Time (s)	0.05 ± 0.12	1.02 ± 0.52	< 0.001
	End Reason	Hit (100)	Goal (84), Hit (16)	1.000
8	Visited Cells (%)	3.82 ± 4.96	17.03 ± 4.78	< 0.001
	Step Count	156.23 ± 398.96	301.65 ± 219.97	**0.0017**
	Total Reward	-253.23 ± 399.65	$-1131682.05 \pm 1748208.68$	< 0.001
	Calculation Time (s)	0.05 ± 0.14	20.06 ± 14.63	< 0.001
	End Reason	Hit (99), Goal (1)	Goal (70), Hit (30)	0.661

Although the integration of richer reward signals leads to moderately increased computation time, performance improvements justify this overhead. Furthermore, while chi-square tests on categorical termination outcomes did not reach statistical significance due to distributional imbalance, the marked increase in successful episodes under DDQN* highlights the efficacy of reward-based guidance. In general, the findings validate that

reward shaping alone, without architectural modification, can significantly enhance the scalability and robustness of value-based reinforcement learning in dynamic navigation tasks.

5. Discussion

The simulation results demonstrate that incorporating structured guidance into a value-based reinforcement learning framework significantly improves learning stability, convergence speed, and overall navigation performance (see Tables 6 and 11). Compared to standalone RL baselines such as DDQN, PPO, and A2C, DDQNA achieved consistently higher success rates across 11 randomly generated environments. This finding supports the hypothesis that hybrid planning-learning approaches can overcome some of the fundamental limitations of purely learning-based methods, particularly in environments with dynamically moving obstacles.

A key factor behind this performance improvement is the guided training mechanism. During training, the A* algorithm provides structured action recommendations based on the static map of the environment, reducing the need for random exploration. This significantly accelerates learning and stabilizes convergence. Importantly, this guidance is not used during the test phase; the agent relies solely on the learned policy. The agent's ability to navigate autonomously after training demonstrates that the guidance has been internalized as a persistent policy rather than relying on external support during execution. The ablation analysis further highlights the importance of maintaining a balance between planning-based guidance and learning-driven exploration. Excessive reliance on A* guidance leads to reduced policy generalization and instability, whereas insufficient guidance slows learning and increases variance (see Table 2). The selected integration probability (p) successfully mediates this trade-off, enabling the agent to exploit optimal trajectories while still learning robust navigation strategies.

Reward design plays an important role in the observed performance gains. By explicitly penalizing repeated cell visits and inefficient movements, the proposed reward structure effectively mitigates looping behaviors that commonly arise in navigation tasks. While reward shaping alone improves baseline DDQN performance, the largest gains are obtained when reward shaping is combined with the hybrid planning-learning architecture (DDQNA) (see Table 7). These findings suggest that problem-specific reward design and algorithmic structure are more effective when considered jointly rather than in isolation. Another important aspect of this study is the comprehensive evaluation strategy adopted to assess navigation performance. Rather than relying on a single success-based metric, the proposed framework is evaluated using multiple complementary indicators, including goal-reaching success, step count, visited cell ratio, and cumulative reward. This multi-metric evaluation enables a more detailed analysis of efficiency and consistency, and measuring repeated cell visits helps identify inefficient looping behaviors.

Despite these strengths, several factors must be considered when interpreting the results in the context of real-world robotic deployment. First, the proposed framework operates in a discrete action space with a limited set of motion primitives, whereas physical robots typically require continuous control over velocity and orientation. Second, the Pygame-based simulation environment abstracts away physical dynamics such as sensor noise, friction, and actuation uncertainty. These abstractions may influence performance when transferring the learned policies to real robotic platforms. As recently demonstrated by Ahmic et al. [57], systematically randomizing physical parameters—such as vehicle mass and tire-road friction—during the RL training process significantly enhances an agent's resilience against parametric uncertainties in continuous autonomous driving tasks. It is essential to emphasize that the primary objective of this work is not merely to

provide a ready-to-deploy controller but to demonstrate that the identified inefficiencies in decision-making under dynamic uncertainty can be effectively mitigated.

Another important consideration is the reward structure and its scalability. As maze dimensions increase, cumulative reward values also grow due to the longer trajectories needed to reach the goal, underscoring the necessity of reward normalization in large-scale environments. While the current formulation effectively shapes agent behavior, future implementations should explore scale-invariant reward designs to ensure numerical stability. Additionally, future work will focus on extending DDQNA to continuous control settings and validating the framework in higher-fidelity simulators and physical platforms.

6. Conclusions

This paper introduced DDQNA, a hybrid navigation algorithm that integrates A* path planning with Double Deep Q-Networks to enable autonomous agents to navigate environments containing both static and dynamic obstacles. By improving the action selection mechanism and redesigning the reward function, the proposed framework significantly enhances path-planning efficiency and learning stability. The simulation results demonstrated that DDQNA consistently outperformed baseline reinforcement learning methods, achieving an average success rate of approximately 85% across 11 randomly generated environments (see Table 7). These findings confirm that guided learning can effectively accelerate convergence and improve generalization in dynamic navigation tasks. While the current implementation is based on a 2D grid environment, the results provide a strong foundation for future extensions toward more realistic, perception-driven, and physically grounded navigation systems.

Future work will focus on:

- extending DDQNA to continuous action spaces to support finer motion control and smoother trajectories.
- integrating perceptual inputs such as cameras, LiDAR, or depth data to improve robustness under partial observability.
- validating the framework in higher-fidelity simulators and on physical robotic platforms, with an emphasis on sensor noise, actuation uncertainty, and real-time constraints.

Author Contributions: Conceptualization, H.A.Ö., S.Ç.Y. and Ç.K.K.; methodology, H.A.Ö. and Ç.K.K.; software, H.A.Ö.; validation, H.A.Ö.; formal analysis, H.A.Ö.; investigation, H.A.Ö.; resources, H.A.Ö.; data curation, H.A.Ö.; writing—original draft preparation, H.A.Ö. ; writing—review and editing, H.A.Ö. and Ç.K.K.; visualization, H.A.Ö.; supervision, S.Ç.Y. and Ç.K.K.; project administration, Ç.K.K.; funding acquisition, Ç.K.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the TÜBİTAK projects 2232-118C332 and 1001-121F348, and the Jiangsu Province 100 Foreign Experts Plan BX2022012.

Data Availability Statement: The data supporting the findings of this study were generated by the authors using custom-designed simulation environments. No external datasets were used. The datasets are available from the corresponding author.

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

Abbreviations

The following abbreviations are used in this manuscript:

AVs	Autonomous Vehicles
A2C	Advantage Actor Critic
ALE	Arcade Learning Environment
COLREG	Convention on the International Regulations for Preventing Collisions at Sea
DQN	Deep Q Network
DQL	Deep Q Learning
DDQN	Double Deep Q Network
RL	Reinforcement Learning
RSS	Received Signal Strength
PPO	Proximal Policy Optimization
TD-MOA	Time Dependent Multiobstacle Avoidance Algorithm
UAV	Unmanned Aerial Vehicle

References

1. Shaik, K.; Banerjee, D.; Begum, R.S.; Srikanth, N.; Narasimharao, J.; El-Ebiary, Y.A.; Thenmozhi, E. Dynamic Object Detection Revolution: Deep Learning with Attention, Semantic Understanding, and Instance Segmentation for Real-World Precision. *International Journal of Advanced Computer Science and Applications* **2024**, *15*. https://doi.org/10.14569/IJACSA.2024.0150141.
2. Chen, D.; Ye, B.; Zhao, Z.; Wang, F.; Xu, W.; Yin, W. Change Detection Converter: Using Semantic Segmentation Models to Tackle Change Detection Task. In Proceedings of the 2022 IEEE International Conference on Multimedia and Expo (ICME), 2022, pp. 1–6. https://doi.org/10.1109/ICME52920.2022.9859973.
3. Muñoz, G.; Barrado, C.; Çetin, E.; Salamí, E. Deep Reinforcement Learning for Drone Delivery. *Drones* **2019**, *3*, 72. https://doi.org/10.3390/drones3030072.
4. Chowdhury, M.M.U.; Erden, F.; Guvenc, I. RSS-Based Q-Learning for Indoor UAV Navigation. In Proceedings of the MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM), 2019, pp. 121–126. https://doi.org/10.1109/MILCOM47813.2019.9020894.
5. Zuluaga, J.G.C.; Leidig, J.P.; Trefftz, C.; Wolffe, G. Deep Reinforcement Learning for Autonomous Search and Rescue. In Proceedings of the NAECON 2018 - IEEE National Aerospace and Electronics Conference, 2018, pp. 521–524. https://doi.org/10.1109/NAECON.2018.8556642.
6. Cestero Portu, J.; Quartulli, M.; Metelli, A.M.; Restelli, M. Storehouse: a Reinforcement Learning Environment for Optimizing Warehouse Management. In Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 1–9. https://doi.org/10.1109/IJCNN55064.2022.9891985.
7. Abliz, P. A controlling estimation bias method: Max_Mix_Min estimator for Q-Learning. *J. Supercomput.* **2024**, *80*, 19248–19273. https://doi.org/10.1007/s11227-024-06181-y.
8. van Hasselt, H.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-Learning. In Proceedings of the Proc. 30th AAAI Conf. Artificial Intelligence (AAAI), Phoenix, AZ, USA, 2016; pp. 2094–2100. [Online]. Available: https://dl.acm.org/doi/10.5555/3016100.3016191.
9. Dorling, K.; Heinrichs, J.; Messier, G.G.; Magierowski, S. Vehicle Routing Problems for Drone Delivery. *IEEE Trans. Syst., Man, Cybern. Syst.* **2017**, *47*, 70–85. https://doi.org/10.1109/TSMC.2016.2582745.
10. Mnih, V.; et al. Volodymyr Mnih. *arXiv preprint* **2013**, *arXiv:1312.5602*.
11. Mnih, V.; et al. Human-Level Control Through Deep Reinforcement Learning. *Nature* **2015**, *518*, 529–533. https://doi.org/10.1038/nature14236.
12. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* **2013**, *47*, 253–279. https://doi.org/10.1613/jair.3912.
13. Kersandt, K.; Muñoz, G.; Barrado, C. Self-Training by Reinforcement Learning for Full-Autonomous Drones of the Future. In Proceedings of the 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), 2018, pp. 1–10. https://doi.org/10.1109/DASC.2018.8569503.
14. Anwar, A.; Raychowdhury, A. Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning. *IEEE Access* **2020**, *8*, 26549–26560. https://doi.org/10.1109/ACCESS.2020.2971172.

15. Lin, H.; Ding, W.; Liu, Z.; Niu, Y.; Zhu, J.; Niu, Y.; Zhao, D. Safety-Aware Causal Representation for Trustworthy Offline Reinforcement Learning in Autonomous Driving. *IEEE Robotics and Automation Letters* **2024**, *9*, 4639–4646. https://doi.org/10.1109/LRA.2024.3379805.
16. Peters, J.; Schaal, S. Policy Gradient Methods for Robotics. In Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 2219–2225. https://doi.org/10.1109/IROS.2006.282564.
17. Kober, J.; Peters, J. Policy Search for Motor Primitives in Robotics. *Machine Learning* **2011**, *84*, 171–203. https://doi.org/10.1007/s10994-010-5223-6.
18. Chua, K.; Calandra, R.; McAllister, R.; Levine, S. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In Proceedings of the Advances in Neural Information Processing Systems 31 (NeurIPS 2018), 2018, pp. 4759–4770.
19. Dib, J.; Sirlantzis, K.; Howells, G. A Review on Negative Road Anomaly Detection Methods. *IEEE Access* **2020**, *8*, 57298–57316. https://doi.org/10.1109/ACCESS.2020.2982220.
20. Yu, H.; Hong, R.; Huang, X.; Wang, Z. Obstacle Detection with Deep Convolutional Neural Network. In Proceedings of the 2013 Sixth International Symposium on Computational Intelligence and Design, 2013, Vol. 1, pp. 265–268. https://doi.org/10.1109/ISCID.2013.73.
21. Choi, J.; Lee, G.; Lee, C. Reinforcement Learning-Based Dynamic Obstacle Avoidance and Integration of Path Planning. *Intell. Serv. Robotics* **2021**, *14*, 663–677. https://doi.org/10.1007/s11370-021-00387-2.
22. Wang, C.; Yang, X.; Li, H. Improved Q-Learning Applied to Dynamic Obstacle Avoidance and Path Planning. *IEEE Access* **2022**, *10*, 92879–92888. https://doi.org/10.1109/ACCESS.2022.3203072.
23. Kim, H.; Lee, W. Dynamic Obstacle Avoidance of Mobile Robots Using Real-Time Q-Learning. In Proceedings of the 2022 International Conference on Electronics, Information, and Communication (ICEIC), 2022, pp. 1–2. https://doi.org/10.1109/ICEIC54506.2022.9748647.
24. Ribeiro, T.; Gonçalves, F.; Garcia, I.; Lopes, G.; Ribeiro, A.F. Q-Learning for Autonomous Mobile Robot Obstacle Avoidance. In Proceedings of the 2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 2019, pp. 1–7. https://doi.org/10.1109/ICARSC.2019.8733621.
25. Chronis, C.; Anagnostopoulos, G.; Politi, E.; Dimitrakopoulos, G.; Varlamis, I. Dynamic Navigation in Unconstrained Environments Using Reinforcement Learning Algorithms. *IEEE Access* **2023**, *11*, 117984–118001. https://doi.org/10.1109/ACCESS.2023.3326435.
26. Tabakis, I.M.; Dasygenis, M. Deep Reinforcement Learning-Based Path Planning for Dynamic and Heterogeneous Environments. In Proceedings of the 2024 Panhellenic Conference on Electronics & Telecommunications (PACET), 2024, pp. 1–4. https://doi.org/10.1109/PACET60398.2024.10496999.
27. Pico, N.; Lee, J.; Montero, E.; Auh, E.; Tadese, M.; Jeon, J.; Alvarez-Alvarado, M.S.; Moon, H. Enhancing Autonomous Robot Navigation Based on Deep Reinforcement Learning: Comparative Analysis of Reward Functions in Diverse Environments. In Proceedings of the 2023 23rd International Conference on Control, Automation and Systems (ICCAS), 2023, pp. 1415–1420. https://doi.org/10.23919/ICCAS59377.2023.10316876.
28. Zhou, C.; Huang, B.; Hassan, H.; Fränti, P. Attention-Based Advantage Actor-Critic Algorithm with Prioritized Experience Replay for Complex 2-D Robotic Motion Planning. *Journal of Intelligent Manufacturing* **2023**, *34*, 151–180. https://doi.org/10.1007/s10845-022-01988-z.
29. Zhai, D.; Yang, D.; Chen, J.; Luo, Z.; Yu, M.; Zhou, Z. Model for the Cooperative Obstacle-Avoidance of the Automated Vehicle Swarm in a Connected Vehicles Environment. *IET Intelligent Transport Systems* **2023**, *17*, 1137–1151. https://doi.org/10.1049/itr2.12359.
30. Xing, X.; Ding, H.; Liang, Z.; Li, B.; Yang, Z. Robot Path Planner Based on Deep Reinforcement Learning and the Seeker Optimization Algorithm. *Mechatronics* **2022**, *88*, 102918. https://doi.org/10.1016/j.mechatronics.2022.102918.
31. Sinha, A.; Macaluso, A.; Klusch, M. Nav-Q: Quantum Deep Reinforcement Learning for Collision-Free Navigation of Self-Driving Cars. *Quantum Machine Intelligence* **2025**, *7*, 19. https://doi.org/10.1007/s42484-024-00226-4.
32. Almazrouei, K.; Kamel, I.; Rabie, T. Dynamic Obstacle Avoidance and Path Planning through Reinforcement Learning. *Applied Sciences* **2023**, *13*, 8174. https://doi.org/10.3390/app13148174.

33. Wu, J.; Huang, C.; Huang, H.; Lv, C.; Wang, Y.; Wang, F.Y. Recent advances in reinforcement learning-based autonomous driving behavior planning: A survey. *Transportation Research Part C: Emerging Technologies* **2024**, *164*, 104654. https://doi.org/10.1016/j.trc.2024.104654.
34. Bilban, M.; İnan, O. Optimizing Autonomous Vehicle Performance Using Improved Proximal Policy Optimization. *Sensors* **2025**, *25*, 1941. https://doi.org/10.3390/s25061941.
35. Nie, J.; Zhang, G.; Lu, X.; Wang, H.; Sheng, C.; Sun, L. Reinforcement learning method based on sample regularization and adaptive learning rate for AGV path planning. *Neurocomputing* **2025**, *614*, 128820. https://doi.org/10.1016/j.neucom.2024.128820.
36. Tang, Z.; Fu, F.; Lu, G.; Chen, D. Reinforcement Learning for Autonomous Agents: Scene-Specific Dynamic Obstacle Avoidance and Target Pursuit in Unknown Environments. *IEEE Access* **2024**, *12*, 145496–145510. https://doi.org/10.1109/ACCESS.2024.3463732.
37. Maw, A.A.; Tyan, M.; Lee, J.W. iADA*: Improved Anytime Path Planning and Replanning Algorithm for Autonomous Vehicle. *Journal of Intelligent & Robotic Systems* **2020**, *100*, 1005–1013. https://doi.org/10.1007/s10846-020-01240-x.
38. Jin, J.; Zhang, Y.; Zhou, Z.; Jin, M.; Yang, X.; Hu, F. Conflict-based search with D* Lite algorithm for robot path planning in unknown dynamic environments. *Computers and Electrical Engineering* **2023**, *105*, 108473. https://doi.org/10.1016/j.compeleceng.2022.108473.
39. Yu, X.; Wang, Y. A time dimension-added multiple obstacles avoidance approach for unmanned surface vehicles. *Ocean Engineering* **2022**, *252*, 111201. https://doi.org/10.1016/j.oceaneng.2022.111201.
40. Maw, A.A.; Tyan, M.; Nguyen, T.A.; Lee, J.W. iADA*-RL: Anytime Graph-Based Path Planning with Deep Reinforcement Learning for an Autonomous UAV. *Applied Sciences* **2021**, *11*, 3948. https://doi.org/10.3390/app11093948.
41. Luo, L.; Zhao, N.; Zhu, Y.; Sun, Y. A* Guiding DQN Algorithm for Automated Guided Vehicle Pathfinding Problem of Robotic Mobile Fulfillment Systems. *Comput. Ind. Eng.* **2023**, *178*, 109112. https://doi.org/10.1016/j.cie.2023.109112.
42. Keselman, A.; Ten, S.; Ghazali, A.; Jubeh, M. Reinforcement Learning with A* and a Deep Heuristic, 2018.
43. Xu, C.; Liu, Z.; Hu, C.; Li, X. Improved Hybrid A* Algorithm Obstacle Avoidance Strategy Based on Reinforcement Learning. In Proceedings of the 2023 42nd Chinese Control Conference (CCC), 2023, pp. 4077–4082. https://doi.org/10.23919/CCC58697.2023.10239886.
44. Liu, X.; Zhang, D.; Zhang, T.; Cui, Y.; Chen, L.; Liu, S. Novel Best Path Selection Approach Based on Hybrid Improved A* Algorithm and Reinforcement Learning. *Appl. Intell.* **2021**, *51*, 9015–9029. https://doi.org/10.1007/s10489-021-02303-8.
45. Dan, H.; Peng, H. 3D Path Planning of UAV Based on Improved Reinforcement Learning. In Proceedings of the Proc. SPIE 12594, Second International Conference on Electronic Information Engineering and Computer Communication (EIECC 2022), 2023, Vol. 12594, p. 1259405. https://doi.org/10.1117/12.2671556.
46. OpenAI. Safety Gym, 2020. Last accessed December 2, 2025.
47. DeepMind. DeepMind Lab, 2017. Last accessed December 2, 2025.
48. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 10 2012, pp. 5026–5033. Last accessed December 2, 2025, https://doi.org/10.1109/IROS.2012.6386109.
49. Rohmer, E.; Singh, S.P.N.; Freese, M. CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework. In Proceedings of the Proc. of The International Conference on Intelligent Robots and Systems (IROS), 2013. Last accessed December 2, 2025.
50. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In Proceedings of the Field and Service Robotics, 2017, [arXiv:1705.05065]. Last accessed December 2, 2025.
51. Community, P. Pygame, 2023. Last accessed December 2, 2025.
52. Bulut, V. Optimal Path Planning Method Based on Epsilon-Greedy Q-Learning Algorithm. *J. Braz. Soc. Mech. Sci. Eng.* **2022**, *44*, 106. https://doi.org/10.1007/s40430-022-03399-w.
53. Montesinos López, O.A.; Montesinos López, A.; Crossa, J., Fundamentals of Artificial Neural Networks and Deep Learning. In *Multivariate Statistical Machine Learning Methods for Genomic*

Prediction; Springer International Publishing: Cham, 2022; pp. 379–425. https://doi.org/10.1007/978-3-030-89010-0_10.

54. Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; de Freitas, N. Dueling Network Architectures for Deep Reinforcement Learning, 2016.
55. Yang, Y.; Zhang, K.; Liu, D.; Song, H. Autonomous UAV Navigation in Dynamic Environments with Double Deep Q-Networks. In Proceedings of the 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), 2020, pp. 1–7. https://doi.org/10.1109/DASC50938.2020.9256455.
56. Rachmawati, D.; Gustin, L. Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. In Proceedings of the J. Phys.: Conf. Ser., 4th Int. Conf. on Computing and Applied Informatics (ICCAI), 2020, Vol. 1566, p. 012061. https://doi.org/10.1088/1742-6596/1566/1/012061.
57. Ahmic, K.; Ultsch, J.; Brembeck, J.; Winter, C. Reinforcement Learning-Based Path Following Control with Dynamics Randomization for Parametric Uncertainties in Autonomous Driving. *Applied Sciences* **2023**, *13*. https://doi.org/10.3390/app13063456.