Highlights

A Map-Informed Dense Reward Function for Efficient Robot Motion Planning Using Deep Reinforcement Learning

Zahra Khan, Khawaja Fahad Iqbal, Muhammad Tauseef Nasir, Yasar Ayaz

- This research introduced an active SLAM-based reward function that integrates map information to enhance environmental perception and encourages efficient exploration in complex environments.
- Furthermore, jerk-based and distance-to-goal rewards were proposed to minimize oscillations, ensure smooth navigation, and encourage shorter paths.
- Conducted a comparative analysis of SAC, TD3, and DDPG to evaluate generalization and robustness in unseen sparse and cluttered environments.
- SAC achieved superior performance in both sparse and cluttered environments compared to TD3 and DDPG. Hence, it demonstrates efficient navigation and generalization to unseen environments.

A Map-Informed Dense Reward Function for Efficient Robot Motion Planning Using Deep Reinforcement Learning

Zahra Khan[a,b], Khawaja Fahad Iqbal[a,b,*], Muhammad Tauseef Nasir[c] and Yasar Ayaz[a,b]

[b]Intelligent Robotics Lab (IRL), National Center of Artificial Intelligence (NCAI), National University of Sciences and Technology (NUST), Sector H-12, Islamabad, 44000, Islamabad, Pakistan

[b]Department of Robotics & Artificial Intelligence, School of Mechanical & and Manufacturing Engineering (SMME), National University of Sciences and Technology (NUST), Sector H-12, Islamabad, 44000, Islamabad, Pakistan

[c]Department Mechanical Engineering, School of Mechanical & and Manufacturing Engineering (SMME), National University of Sciences and Technology (NUST), Sector H-12, Islamabad, 44000, Islamabad, Pakistan

ARTICLE INFO

Keywords:
Autonomous Navigation
Deep Reinforcement Learning
Soft Actor-Critic (SAC)
Twin Delayed Deep Deterministic Policy Gradient (TD3)
Deep Deterministic Policy Gradient (DDPG)
Robot Operating System (ROS)

ABSTRACT

Developing an effective motion planning algorithm is essential for autonomous robots to navigate challenging environments. Sampling-based algorithms explore high dimensional spaces efficiently through random sampling. However, they face challenges in the navigation of complex and cluttered environments due to random sampling and slow convergence. Recent progress in Deep Reinforcement Learning (DRL) mitigates these challenges by learning optimal policies through interaction with the environment. This enables autonomous agents to achieve faster convergence, adapt to complex environments, and navigate more effectively. However, relying on the episodic and sparse reward function leads to inefficient exploration and generates suboptimal paths. To address this issue, this research proposes a reward function that integrates map-based information. This reward improves environmental perception, enhances navigation efficiency, and generalizes to unseen environments. Moreover, the distance and trajectory smoothness rewards generate shorter paths and reduce oscillations in robot movement. Our proposed dense reward function is tested and compared with other reward functions from the literature, as well as with DRL algorithms such as Soft Actor Critic (SAC), Twin Delayed Deep Deterministic Policy Gradient (TD3) and Deep Deterministic Policy Gradient (DDPG) in both sparse and cluttered environments. Experimental results demonstrate that SAC achieved superior performance. In sparse environment SAC achieved 100% success rate, outperforming TD3 at 98.44% and DDPG at 85.94%. In cluttered environments SAC achieved 87.5% success rate, significantly surpassing TD3 at 46.9% and DDPG at 15.6%. These results highlight SAC greater efficiency in navigating unseen environments, and its ability to generalize better compared to other algorithms.

1. Introduction

Motion planning plays a fundamental role in enabling autonomous robots to navigate efficiently through complex and cluttered environments [36], ensuring that they can avoid obstacles, optimize paths, and perform tasks with precision in real-world scenarios. In real-world applications, such as warehouse automation [25], indoor navigation [6], and search and rescue missions [3] require robots to plan collision free and optimal paths. Safe navigation [28] and efficient exploration [10] in unfamiliar environments remain major concerns in the field of mobile robotics. In real-world surroundings, robots operate with limited knowledge of their surroundings, making it difficult to devise efficient and collision-free paths. These challenges increase in complex and cluttered environments [31], where rapid adaptation is crucial for generating optimal and collision-free routes.

Graph-based algorithms such as Dijkstra [7] and A* [12] find the shortest route through the environment by building a network of connected nodes that represent potential routes. These techniques are suitable for known environments, but they are less effective in unknown and dynamic environments [14]. Sampling-based algorithms, including Probabilistic Road Map (PRM) [15], and Random Exploring Random Tree (RRT) [20], generate feasible paths by randomly sampling configuration spaces and are effective for high-dimensional spaces. However, they struggle in dynamic [41], and cluttered environments [9] where frequent replanning is required, and they often generate suboptimal, inefficient routes. Optimization-based algorithms, such as genetic algorithms [13], generate paths by iteratively optimizing

zahra.khan@nust.edu.pk (Z. Khan); fahad.iqbal@smme.nust.edu.pk (K.F. Iqbal); muhammad.tauseef@smme.nust.edu.pk (M.T. Nasir); yasar@smme.nust.edu.pk (Y. Ayaz)
ORCID(s): 0000-0001-6711-2574 (K.F. Iqbal)

solutions to produce efficient and smooth trajectories. However, they are computationally demanding, struggle in unfamiliar environments, and heavily rely on accurate models.

In recent years, Deep Reinforcement Learning (DRL) [27] has evolved as a promising solution to address the limitations of traditional motion planning algorithms [27]. DRL allows robots to learn to traverse the environment directly from experience, thus enabling them to navigate without requiring for manually generated heuristics and pre-built maps [22]. DRL-based approaches iteratively interact with the environment and learn optimal policies and can adapt to complex and unseen environments. This enables robots to navigate effectively in complex and high dimensional spaces.

Despite numerous advantages of DRL, it faces significant challenges. These include limited generalization and high sample complexity. DRL struggles with inefficient exploration [5] and prolonged training time in unfamiliar environments. Furthermore, DRL approaches rely on sparse reward functions [21], which provide episodic reward only once the goal is reached, thereby hindering efficient exploration. In large-scale and complex environments, the lack of a reward function often results in inefficient exploration, slow convergence, and the learning of suboptimal policies. These limitations restrict the ability of DRL to generalize and reduce its efficiency in complex and unknown environments.

To overcome these challenges, this research proposes a dense reward function. This dense reward function encourages more efficient exploration and reduces reliance on sparse goal-based rewards by using real-time SLAM information to provide continuous reward function during navigation. This approach improves path robustness, enhance exploration efficiency, and accelerates policy learning. A comparative analysis is conducted, which includes a comparison of various reward functions from the literature, and the performance of our reward function was tested with DRL algorithms, including Soft Actor Critic (SAC), Twin Delayed DDPG (TD3), and Deep Deterministic Policy Gradient (DDPG). Our dense reward function and SAC achieved the highest cumulative rewards and success rates in both sparse and cluttered environments. The comparison results indicate better efficiency and generalization to unseen environments. The primary contributions of this research are given as

- This research proposes a reward function that incorporates an active SLAM-based reward function to enhance environmental perception and ensure efficient exploration in complex and large-scale environments. It also introduces jerk minimization, and distance to the goal rewards are introduced to generate smoother, more stable navigation, and shorter trajectories.
- Performed a comparative evaluation of SAC, TD3, and DDPG to assess their robustness and generalization in unseen and complex environments.
- The performance of our proposed reward function is also compared with reward functions from existing literature to evaluate its effectiveness.
- The comparison of algorithms and reward functions is evaluated using performance metrics such as path length, traversal time, step size, linear and angular jerk, and episodic reward.
- Our proposed dense reward function is tested with various DRL algorithms, including Soft Actor Critic (SAC), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Deep Deterministic Policy Gradient (DDPG).

The remainder of this paper is organized as follows. Section 2 provides a review of related work and existing approaches to motion planning. Section 3 presents the methodology, including the proposed dense reward function and the implementation of SAC, TD3, and DDPG. Section 4 discusses the experimental setup and evaluation metrics. Section 5 presents the results and performance analysis of the proposed framework. Finally, Section 6 concludes the paper with key findings and suggestions for future work.

2. Literature Review

Autonomous robots require motion planning to generate smooth and collision-free trajectories. Classical algorithms such as Simulation Location Mapping (SLAM) [add] separate planning and control. The global planner computes the feasible path, and the local planner ensures real-time control and obstacle avoidance. However, this results in inefficient

navigation, as the generated path is not adaptable [29] to changes in real time. Classical motion planning algorithms are classified into graph-based, optimization-based, sampling-based, and reactive algorithms.

2.1. Classical Motion Planning

Graph-based algorithms, including Dijkstra [7], A* [12], and D* [34] are widely used path planning algorithms. These approaches build a discrete graph of the environment to find the shortest optimal path using a cost function. They are suitable in static and low dimensional spaces but struggle in complex and high dimensional spaces. Moreover, they are unable to operate in highly dynamic and changing environment.

Sampling-based algorithms are widely used in high-dimensional environments due to their ability to explore complex environments. These algorithms randomly sample points in configuration space and gradually build a graph of connected nodes to create a feasible and asymptotically optimal path. Their common algorithms include Probabilistic Roadmaps (PRM) [15] and Rapidly Exploring Random Trees (RRT) [20]. These approaches are effective in continuous and high-dimensional spaces, but face difficulty in highly dynamic and cluttered environments due to slow convergence. Despite advances such as RRT * and Sparse-RRT * [2], these algorithms cannot provide optimal solutions, limiting their deployment in real-time applications.

Reactive-based algorithms computes low level motion commands directly from real time sensor data. These algorithms include Artificial Potential Field (APF) [17], enabling real-time motion, by attracting the robot towards the goal position and repelling it from obstacles. These algorithms are computationally efficient and highly responsive and well suitable for dynamic environments. However, due to a lack to information of the environment they are prone to local minima, limiting their performance in complex and cluttered environments.

Optimization-based algorithm ensures that motion and safety constraints are satisfied while finding paths by minimizing the cost function. Optimization-based algorithms, including Particle Swarm Optimization (PSO) [43], Genetic Algorithm (GA) [13], and Random Search Optimization (RSO) [40] are used to build optimal paths. However, these algorithms are computationally demanding, tend to converge slowly, and may produce suboptimal solutions. As a result, they are less capable of operating in large and complex environments.

2.2. Learning-Based Motion Planning

Recent research focuses on learning-based algorithms [1] that increase robustness, efficiency, and adaptability with the objective to reduce the challenges of traditional algorithms. In contrast to traditional approaches, learning-based methods use data to extract motion patterns and environment constraints, enabling improved navigation in challenging situations. Learning-based algorithms, including probabilistic models [19], supervised learning [32], and imitation learning [24], are crucial methods that do not rely on intricate optimization computations or explicit geometric maps. Learning-based motion planning is further improved by deep learning algorithms, which offer systems the ability to detect obstacles, predict pathways, and make sequential decisions in challenging situations. Moreover, Convolutional Neural Networks (CNNs) [42] enable obstacle detection and safe path prediction, while recurrent neural networks (RNNs) [16] and long short-term memory (LSTMs) [35] lead decision-making over time. Graph Neural Networks (GNNs) [44] are also used to create complex trajectories by modelling spatial relationships. These methods increase flexibility in real time, but they need huge datasets and often struggle in unseen and highly dynamic situations.

Imitation learning [24] is another popular learning-based technique in motion planning. This approach uses human-provided demonstrations to enable robots to mimic expert driving behaviors. Robots can learn complex navigation patterns that are challenging to program manually by imitating expert movements using techniques like behaviour cloning [4] and dataset aggregation [30]. Robots can navigate accurately and smoothly in environments that are like the training scenarios. Imitation learning, however, frequently falls short in unfamiliar, extremely dynamic, or unstructured environments where the robot comes across scenarios not addressed in the demonstrations. Furthermore, errors can be accumulated during execution, particularly if the model has not seen a situation during training. Recent studies have increasingly used deep reinforcement learning [27] for motion planning to overcome these drawbacks and enhance generalisation in unseen environments.

2.3. Deep Reinforcement Learning for Motion Planning

Deep Reinforcement Learning (DRL) combines planning and control and improves adaptability in challenging situations by learning actions directly from sensor data. DRL are categorized into value-based and policy-based algorithms. Value-based deep reinforcement learning (DRL) algorithms generate optimal motion planning decisions by estimating the expected return of actions. The Deep Q-Network (DQN) [27] combines Q-learning [38] with deep neural networks to operate in high-dimensional spaces. However, it cannot work in continuous action spaces and suffers from overestimation bias and instability. Double DQN (DDQN) [33] enhances value estimation by splitting action selection from evaluation, but issues such as ineffective exploration and sample inefficiency still exist. Weighted Duelling Double DQN (WD3QN) [39] adds adaptive weighting and enhances stability significantly, but it increases the complexity of the model.

Policy-based DRL methods are perfect for continuous action spaces because they learn policies that translate states to actions. Proximal Policy Optimization (PPO) [37] is frequently used for motion planning and continuous control. PPO ensures reliable learning by avoiding large policy updates using clipped objective functions, but it requires more samples. Several policy-based approaches are based on actor-critic frameworks [18], which combine direct policy learning (actor) and value estimation (critic) to handle continuous action spaces efficiently. Although the Deep Deterministic Policy Gradient (DDPG) algorithm [23] uses an actor-critic architecture to achieve smooth control in continuous environments, it has poor exploration and suffers from overestimation bias. Twin Delayed DDPG (TD3) [8] is introduced to mitigate DDPG limitations by delayed policy updates, improving stability and reducing overestimation bias. The Soft Actor-Critic (SAC) [11] enhances exploration, ensures steady learning, and operates effectively in complex and continuous environments.To increase generalization efficiency and stability, we propose a continuous, SLAM-based dense reward function. This study enhances safe and efficient motion planning by integrating this reward with the Soft Actor-Critic (SAC) algorithm. While [26] utilized a sparse and episodic reward strategy with SAC to support policy learning, their approach suffers from limited sensor range and struggles in complex or unfamiliar environments. In contrast, our dense reward leverages SLAM-generated map information to guide the agent more effectively toward the goal, enabling faster convergence and better generalization. Additionally, a trajectory smoothness reward promotes stable and fluid navigation, further improving safety and adaptability in unseen environments.

3. Methodology

Motion planning is essential in complex and high-dimensional environments. Traditional motion planning algorithms rely on a prebuilt map of the environment. This research proposed using Deep Reinforcement Learning (DRL) to navigate in the environment without map information. DRL uses sensor information and selects the optimal action using a learned policy. This enables the robot to efficiently navigate towards the goal position in unseen and complex environments. Proposed navigation framework Figure 1.

3.1. State space

The state space s includes absolute linear v and angular ω velocities, relative position to the goal position, and lidar data. State space vector given as

$$s = \begin{bmatrix} p & v & \omega & l \end{bmatrix}^T \tag{1}$$

The laser data is divided into 21 equal groups, and in each group, the lowest value of the laser data forms the state space. The raw sensor data is given to the neural network to learn the policy.

3.2. Action Space

Once the robot receives the above-mentioned input, the RL algorithm generates the action commands.

$$a_t = \begin{bmatrix} v_t & \omega_t \end{bmatrix} \tag{2}$$

to interact with the environment. The action space consists of continuous linear velocity $v_t \in [0, 1]$ and angular velocity $\omega_t \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ enabling smooth and precise motion control. These actions are applied at each time step to steer the robot toward the goal while avoiding obstacles.

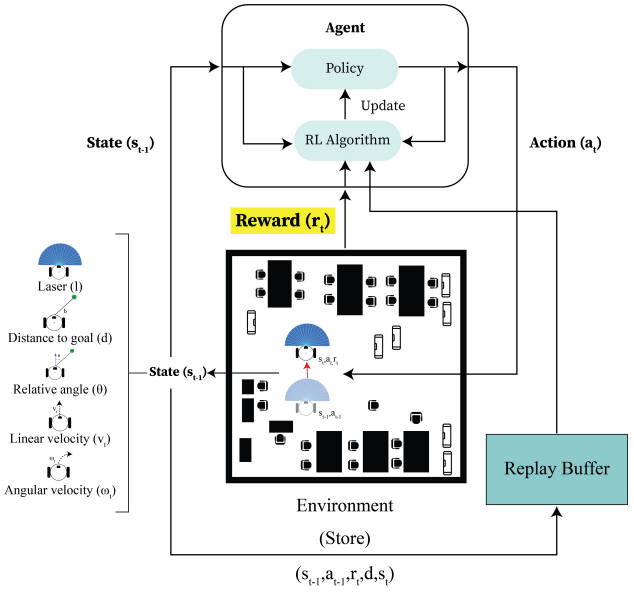


Figure 1: Reinforcement learning-based motion planning framework for autonomous navigation.

3.3. Reward Function

This research proposed a dense reward function, which selects optimal actions and allows the robot to reach the goal position without colliding with the obstacles. This reward function is also capable of generating smooth trajectories by lowering linear and angular jerks and generalizes to unseen and complex environments. Dense reward function represented as:

$$R = \begin{cases} 200, & \text{if } D_t < \eta_D \\ -100, & \text{if collision} \\ r_{\text{vel}} + r_{\text{safety}} + r_{\text{map}} + r_{\text{dist}} + r_{\text{smooth}}, & \text{otherwise} \end{cases} \tag{3}$$

whereas

$$r_{\text{vel}} = \frac{v_t - |\omega_t|}{2} \tag{4}$$

To promote smoother and more efficient paths, it r_{vel} encourages the robot to move forward by taking linear velocity v_t and discourages excessive turning by penalizing taking angular velocityω_t.

$$r_{safety}(l) = \begin{cases} 1 - l, & l < 1 \\ 0, & l \geq 1 \end{cases}, \quad l = \min(\text{laser}) \tag{5}$$

This reward penalizes the robot when the minimum laser distance l is less than $1.0m$. The closer the robot gets to an obstacle, the larger the penalty $(l - 1)$. If the robot maintains a safe distance $(l \geq 1)$, no penalty is applied. This encourages the robot to navigate efficiently while avoiding obstacles. This reward function is further divided by 2 to normalize it, keeping its values within a smaller and more stable range for training.

The purpose of the map reward term is to encourage the robot to navigate towards the target and also to emphasize exploring new areas. This reward discourages the robot from returning to previously explored areas that do not contribute to reaching the goal position and also reduces the local minima issue.

$$G_t = \sum(Oc_t + Fc_t) - \sum(Oc_{t-1} + Fc_{t-1}) \tag{6}$$

The number of occupied cells at time t is denoted by Oc_t, while the number of free cells at time t is denoted by Fc_t. ratio represented as r_{map} as follows:

$$r_{\text{map}} = \frac{G_t}{d} \tag{7}$$

where the current distance to the goal is represented by d. This encourages the robot to map new areas when it is far from the goal and prioritizes reaching it.

$$r_{\text{dist}} = \frac{10 \times (dist_{\text{old}} - d)}{\text{total\_distance}} \tag{8}$$

This reward function allows the robot to move towards the goal position faster by giving a higher reward for stepping towards the goal position. This reward function allows the robot to move towards the goal position faster by giving a higher reward for stepping towards the goal position.

$$r_{\text{smooth}} = -\frac{|10 \times (\omega_t - \omega_{\text{prev}})|}{4} \tag{9}$$

To encourage smooth turns and reduce jerky motion, r_{smooth} penalizes abrupt shifts in angular velocity between successive actions.

3.4. Policy Learning Algorithms

3.4.1. Deep Deterministic Policy Gradient Algorithm (DDPG)

Deep Deterministic Policy Gradient Algorithm (DDPG) is a model-free RL algorithm, indicating it learns by interacting with the environment directly and doesn't require an environment model. It is based on an actor-critic architecture and contributes significantly to the continuous action space. The approach known as Deep Deterministic Policy Gradient (DDPG) uses two actor-critic neural network pairs: two actors (policy networks) and two critics (Q-networks). In this approach, a current and a target network are used for both the critic and the actor. The target networks provide stable training by slowly tracking the updates of the current networks. The target Q network uses the tuple from the environment replay buffer $(s_t, a_t, r_t, s_{t+1}) \sim D$ to estimate the discounted target Q value for the state s_{t+1} and action a_{t+1} produced by the target network. The reward from the replay buffer is added to the replay buffer for the discounted target estimate, and the result is compared with the current Q network to compute the temporal difference $L(Q_\theta, D)$.

$$L(Q_\theta, D) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim D} \left[\left(Q_\theta(s_t, a_t) - \left(r_t + \gamma(1-d) Q_\theta^{\text{target}}(s_{t+1}, \pi_\theta^{\text{target}}(s_{t+1})) \right) \right)^2 \right] \tag{10}$$

where d is a binary indicator for the end of the episode and γ is the discount factor. To maximize the Q-value determined by the critic network Q_θ, the current policy π_θ generates actions in a continuous action space. The goal of the actor network is to maximize the Q_θ by optimizing the policy π_θ.

$$L(\pi_\theta, D) = \max_\theta \ \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim D} \left[Q_\theta \left(s_t, \pi_\theta(s_t) \right) \right] \tag{11}$$

Finally, DDPG updates the target network softly, this soft update is essential for maintaining training stability since it reduces the possibility of the Q-value oscillations that are common of deterministic policies. The network diagram of the DDPG is shown in Figure 3

3.4.2. Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) introduced reduces the overestimation bias problem in the DDPG algorithm. TD3 introduces the twin pair of critic networks and the target network. The minimum of both the twin networks contributes to computing the loss, but this leads to an underestimation of the bias. The loss function is given below:

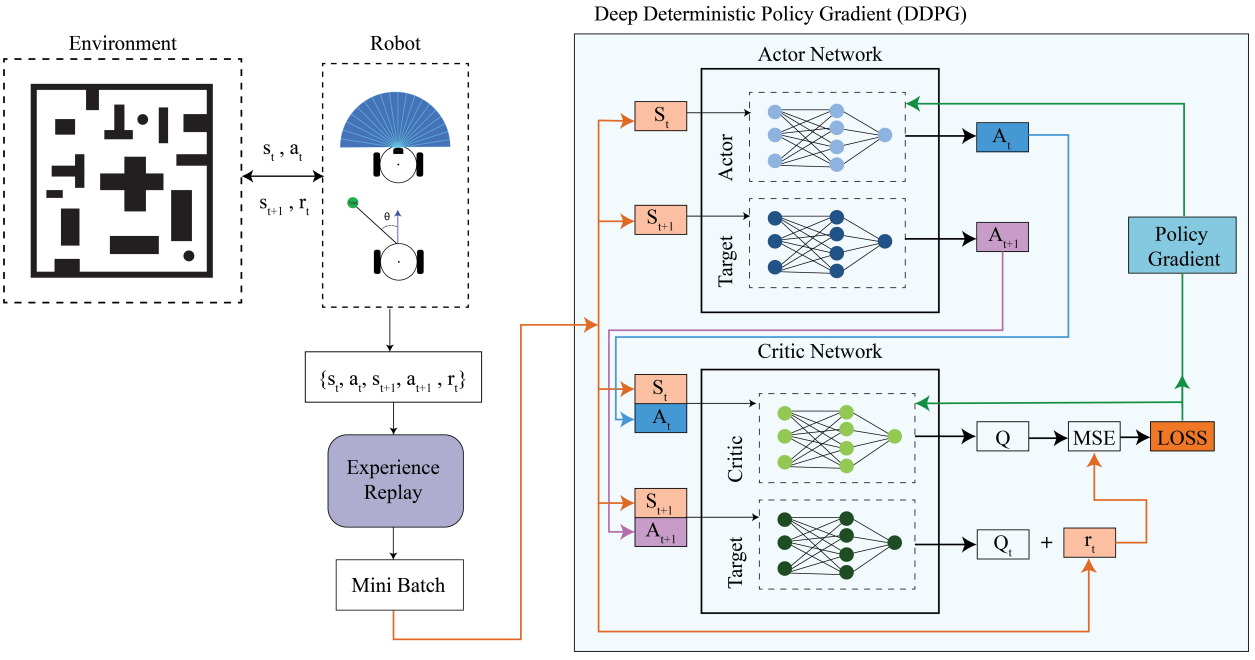


Figure 2: Deep Deterministic Policy Gradient (DDPG) network diagram.

$$Q_{\theta^{\text{target}}}\left(s_{t+1}, \pi_{\theta^{\text{target}}}(s_{t+1})\right) = \min\left(Q_{\theta_1^{\text{target}}}\left(s_{t+1}, \pi_{\theta^{\text{target}}}(s_{t+1})\right),\ Q_{\theta_2^{\text{target}}}\left(s_{t+1}, \pi_{\theta^{\text{target}}}(s_{t+1})\right)\right) \tag{12}$$

$$\begin{aligned} L(Q_\theta, D) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}\Big[&\left(Q_{\theta_1}(s_t,a_t) - \left(r_t + \gamma(1-d)Q_{\theta^{\text{target}}}(s_{t+1}, \pi_{\theta^{\text{target}}}(s_{t+1}))\right)\right)^2 \\ &+ \left(Q_{\theta_2}(s_t,a_t) - \left(r_t + \gamma(1-d)Q_{\theta^{\text{target}}}(s_{t+1}, \pi_{\theta^{\text{target}}}(s_{t+1}))\right)\right)^2\Big] \end{aligned} \tag{13}$$

TD3 adds delayed actor updates, which cause the actor network to update less often than the critic, to stabilize actor learning. As a result, the actor is less likely to depend on unstable critic updates. The robustness of the policy is further improved by clipping the noise added to the action space; this ensures that the noise regulates actions during the training phase. The target actor and the target critic are subjected to soft updates, based on the current network their weights are probabilistically updated with a delay instead of at each epoch. The network diagram of the TD3 shown in Figure **??**

3.4.3. Soft Actor Critic (SAC)

Soft Actor-Critic (SAC) combines exploration and learning efficiency in continuous action spaces by optimizing three interconnected functions: the state-value function, Q-function, and policy function. By choosing the lowest Q-value between two critics, TD3 is subject to overestimation bias, which may result in conservative action estimates. Soft Actor-Critic (SAC) is an entropy-maximizing reinforcement learning method that balances exploration and exploitation by combining expected reward with policy entropy. The goal of SAC is to maximize the policy's entropy and expected return, which is represented by the objective function that follows:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\,\mathcal{H}\left(\pi(\cdot \mid s_t)\right)\right] \tag{14}$$

The trade-off between exploitation (reward) and exploration (entropy) is managed by α. Bellman residual minimization is used to update the soft Q-function:

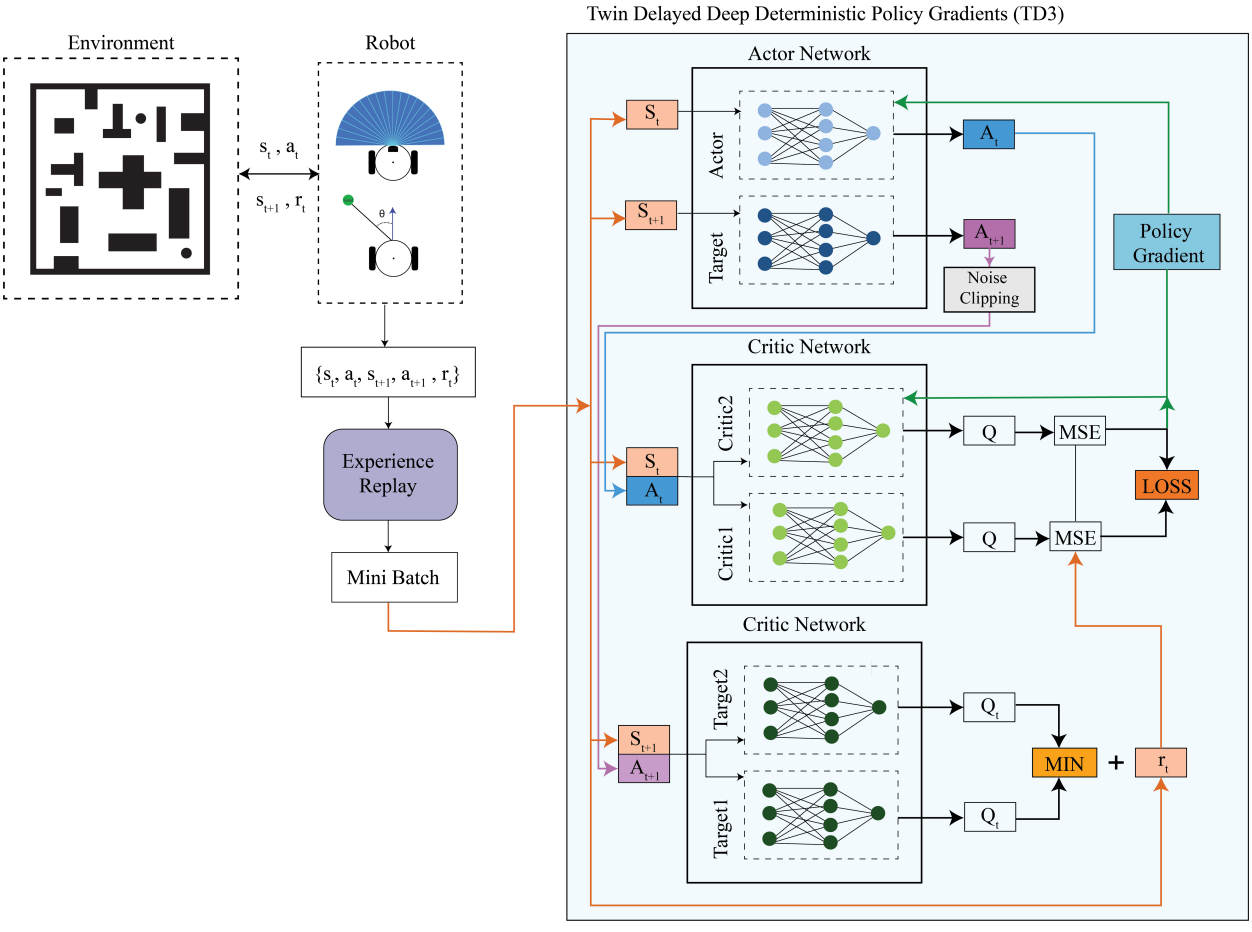


Figure 3: Twin Delayed Deep Deterministic Policy Gradients (TD3) network diagram.

$$J(Q) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}\left[\left(Q(s_t,a_t) - \left(r_t + \gamma\, \mathbb{E}_{a_{t+1}\sim\pi}\left[Q(s_{t+1},a_{t+1}) - \alpha \log \pi(a_{t+1} \mid s_{t+1})\right]\right)\right)^2\right] \tag{15}$$

The updated policy aims to maximize entropy and the expected Q value.

$$J(\pi) = \mathbb{E}_{s_t\sim D,\, a_t\sim\pi}\left[\alpha \log\left(\pi(a_t \mid s_t)\right) - Q(s_t,a_t)\right] \tag{16}$$

By directing SAC's policy towards a balanced exploration and reward-driven action selection, these equations strengthen the system's robustness in challenging circumstances. The network diagram of the SAC is shown in Figure 4

4. Experimental Setup and Evaluation

This section outlines the network architecture and experimental setup used to evaluate the proposed approach. The structure of the deep-reinforcement learning (DRL) models, training parameters, and performance metrics used to assess SAC, TD3, and DDPG models.

4.1. Network Structure

All agents were trained under identical circumstances to ensure a fair and consistent comparison. This requires using the same training environments, hyperparameters, and network architectures. All algorithms adhere to the actor-critic framework, in which the actor network converts observed states into continuous actions, and the critic network

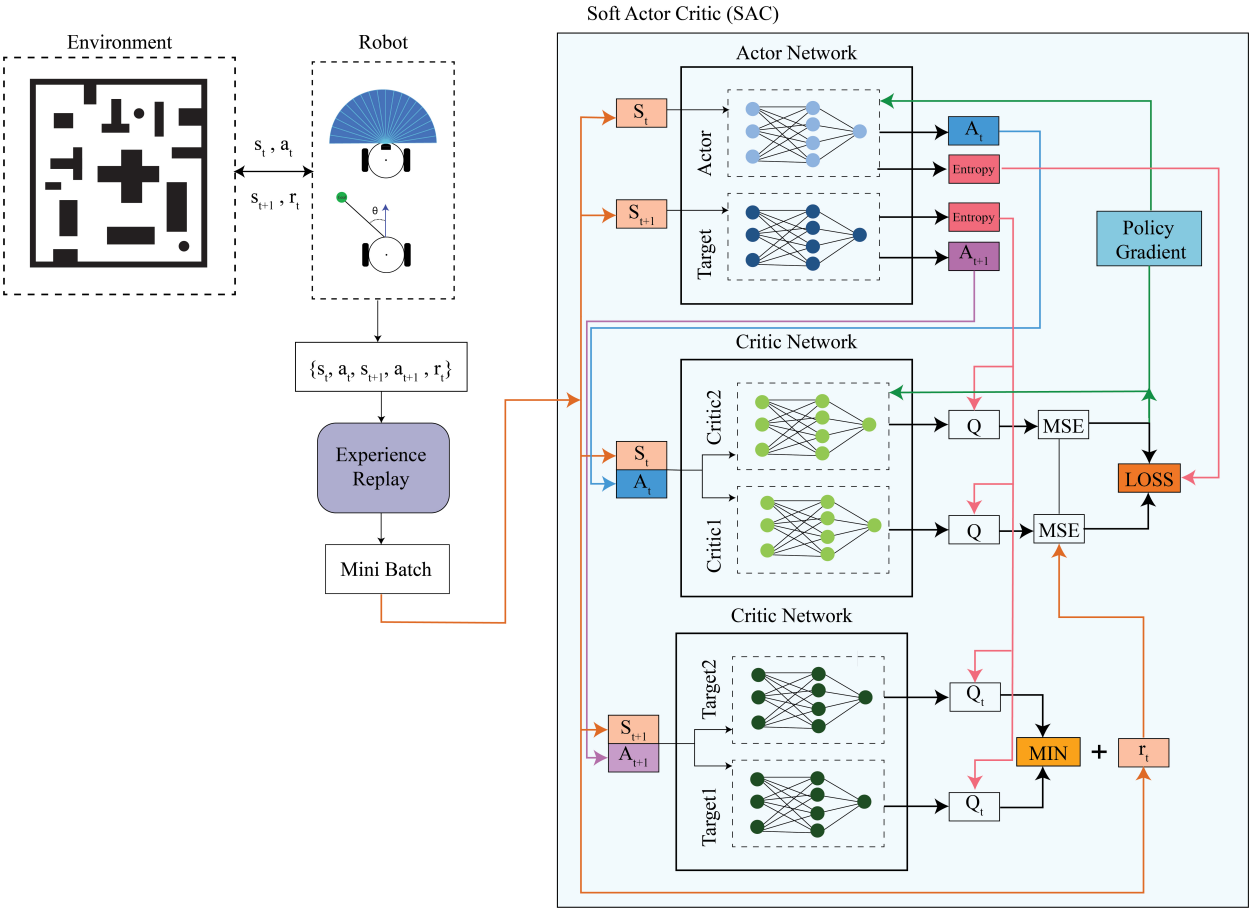


Figure 4: Twin Delayed Deep Deterministic Policy Gradients (TD3) network diagram.

evaluates the value of state-action pairs. The actor network is composed of a state space input layer, two dense hidden layers, and an output action layer. The action generated by the actor is given and the state space as input to the critic network, which generates a Q value.

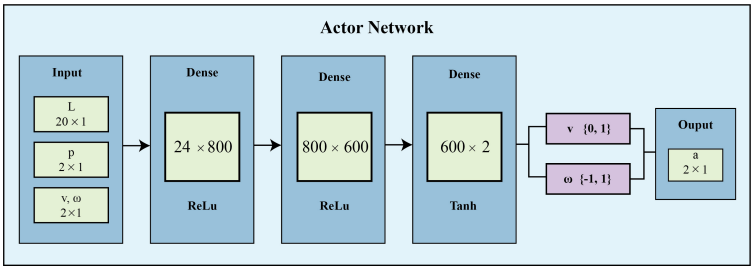


Figure 5: Actor Network mapping states to continuous action space.

The Gazebo simulator and ROS Noetic were used to create a simulated environment for the agents' training. A differential drive robot, Pioneer-3dx, with a 2D lidar, is used for autonomous navigation. All episodes start with a random goal and starting position, and the robot learns to navigate towards the goal while minimizing path length and motion jerks. The PyTorch deep learning framework was used to simulate on a computer with an NVIDIA GPU RTX 3070. Over 15 thousand episodes were used to train each agent, and their generalization was evaluated in unseen environments. The hyperparameters of all algorithms are shown in Table 1.

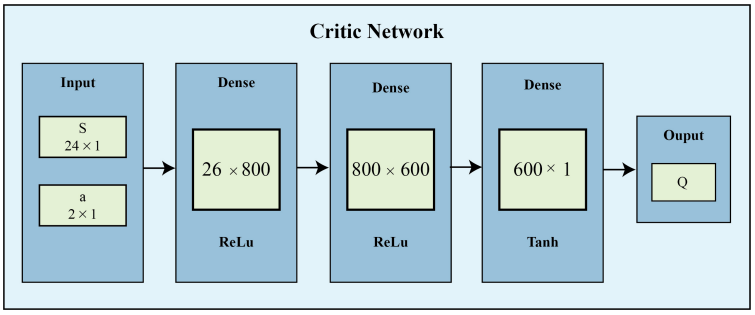


Figure 6: Critic Network estimating Q values from states and action space.

Table 1
Hyperparameter settings for all DRL algorithms

Hyperparameter	DDPG	TD3	SAC
Actor learning rate	1e-4	1e-4	3e-4
Critic learning rate	1e-3	1e-3	3e-4
Batch size	256	256	256
Discount factor (γ)	0.99	0.99	0.99
Replay buffer size	10^6	10^6	10^6
Target update rate (τ)	0.005	0.005	0.005
Policy noise std. dev.	–	0.2	–
Noise clip range	–	0.5	–
Policy delay	–	2	–
Target entropy	–	–	–3
Temperature (α)	–	–	Auto-tuned
Optimizer	Adam	Adam	Adam
Hidden layers (Actor/Critic)	[800, 600]	[800, 600]	[800, 600]
Activation function	ReLU	ReLU	ReLU

4.2. Performance Metrics

In order to evaluate the efficiency and smoothness of the robot's movement, several key performance metrics are defined. These metrics not only provide quantitative measures of the robot's behavior, but also help identify areas for improvement in navigation and trajectory planning. The following performance metrics are essential for assessing the overall performance of the robot navigation:

1. **Traversal time:** The total traversal time required by the robot to reach the goal. This metric is important because it reflects the efficiency of the robot's trajectory planning. A shorter execution time generally indicates a more efficient path.

$$T = \int_0^{T_{\text{tot}}} dt \tag{17}$$

2. **Path length:** The goal distance covered from the robot's initial position to the goal position. The path length is represented by:

$$P_{\text{len}} = \int_{x_i}^{x_g} \sqrt{1 + (f'(x))^2}\, dx \tag{18}$$

In the above equation, x_i and x_g denote the coordinates of the initial and goal position, respectively.

3. **Accumulated jerk:** The jerk refers to abrupt changes in acceleration. It quantifies the smoothness of a robot's movement along its trajectory. Specifically, the accumulated linear jerk measures the sudden variations in the robot's linear velocity, whereas the accumulated angular jerk captures sudden changes in angular velocity. These jerks are defined as follows:

$$J_{\text{acc}} = \frac{1}{T_{\text{tot}}} \int_0^{T_{\text{tot}}} \left[\ddot{v}(t)^2\right] \, dt \tag{19}$$

$$\zeta_{\text{acc}} = \frac{1}{T_{\text{tot}}} \int_0^{T_{\text{tot}}} \left[\ddot{w}(t)^2\right] \, dt \tag{20}$$

In above equation J_{acc} tells accumulated linear jerk, and ζ_{acc} represents angular jerk.

4. **Steps Size:** The total number of steps taken by the robot to reach the goal position to evaluate the efficiency of the learned policy. It represents the sum of steps taken by the robot in multiple episodes.

$$S_{\text{total}} = \sum_{i=1}^{N} S_i \tag{21}$$

Where S_i denotes the number of steps in the i-th episode, and N is the total number of episodes.

5. **Average Success Rate:** The average success rate represents the proportion of the successful episodes out of 64 episodes, calculated as:

$$\text{SR}_{\text{avg}} = \frac{1}{64} \sum_{i=1}^{64} (S_i \leq T_{\text{goal}}) \tag{22}$$

The equation for SR_{avg} computes the average success rate by summing the successful episodes (where $S_i \leq T_{\text{goal}}$) and dividing by the total number of episodes.

5. Experimental Results

To evaluate the robustness of the deep reinforcement learning algorithm, a comparative analysis of different algorithms is performed. These include the deep-deterministic policy gradient (DDPG), the twin-delayed deep-deterministic policy gradient (TD3), and the soft actor-critic (SAC). All algorithms are trained on a single environment and tested on multiple unseen environments. The algorithms were evaluated using a set of performance metrics. This experimental setup provides critical insights into the DRL algorithm's performance in different environments.

5.1. Evaluation Scenario

This research evaluates motion planning algorithms based on deep reinforcement learning in various unseen environments, including sparse and cluttered scenarios. In both of these evaluation environments, all DRL algorithms, including SAC, TD3, and DDPG are evaluated for their robustness and generalization in unseen environments. Furthermore, the proposed reward function is evaluated with two other reward functions to assess its performance in unseen environments. The list of reward functions is shown in Table 2.

The sparse reward function provides episodic reward only when the goal is reached and penalizes the robot when it collides with the obstacle. This reward function also emphasizes taking linear velocity into account and making fewer turns as higher linear velocity will allow the robot to navigate to the goal position faster. This reward function is computationally efficient, but it provides limited feedback and uses a delayed reward function.

This slam-based reward function provides an episodic reward when the target position is reached and penalizes for collision. Furthermore, it allows the robot to explore unexplored areas and limits exploration of areas that have already been explored. This reward function also emphasizes efficient navigation to the goal position. This reward function prioritizes exploration of unknown spaces, allowing the robot to continue mapping when far from the goal. As a result,

Table 2
Reward Functions used for the comparison of DRL Algorithms

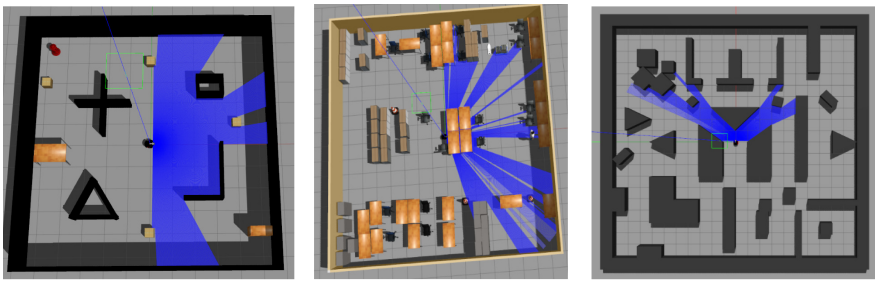
Serial No	Reward Function	Type	Reference
1	$\mathcal{R}_g = \begin{cases} r_g & \text{if } D_t < \eta_D \\ r_c & \text{if collision} \\ v - \lvert\omega\rvert & \text{otherwise} \end{cases}$	Sparse	[5]
2	$\mathcal{R}_{\text{slam}} = \begin{cases} r_g & \text{if } D_t < \eta_D \\ r_c & \text{if collision} \\ v - \lvert\omega\rvert + r_{\text{slam}} & \text{otherwise} \end{cases}$	SLAM	[26]
3	$\mathcal{R} = \begin{cases} r_g & \text{if } D_t < \eta_D \\ r_c & \text{if collision} \\ r_{\text{vel}} + r_{\text{safety}} + r_{\text{map}} + r_{\text{dist}} + r_{\text{smooth}} & \text{otherwise} \end{cases}$	Dense	Proposed

it can delay the goal-reaching behavior, especially when the robot is not close to the target.

The proposed reward function integrates an SLAM-based reward to encourage exploration of unknown areas while equally emphasizing goal-directed navigation. A distance-to-goal reward function is included to enable the robot to reach the target more efficiently. Additionally, an angular jerk minimization term is incorporated to reduce oscillations and promote smoother motion. The reward also gives a large value when the robot reaches the goal position and applies a penalty in the case of a collision. This reward function divides the lidar laser scan into 21 equal regions and uses the minimum value in each region to detect potential collisions. Overall, this proposed reward effectively balances exploration, motion smoothness, and goal reaching, making it suitable for navigating complex environments.

5.2. Sparse Environment Evaluation

In this section, the performance of Deep Reinforcement Learning (DRL) algorithms are evaluated in an unseen sparse environment. All algorithms are tested with different reward functions listed in Table 2. DRL algorithms are trained in the same environment shown below and tested on a sparse environment shown below.



(a) (b) (c)

Figure 7: Image (a) shows the training environment of the gazebo, (b) shows the sparse environment, and (c) shows the cluttered environment for testing.

In a sparse environment, 64 episodes are used for testing. All episodes start from the same initial position, while the goal position varies to evaluate performance. The success rate of all algorithms are presented in Figure 8.

Figure 8 demonstrates that SAC with the proposed reward function achieved a 100% success rate, while other reward functions with SAC and all TD3 variants achieved 98.44% success rate. This indicates strong generalization and robustness of SAC when trained with the proposed dense reward function. On the other hand, DDPG-Dense and DDPG-Sparse performed reasonably well, achieving a success rate of 85.94% and 82.81%, respectively. However, DDPG-SLAM underperformed significantly with a success rate of only 34.38%. These results highlight the effectiveness of SAC and TD3 with the proposed reward function in sparse environments, whereas DDPG appears to be more sensitive to reward design and may require additional tuning for reliable performance. The algorithms were further evaluated on

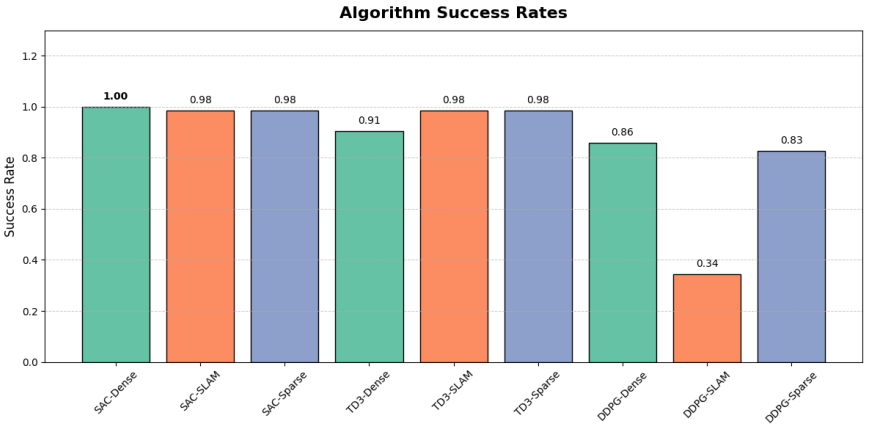


Figure 8: Success rate of all DRL algorithms in sparse environments.

common successful episodes in which the robot reached the goal. The analysis focuses on the number of steps taken, time required, and path length to reach the goal, with the results presented in the image Figure 9.

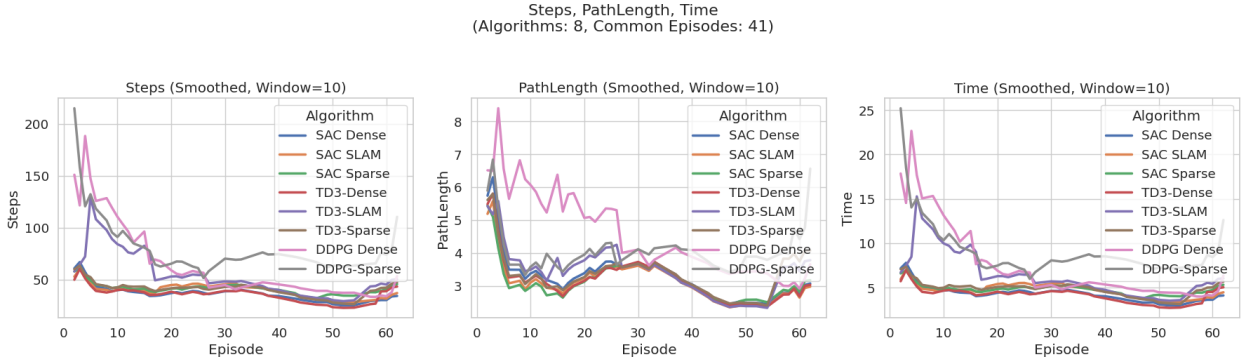


Figure 9: Comparison of all DRL algorithms in a sparse environment.

Figure 9 the results indicate that SAC dense significantly outperformed all other algorithms across all metrics including path length, time and step size. Although SAC dense requires more steps and time initially, it shows significant improvements in later episodes, leading to faster reduction in all the metrics. SAC slam follows a similar trend but has higher values to traversal time, path length, and step size compared to SAC dense. The TD3 dense exhibit noticeable improvements over time, but lacks the efficiency of SAC dense. In contrast, DDPG algorithms tend to converge more slowly, show higher values of all metrics. Ultimately, SAC with a proposed dense reward function is the most efficient algorithm, providing superior performance in minimizing traversal time, path length, and steps to reach the goal position.

Figure 10 shows the linear and angular jerks of all algorithms in 64 common episodes. In terms of average linear jerk, TD3 with a dense reward function achieved the lowest value, indicating smoother linear motion. Other TD3 variants showed higher linear jerks. While SAC dense exhibited lower linear jerk compared to SAC with slam and SAC with sparse reward. On the other hand, DDPG sparse demonstrates lower linear jerk relative to the other DDPG variants.

Furthermore, SAC with the proposed dense reward function achieved the lowest angular jerk, indicating SAC has a smoother rotational motion. Among all algorithms, SAC Dense consistently generates the lowest angular jerks, and SAC with other reward functions showed higher angular jerks. TD3 Dense also performed well in minimizing both linear and angular jerks, with TD3 slam displaying comparable angular smoothness. TD3 sparse showed slightly lower angular jerks compared to other TD3 variants. In contrast, DDPG variants exhibited higher angular jerk, resulting in

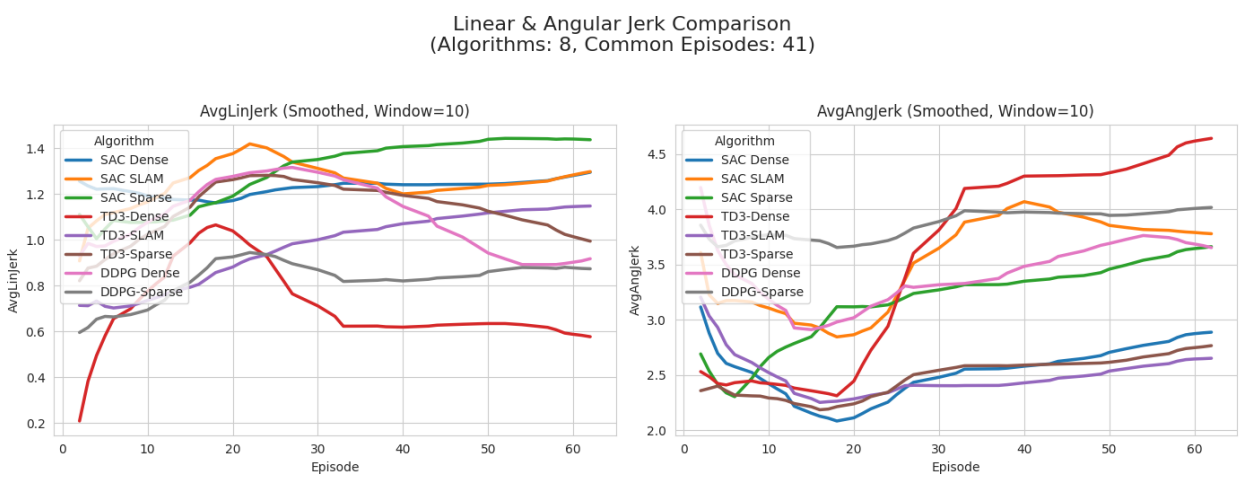


Figure 10: Comparison of all DRL algorithms sparse environment

unstable and jerky motion. Overall, TD3 dense and SAC dense demonstrated the best performance in minimizing linear and angular velocities, respectively. These results highlight their ability to generate smooth and stable trajectories in complex environments.

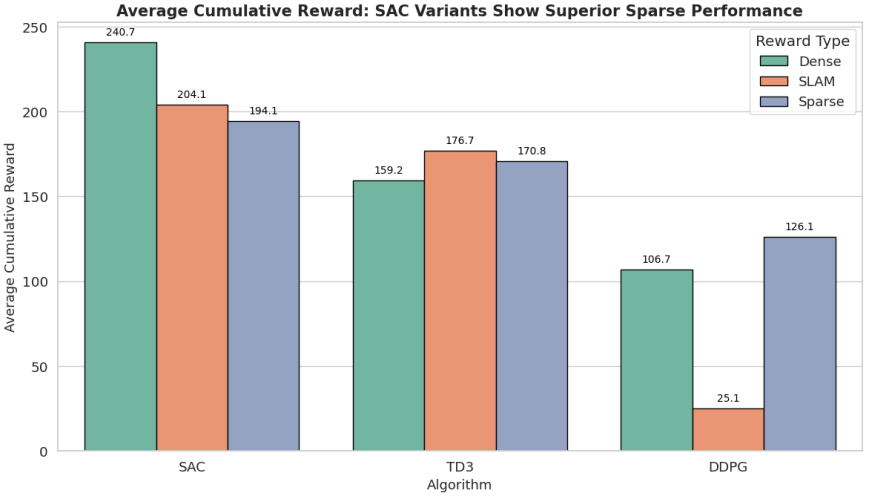


Figure 11: Average cumulative reward function of all algorithms in a sparse environment.

Figure 11 testing result shows that SAC dense outperformed other algorithms in terms of average cumulative reward of common episodes. SAC obtained the highest reward of 240.7, followed by SAC slam with 204.1 and SAC sparse with 194.1 in the testing phase. TD3 also demonstrates competitive performance. TD3 dense obtained a 159.2 cumulative reward, while TD3 slam and TD3 sparse achieved 176.7 and 170.8, respectively. In comparison, DDPG performs significantly worse with its dense reward function, achieving 106.7, and DDPG sparse only 126.1. The results highlight that SAC with our proposed reward function provides superior performance across all reward functions. The above results demonstrate that SAC with the proposed dense reward function outperformed all algorithms in sparse environments, showing better generalization and adaptability in unseen environments.

5.3. Cluttered Environment Evaluation

All algorithms were tested in a cluttered environment to assess their adaptability, generalization, and efficiency in an unseen environment. The cluttered environment is shown in Figure 7.

The Figure 12 shows that SAC dense outperformed all other algorithms, achieving the highest success rate of 87.50%. SAC sparse and SAC slam achieved success rates of 85.94% and 75.0%, respectively. TD3 slam and TD3 sparse achieved success rates of 71.88% and 62.50%, respectively, while TD3 dense achieved a success rate of 46.88%. Both DDPG dense and DDPG sparse performed poorly, with rates of 41.31% and 15.6%, respectively, and DDPG slam had the lowest success rate. Thus, SAC dense demonstrates the best performance in this sparse environment.

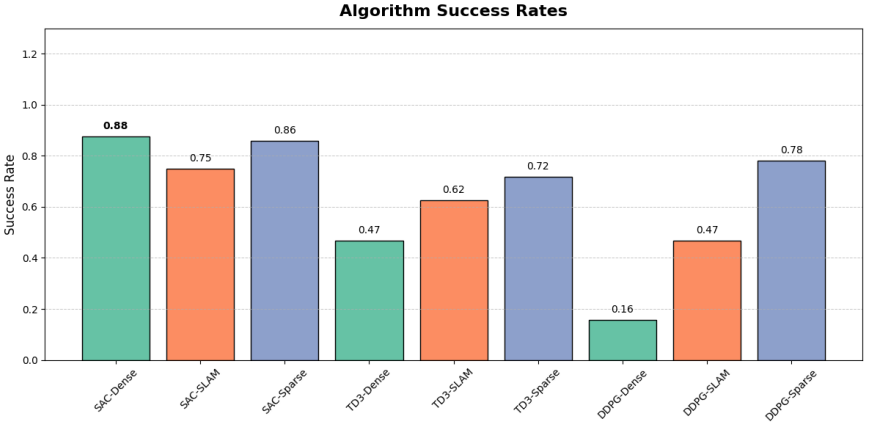


Figure 12: Success rate of all DRL algorithms in cluttered environments.

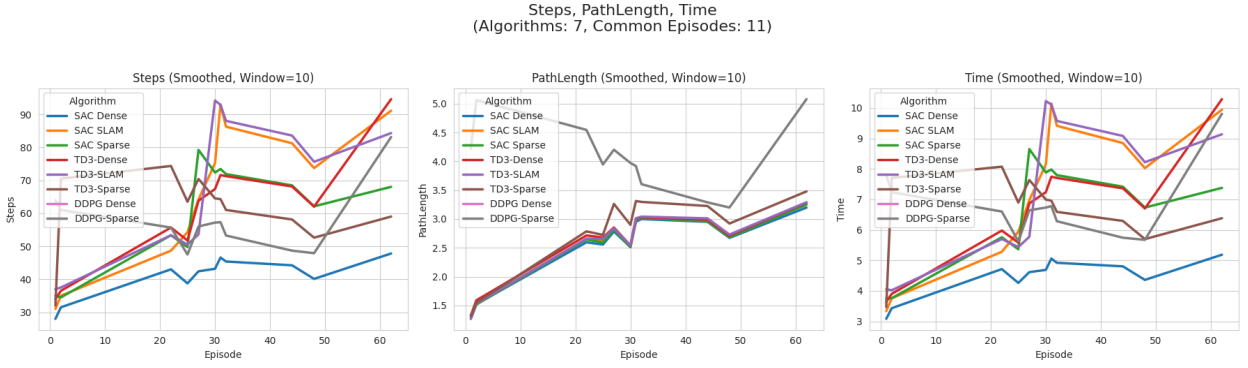


Figure 13: Cluttered environment for algorithm evaluation.

The Figure 13 results indicate that SAC with the proposed dense reward function outperformed all other algorithms across all metrics, including step size, path length, and traversal time, showing the highest efficiency. SAC dense consistently requires the fewest steps, the shortest path length, and the least traversal time to reach the goal position. SAC slam and SAC sparse also perform well, with slightly higher step counts and path length, but still exhibit efficient performance overall. TD3 dense, TD3-slam, and TD3 sparse exhibit moderate performance, characterized by higher step and path lengths, as well as traversal time, compared to SAC. DDPG dense and DDPG sparse perform the worst, requiring significantly more steps, longer path length, and more traversal time. Overall, SAC with our proposed dense reward function demonstrates superior efficiency in all aspects, whereas DDPG performs the worst.

Figure 13 shows, the comparison result of linear and angular jerks, highlighting that SAC dense outperformed other algorithms and achieved the lowest values of linear and angular jerks. The results demonstrate the smooth and efficient navigation in an unseen, cluttered environment. The SAC slam and SAC sparse, TD3 slam and TD3 sparse exhibit a more significant increase in jerk, indicating oscillations. DDPG dense and DDPG sparse perform worst with the highest jerk values in both linear and angular motion, indicating poor control and less efficient learning. Overall, SAC

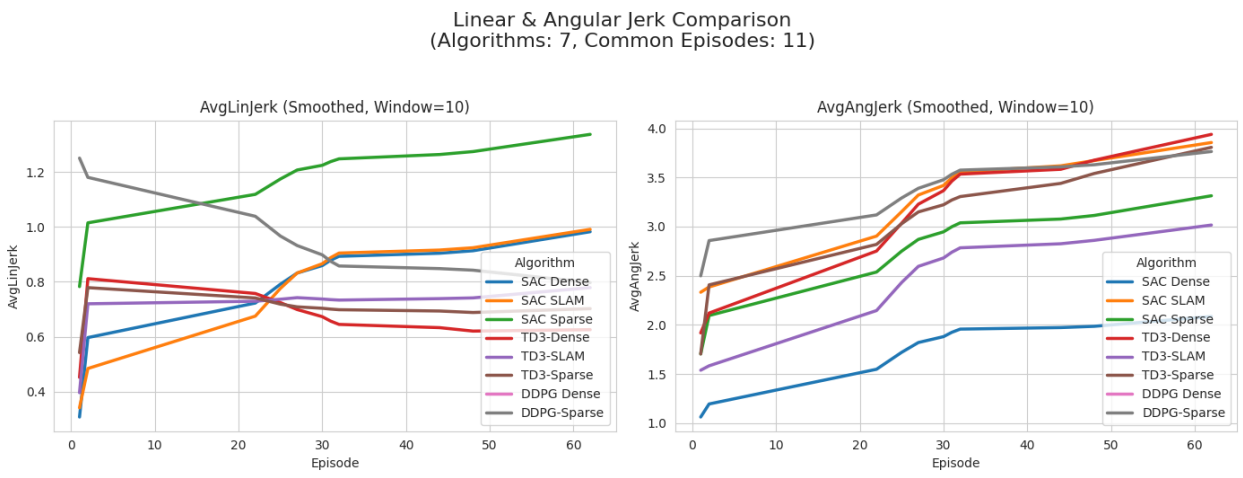


Figure 14: Comparison of linear and angular jerks in cluttered environments.

dense remains the best algorithm for minimizing both linear and angular jerks, ensuring smoother and more controlled motion compared to the others.

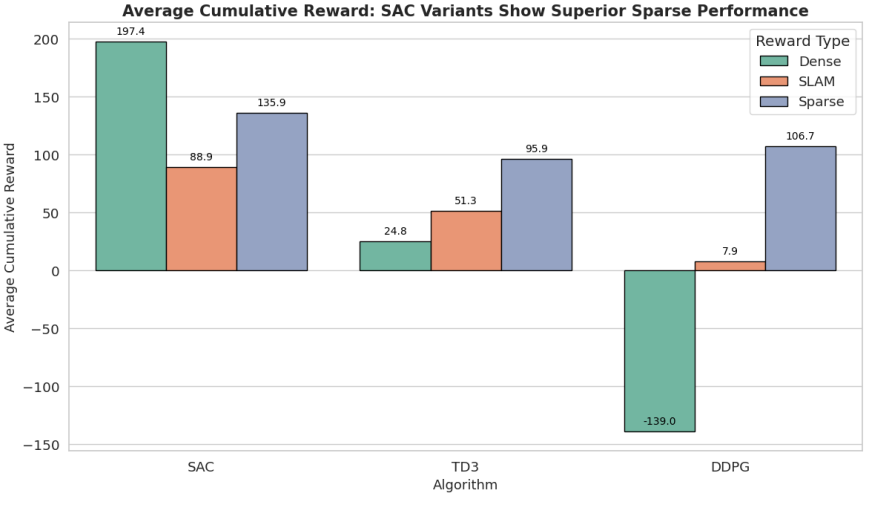


Figure 15: Cumulative reward function of DRL algorithms in cluttered environments.

The graph results in Figure 15 illustrate the average cumulative reward for different DRL algorithms. The results demonstrate that SAC with the proposed dense reward function outperforms the other, achieving the highest average cumulative reward of 197.4. The dense rewards function with TD3 and DDPG also produce positive rewards with scoring, with TD3.9 and DDPG 106.7, respectively. In contrast, sparse reward exhibits better performance for SAC and TD3, by achieving 95.9 and 51.3 rewards, respectively. The sparse reward leads to a negative reward for DDPG at -139.0. SLAM reward performs better, with SAC achieving an 88.9 reward, TD3 scoring 24.8, and DDPG attaining 7.9. Overall, SAC with the proposed dense reward function significantly performed better.

5.4. Statistical Evaluation

The statistical analysis is conducted to evaluate the performance of each algorithm, enabling a more precise comparison across several key metrics, including average angular jerk, linear jerk, path length, number of steps, and time. By computing the median, mean, and interquartile range (IQR) for each metric, we were able to evaluate the consistency, efficiency, and robustness of the algorithms within both sparse and cluttered environments.

Table 3
Mean Values of All Metrics for Each Algorithm in **Sparse** environment (lowest values highlighted)

Algorithm	AvgAngJerk	AvgLinJerk	PathLength	Steps	Time
SAC-Dense	**2.485**	1.238	2.099	26.000	3.108
SAC-SLAM	3.794	1.248	**1.968**	33.000	3.942
SAC-Sparse	3.315	1.419	2.268	34.000	3.840
TD3-Dense	4.159	**0.634**	2.227	**25.000**	**3.053**
TD3-SLAM	**2.484**	1.079	**1.968**	33.500	3.897
TD3-Sparse	2.545	1.179	2.048	30.500	3.750
DDPG-Dense	3.438	1.070	3.787	56.833	6.797
DDPG-SLAM	4.717	1.000	2.636	63.500	7.560
DDPG-Sparse	3.919	0.855	2.447	41.000	4.517

Table 4
Median Values of All Metrics for Each Algorithm in **Sparse** environment (lowest values highlighted)

Algorithm	AvgAngJerk	AvgLinJerk	PathLength	Steps	Time
SAC-Dense	**2.485**	1.238	2.099	26.000	3.108
SAC-SLAM	3.794	1.248	**1.968**	33.000	3.942
SAC-Sparse	3.315	1.419	2.268	34.000	3.840
TD3-Dense	4.159	**0.634**	2.227	**25.000**	**3.053**
TD3-SLAM	**2.484**	1.079	**1.968**	33.500	3.897
TD3-Sparse	2.545	1.179	2.048	30.500	3.750
DDPG-Dense	3.438	1.070	3.787	56.833	6.797
DDPG-SLAM	4.717	1.000	2.636	63.500	7.560
DDPG-Sparse	3.919	0.855	2.447	41.000	4.517

5.4.1. Statistical Analysis in sparse environment

The Table 3 presents the mean value of all various performances for all algorithms. SAC-Dense performed significantly better across all algorithms and achieved the lowest mean value for the average angular jerk of 2.583, indicating the smoothest angular motion. SAC-Dense also achieved the lowest mean value of 0.634 for the average linear jerk, reinforcing its ability to provide smooth and efficient motion with minimal linear jerk. SAC-Dense and SAC-SLAM both achieved the lowest value for the path length of 2.099 and 2.048, respectively. TD3-Dense achieved the lowest minimum value of 25.0 for the step size. Lastly, TD3-Dense has the lowest mean value of 3.05 for time, indicating better efficiency and generalization in unseen sparse environments.

The mean Table 3 and median Table 4 values are all the same due to the absence of significant outliers. SAC-Dense and TD3-Dense performed best, with SAC-Dense achieving the lowest angular and linear jerks, and TD3-Dense requiring the fewest steps and least time. Overall, SAC-Dense and TD3-Dense showed superior performance in key metrics, with the lowest mean across the board.

The Table 5 presents the IQR values of all DRL algorithms. The interquartille Range (IQR) is applied to measure the spread of the middle 50% of the data, to identify variability while excluding outliers. The Table 5 shows DDPG-SLAM achieved the lowest IQR for the average angular jerk with 0,117, indicating the smoothest motion. SAC-Dense had the lowest IQR for average linear jerk at 0.046, demonstrating its efficient performance. DDPG-Sparse performed best in path length with 1.417, steps with 12.00, and the time with 1.424, showcasing its overall efficiency is completing tasks with minimal path, fewer steps, and faster completion.

The Table 6 t-test shows that SAC demonstrates significant differences between Dense-sparse and Dense-Slam reward functions, while sparse-Slam is not statistically different because both reward functions similarly guide exploration. TD3 exhibits a significant difference only between sparse-slam, whereas Dense-sparse and Dense-slam show no difference, suggesting that TD3 is less sensitive to reward shaping. DDPG reveals strong differences between Dense-slam and sparse-slam, but not between Dense-sparse, indicating both reward types provide comparable learning signals for this algorithm. At the algorithmic level, all pairwise comparisons includes DDPG-SAC, DDPG-TD3 and SAC-TD3, show highly significant differences, indicating that SAC performed the best overall among the algorithms.

Table 5
IQR Values of All Metrics for Each Algorithm in **Sparse** environment (lowest values highlighted)

Algorithm	AvgAngJerk	AvgLinJerk	PathLength	Steps	Time
SAC-Dense	0.410	0.046	1.647	21.500	2.605
SAC-SLAM	0.951	0.086	**1.417**	20.000	2.462
SAC-Sparse	0.416	0.155	1.561	**12.000**	**1.424**
TD3-Dense	2.175	0.119	1.699	19.750	2.288
TD3-SLAM	0.252	0.231	1.893	21.750	2.562
TD3-Sparse	0.275	0.167	1.763	18.500	2.222
DDPG-Dense	0.453	0.324	3.063	42.000	4.881
DDPG-SLAM	**0.117**	0.149	2.193	43.750	4.847
DDPG-Sparse	0.205	**0.101**	4.158	65.500	7.677

Table 6
Pairwise t-test results for different algorithms and reward type comparisons. Statistically significant differences ($p < 0.05$) are highlighted.

Category	Comparison	t-statistic	p-value	Result
Soft Actor-Critic (SAC)	**Dense vs Sparse**	**6.7502**	**3.46e-09**	**Yes**
	Dense vs SLAM	**5.3763**	**9.30e-07**	**Yes**
	Sparse vs SLAM	-1.0557	0.2931	No
Twin Delayed DDPG (TD3)	Dense vs Sparse	-0.6131	0.5420	No
	Dense vs SLAM	-0.9238	0.3591	No
	Sparse vs SLAM	**-7.2656**	**9.31e-11**	**Yes**
Deep Deterministic Policy Gradient (DDPG)	Dense vs Sparse	-0.6571	0.5124	No
	Dense vs SLAM	**4.2668**	**3.92e-05**	**Yes**
	Sparse vs SLAM	**5.6325**	**1.13e-07**	**Yes**
All Algorithms	**DDPG vs SAC**	**-10.8425**	**3.30e-22**	**Yes**
	DDPG vs TD3	**-6.9878**	**2.05e-11**	**Yes**
	SAC vs TD3	**6.1035**	**3.23e-09**	**Yes**

Table 7
Mean Values of All Metrics for Each Algorithm in **Cluttered** Environment (lowest values highlighted)

Algorithm	AvgAngJerk	AvgLinJerk	PathLength	Steps	Time
SAC-Dense	**1.496**	0.729	**1.073**	**22.000**	**2.409**
SAC-SLAM	2.975	**0.656**	1.103	25.333	2.720
SAC-Sparse	2.544	1.133	1.081	24.667	2.701
TD3-Dense	2.917	0.710	1.131	26.667	2.855
TD3-SLAM	2.112	0.735	1.088	26.333	2.839
TD3-Sparse	3.087	0.732	1.094	48.000	5.248
DDPG-Dense	2.663	1.990	1.102	33.333	3.647
DDPG-SLAM	3.162	0.757	1.966	65.667	7.564
DDPG-Sparse	3.180	1.050	4.172	54.333	6.506

5.4.2. Statistical Analysis in Cluttered Environment

In this section, the statistical analysis of all DRL algorithms is presented in a cluttered environment. As shown in Table 7, the SAC-Dense algorithm outperforms other algorithms in cluttered environments. It achieves the lowest mean values for average angular jerk of 1.496, the number of steps of 22.0, and the lowest time value of 2.409, indicating a smoother and more efficient motion. Although SAC-SLAM shows the lowest average linear jerk value of 0.656, SAC-Sparse performs slightly better in the path length of 1.081. The above results show that SAC-Dense and TD3-Dense have performed better, and TD3-Sparse and DDPG-sparse exhibit significantly higher mean values in step and time, demonstrating less efficient navigation in cluttered environments.

Table 8
Median Values of All Metrics for Each Algorithm in **Cluttered** Environment (lowest values highlighted)

Algorithm	AvgAngJerk	AvgLinJerk	PathLength	Steps	Time
SAC-Dense	**1.327**	0.886	**1.269**	**28.000**	**3.084**
SAC-SLAM	2.438	0.627	1.316	31.000	3.337
SAC-Sparse	2.485	1.247	1.275	34.000	3.743
TD3-Dense	2.322	**0.508**	1.327	34.000	3.642
TD3-SLAM	1.628	0.765	1.271	37.000	3.992
TD3-Sparse	3.110	0.640	1.313	32.000	3.472
DDPG-Dense	1.954	2.433	1.335	40.000	4.400
DDPG-SLAM	3.219	0.529	1.402	64.000	7.803
DDPG-Sparse	3.216	1.110	4.219	58.000	6.668

Table 9
IQR Values of All Metrics for Each Algorithm in **Cluttered** Environment (lowest values highlighted)

Algorithm	AvgAngJerk	AvgLinJerk	PathLength	Steps	Time
SAC-Dense	**0.519**	0.343	0.803	16.000	1.711
SAC-SLAM	0.912	0.329	0.773	16.500	1.767
SAC-Sparse	0.869	0.294	0.773	15.000	1.600
TD3-Dense	1.297	0.360	0.820	16.000	1.703
TD3-SLAM	0.817	0.324	0.794	17.000	1.789
TD3-Sparse	1.372	**0.237**	0.802	53.000	5.800
DDPG-Dense	1.386	1.213	**0.754**	23.000	2.390
DDPG-SLAM	0.634	0.379	1.971	**8.500**	**1.035**
DDPG-Sparse	0.663	0.230	1.757	11.500	1.378

As shown in Table 8, SAC-Dense achieves the lowest median values for key metrics, angular jerk value of 1.327, path length of 1.269, step size of 28.0, and time value of 3.084. These results highlight its smooth and efficient motion in cluttered environments. SAC-SLAM shows the lowest linear jerk value of 0.627, while TD3-Dense slightly improves it with a median of 0.508. Moreover, TD3-SLAM achieves the lowest angular jerk of 1.628, and TD3-Sparse shows a slightly lower step count of 32.0 compared to other algorithms. DDPG algorithms show higher median values across most metrics, indicating inefficient performance.

Table 9 shows that SAC-Sparse achieves the lowest IQR in the path length of 0.773, the step size of 15.00, and the time of 1.60, indicating a highly consistent performance. TD3-Sparse shows the lowest variation in linear jerk of 0.237, while DDPG-SLAM has the lowest angular jerk variability of 0.643. Overall, SAC-Sparse offers the most stable behaviours across key metrics in the cluttered environments.

In the cluttered environment reward function analysis, a pairwise t-test is applied to evaluate statistical differences across algorithms and different reward functions, and the results are presented in Table 10. The findings reveal that SAC shows a significant difference between Dense-SLAM and Dense-Sparse, while Sparse-SLAM does not differ statistically. TD3 demonstrates significant improvement for Dense-sparse and Dense-slam, but not for sparse-slam. Moreover, DDPG shows strong differences across all comparisons, indicating is sensitivity to different reward functions. By comparing all algorithms, the results indicate that SAC outperforms DDPG and TD3 with all reward functions.

SAC with the proposed dense reward function outperforms all other algorithms, achieving the highest success rate, the shortest path length and the fewer step size and the lowest traversal times. I also minimize linear and angular jerks, ensuring smoother navigation. The cumulative reward results further highlight that SAC achieves higher rewards. Conversely, TD3 performs moderately, while DDPG shows poor performance especially with sparse rewards. Overall, SAC with a dense reward function is the most efficient and effective in cluttered environments.

Table 10
Pairwise t-test results for different algorithms and reward type comparisons. Statistically significant differences ($p < 0.05$) are highlighted.

Category	Comparison	t-statistic	p-value	Result
Soft Actor-Critic (SAC)	**Dense vs Sparse**	**2.6752**	**0.0085**	**Yes**
	Dense vs SLAM	**4.3822**	**0.0000**	**Yes**
	Sparse vs SLAM	1.9666	0.0515	No
Twin Delayed DDPG (TD3)	**Dense vs Sparse**	**-3.6399**	**0.0004**	**Yes**
	Dense vs SLAM	**-2.2358**	**0.0272**	**Yes**
	Sparse vs SLAM	1.5043	0.1350	No
Deep Deterministic Policy Gradient (DDPG)	**Dense vs Sparse**	**-7.9727**	**0.0000**	**Yes**
	Dense vs SLAM	**-4.6495**	**0.0000**	**Yes**
	Sparse vs SLAM	**2.9138**	**0.0042**	**Yes**
All Algorithms	**DDPG vs SAC**	**-8.2163**	**0.0000**	**Yes**
	DDPG vs TD3	**-2.4165**	**0.0161**	**Yes**
	SAC vs TD3	**5.8708**	**0.0000**	**Yes**

6. Conclusion

In this paper, we introduced a novel dense reward function that utilizes active SLAM for improved environmental perception and efficient exploration in autonomous robot motion planning. Our experimental results demonstrate that SAC with the proposed dense reward function outperforms other algorithms such as TD3 and DDPG in both sparse and cluttered environments. SAC dense achieved the highest success rates, shortest path lengths, and minimal traversal time, proving its superior efficiency and adaptability to unseen environments. Additionally, SAC dense minimized both linear and angular jerks, ensuring smoother navigation. The comparative analysis showed that SAC, with the proposed reward function, achieved the best performance across multiple metrics. TD3 demonstrated moderate performance, whereas DDPG struggled, particularly when using sparse rewards. Overall, SAC dense proved to be the most effective algorithm for navigating complex environments and generalization to new and unseen scenarios. While the dense reward function significantly improved the performance of SAC, it still depends on accurate SLAM-based map generation, which can be challenging in dynamic or noisy environments. Future work will focus on enhancing the robustness of the reward function in more dynamic scenarios and exploring real-world implementations to validate the proposed approach further.

7. CRediT authorship contribution statement

Khawaja Fahad Iqbal: Supervision, Methodology, Review. **Muhammad Tauseef Nasir:** Supervision, Review.

References

[1] Abdelwahed, M.F., Mohamed, A.E., Saleh, M.A., 2020. Solving the motion planning problem using learning experience through case-based reasoning and machine learning algorithms. Ain Shams Engineering Journal 11, 133–142.

[2] Arab, A., Yu, K., Yi, J., Song, D., 2016. Motion planning for aggressive autonomous vehicle maneuvers, in: 2016 IEEE International Conference on Automation Science and Engineering (CASE), IEEE. pp. 221–226.

[3] Arnold, R.D., Yamaguchi, H., Tanaka, T., 2018. Search and rescue with autonomous flying robots through behavior-based cooperative intelligence. Journal of International Humanitarian Action 3, 1–18.

[4] Bain, M., Sammut, C., 1995. A framework for behavioural cloning., in: Machine intelligence 15, pp. 103–129.

[5] Cimurs, R., Suh, I.H., Lee, J.H., 2021. Goal-driven autonomous exploration through deep reinforcement learning. IEEE Robotics and Automation Letters 7, 730–737.

[6] Da Mota, F.A., Rocha, M.X., Rodrigues, J.J., De Albuquerque, V.H.C., De Alexandria, A.R., 2018. Localization and navigation for autonomous mobile robots using petri nets in indoor environments. IEEE access 6, 31665–31676.

[7] Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271. URL: https://doi.org/10.1007/BF01386390, doi:10.1007/BF01386390.

[8] Fujimoto, S., Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods, in: International conference on machine learning, PMLR. pp. 1587–1596.

[9] Ganesan, S., Ramalingam, B., Mohan, R.E., 2024. A hybrid sampling-based rrt* path planning algorithm for autonomous mobile robot navigation. Expert Systems with Applications 258, 125206.

[10] González-Banos, H.H., Latombe, J.C., 2002. Navigation strategies for exploring indoor environments. The International Journal of Robotics Research 21, 829–848.

[11] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International conference on machine learning, Pmlr. pp. 1861–1870.

[12] Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics 4, 100–107.

[13] Holland, J.H., 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

[14] Jing, X.J., 2005. Behavior dynamics based motion planning of mobile robots in uncertain dynamic environments. Robotics and Autonomous Systems 53, 99–123. URL: https://www.sciencedirect.com/science/article/pii/S0921889005001211, doi:https://doi.org/10.1016/j.robot.2005.09.001.

[15] Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M., 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12, 566–580. doi:10.1109/70.508439.

[16] Khan, A., Zhang, F., 2017. Using recurrent neural networks (rnns) as planners for bio-inspired robotic motion, in: 2017 IEEE Conference on Control Technology and Applications (CCTA), IEEE. pp. 1025–1030.

[17] Khatib, O., 1986. The potential field approach and operational space formulation in robot control, in: Adaptive and Learning Systems: Theory and Applications. Springer, pp. 367–377.

[18] Konda, V.R., Tsitsiklis, J.N., 2003. Onactor-critic algorithms. SIAM journal on Control and Optimization 42, 1143–1166.

[19] Konur, S., Dixon, C., Fisher, M., 2012. Analysing robot swarm behaviour via probabilistic model checking. Robotics and Autonomous Systems 60, 199–213.

[20] LaValle, S., 1998. Rapidly-exploring random trees: A new tool for path planning. Research Report 9811 .

[21] Lei, H., Weng, P., Rojas, J., Guan, Y., 2022. Planning with q-values in sparse reward reinforcement learning, in: International Conference on Intelligent Robotics and Applications, Springer. pp. 603–614.

[22] Li, B., Huang, Z., Chen, T.W., Dai, T., Zang, Y., Xie, W., Tian, B., Cai, K., 2022. Msn: Mapless short-range navigation based on time critical deep reinforcement learning. IEEE Transactions on Intelligent Transportation Systems 24, 8628–8637.

[23] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 .

[24] Mandlekar, A., Garrett, C.R., Xu, D., Fox, D., 2023. Human-in-the-loop task and motion planning for imitation learning, in: Conference on Robot Learning, PMLR. pp. 3030–3060.

[25] Martínez-Barberá, H., Herrero-Pérez, D., 2010. Autonomous navigation of an automated guided vehicle in industrial environments. Robotics and Computer-Integrated Manufacturing 26, 296–311.

[26] Miranda, V.R., Neto, A.A., Freitas, G.M., Mozelli, L.A., 2023. Generalization in deep reinforcement learning for robotic navigation by reward shaping. IEEE Transactions on Industrial Electronics 71, 6013–6020.

[27] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. nature 518, 529–533.

[28] Pairet, È., Hernández, J.D., Carreras, M., Petillot, Y., Lahijanian, M., 2021. Online mapping and motion planning under uncertainty for safe navigation in unknown environments. IEEE Transactions on Automation Science and Engineering 19, 3356–3378.

[29] Pan, H., Luo, M., Wang, J., Huang, T., Sun, W., 2024. A safe motion planning and reliable control framework for autonomous vehicles. IEEE transactions on intelligent vehicles 9, 4780–4793.

[30] Ross, S., Gordon, G., Bagnell, D., 2011. A reduction of imitation learning and structured prediction to no-regret online learning, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings. pp. 627–635.

[31] Selekwa, M.F., Dunlap, D.D., Shi, D., Collins Jr, E.G., 2008. Robot navigation in very cluttered environments by preference-based fuzzy behaviors. Robotics and Autonomous Systems 56, 231–246.

[32] Selvarajan, S., Manoharan, H., Shankar, A., 2024. Sl-ri: Integration of supervised learning in robots for industry 5.0 automated application monitoring. Measurement: Sensors 31, 100972.

[33] Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI conference on artificial intelligence.

[34] Vukolov, A., 2024. D-star-based optimized trajectory planner for mobile robots operating in dense environments, in: European Conference on Mechanism Science, Springer. pp. 294–301.

[35] Wang, H., Lu, B., Li, J., Liu, T., Xing, Y., Lv, C., Cao, D., Li, J., Zhang, J., Hashemi, E., 2021. Risk assessment and mitigation in local path planning for autonomous vehicles with lstm based predictive model. IEEE Transactions on Automation Science and Engineering 19, 2738–2749.

[36] Wang, H., Zhang, L., Kong, Q., Zhu, W., Zheng, J., Zhuang, L., Xu, X., 2022. Motion planning in complex urban environments: An industrial application on autonomous last-mile delivery vehicles. Journal of Field Robotics 39, 1258–1285.

[37] Wang, Y., He, H., Tan, X., 2020. Truly proximal policy optimization, in: Uncertainty in artificial intelligence, PMLR. pp. 113–122.

[38] Watkins, C.J., Dayan, P., 1992. Q-learning. Machine learning 8, 279–292.

[39] Wu, X., Li, R., He, Z., Yu, T., Cheng, C., 2023. A value-based deep reinforcement learning model with human expertise in optimal treatment of sepsis. NPJ Digital Medicine 6, 15.

[40] Xu, W., Wang, Q., Dolan, J.M., 2021. Autonomous vehicle motion planning via recurrent spline optimization, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 7730–7736.

[41] Ye, L., Chen, J., Zhou, Y., 2022. Real-time path planning for robot using op-prm in complex dynamic environment. Frontiers in Neurorobotics 16, 910859.

[42] Ying, K.C., Pourhejazy, P., Cheng, C.Y., Cai, Z.Y., 2021. Deep learning-based optimization for motion planning of dual-arm assembly robots. Computers & Industrial Engineering 160, 107603.

[43] Zhang, L., Zhang, Y., Li, Y., 2020. Mobile robot path planning based on improved localized particle swarm optimization. IEEE Sensors Journal 21, 6962–6972.

[44] Zhao, Y., Wang, Y., Zhang, J., Liu, X., Li, Y., Guo, S., Yang, X., Hong, S., 2022. Surgical gan: Towards real-time path planning for passive flexible tools in endovascular surgeries. Neurocomputing 500, 567–580.