Robot Motion planning using Deep Reinforcement Learning

By

Zahra Khan

(Registration No: 00000364958)

School of Mechanical and Manufacturing Engineering

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2025)

Robot Motion Planning using Deep Reinforcement Learning

By

Zahra Khan

(Registration No: 00000364958)

A thesis submitted to the National University of Sciences and Technology, Islamabad,

in partial fulfillment of the requirements for the degree of

Master of Science in

Robotics & AI

Supervisor: Dr. Fahad Iqbal

School of Mechanical and Manufacturing Engineering

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2025)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Robot Motion Planning Using Deep Reinforcement Learning" written by Zahra Khan, (Registration No 00000364958), of SMME has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: 

Name of Advisor: Dr. Khawaja Fahad Iqbal

Date: 19-Aug-2025

HoD/Associate Dean: 

Date: 19-Aug-2025

Signature (Dean/Principal): 

Date: 19-Aug-2025

FORM TH-4

National University of Sciences & Technology

MASTER THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: (Student Name & Reg. #) Zahra Khan [00000364958]

Titled: Robot Motion Planning Using Deep Reinforcement Learning

be accepted in partial fulfillment of the requirements for the award of MS in Robotics & Intelligent Machine Engineering degree.

Examination Committee Members

Committee Member	Role	Signature	Sign Date
Khawaja Fahad Iqbal	Advisor		27-Aug-2025 2:28 PM
Yasar Ayaz	Committee Member		27-Aug-2025 5:04 PM
Muhammad Tauseef Nasir	Committee Member		27-Aug-2025 7:28 PM

Kunwar Khan
HoD /

28-August-2025
Date

COUNTERSINGED

28-August-2025
Date

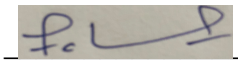
Javaid Iqbal
Principal

THIS FORM IS DIGITALLY SIGNED

Approval

It is certified that the contents and form of the thesis entitled "Robot Motion Planning Using Deep Reinforcement Learning" submitted by Zahra Khan have been found satisfactory for the requirement of the degree

Advisor : Dr. Khawaja Fahad Iqbal

Signature: 

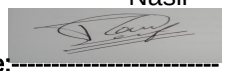
Date: 19-Aug-2025

Committee Member 1:Dr Yasar Ayaz

Signature: 

Date:21-Aug-2025

Committee Member 2:Dr Muhammad Tauseef Nasir

Signature: 

Date:19-Aug-2025

AUTHOR'S DECLARATION

I hereby declare that this submission titled "Robot Motion Planning Using Deep Reinforcement Learning" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SMME or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SMME or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name:Zahra Khan

Student Signature: Zahra Khan

Date: 19-Aug-2025

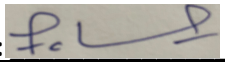
Certificate for Plagiarism

It is certified that PhD/M.Phil/MS Thesis Titled "Robot Motion Planning Using Deep Reinforcement Learning" by Zahra Khan has been examined by us. We undertake the follows:

a. Thesis has significant new work/knowledge as compared already published or are under consideration to be published elsewhere. No sentence, equation, diagram, table, paragraph or section has been copied verbatim from previous work unless it is placed under quotation marks and duly referenced.
b. The work presented is original and own work of the author (i.e. there is no plagiarism). No ideas, processes, results or words of others have been presented as Author own work.
c. There is no fabrication of data or results which have been compiled/analyzed.
d. There is no falsification by manipulating research materials, equipment or processes, or changing or omitting data or results such that the research is not accurately represented in the research record.
e. The thesis has been checked using TURNITIN (copy of originality report attached) and found within limits as per HEC plagiarism Policy and instructions issued from time to time.

Name & Signature of Supervisor

Dr. Khawaja Fahad Iqbal

Signature : 

DEDICATION

My parents, who have provided me with constant support and encouragement during this journey, are the ones to whom I dedicate my thesis. I owe everything to my mother for her love and knowledge and to my father for his fortitude and resiliency. To my older sister, whose unwavering encouragement and faith in me has given me courage. Your inspiring remarks have always motivated me to pursue greatness.

I also thank my mentors for their support and confidence in me, especially Dr. Fahad, to whom I dedicate this effort. Even in the most challenging times, your support has served as a source of inspiration. Finally, I dedicate this thesis to everyone who has encouraged me to follow my passion for robotics and artificial intelligence, believed in me, and supported my aspirations.

ACKNOWLEDGEMENTS

I would like to sincerely thank Dr. Fahad Iqbal, my supervisor, for his invaluable advice, unwavering support, and enlightening criticism during this research. This work has been greatly influenced by his knowledge and support. For providing the tools and scholarly setting required for my thesis to be completed successfully, I am also grateful to the faculty and staff of the School of Mechanical and Manufacturing Engineering (SMME). I would especially like to thank my RIME21 batch colleagues for their collaboration, encouragement, and technical discussions that helped me learn. Lastly, I want to express my sincere gratitude to my family for their unwavering support, patience, and encouragement during this journey.

TABLE OF CONTENTS

LIST OF TABLES

Page No.

LIST OF FIGURES

Page No.

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

DRL	Deep Reinforcement Learning
SAC	Soft Actor Critic
TD3	Twin Delayed Deep Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
RRT	Rapidly Exploring Random Tree
PRM	Probabilistic Roadmap
CNN	Convolution Neural Network
PSO	Particle Swarm Optimization
ML	Machine Learning
DL	Deep Learning

ABSTRACT

Effective motion planning is needed for autonomous robots in challenging environments. Sampling-based algorithms sample random points in high-dimensional spaces but struggle to work well in complex environments due to slow convergence and inefficiencies. Improvements in Deep Reinforcement Learning (DRL) overcome this strategy through learning optimal policies from acting in the environment, reducing reliance on environmental data and faster rates of convergence. But DRL resorts to sparse reward functions to produce suboptimal paths and poor exploration. To advance beyond such shortcomings, we propose an active SLAM-sourced information reward function. SLAM-weighted reward enhances navigation efficiency with richer environment perception and robustness in unexplored areas. It comprises distance-to-target and smoothness terms over trajectory that encourage reduced distances, reduced oscillation, and more stable robot performance. We use the Soft Actor-Critic (SAC) algorithm in our reward function, because it is best to perform well in new environments. Comparison experiments using Twin Delayed Deep Deterministic Policy Gradient (TD3) and Deep Deterministic Policy Gradient (DDPG) in cluttered and sparse environments demonstrated the superior performance of SAC. In sparse settings, SAC was 100% successful, performing 9.4% better than TD3 and 14.1% better than DDPG, and using 35% fewer steps than TD3 and 63% fewer than DDPG. In chaotic settings, SAC was 87.5% successful, performing 40.6% better than TD3 and 71.9% better than DDPG. These performances attest to the unprecedented effectiveness of the SAC algorithm in the direction of autonomous robots.

Keywords: Deep Reinforcement Learning, Differenrial derive robot, Model Free, DDPG, TD3, SAC, ROS1, Gazeebo

CHAPTER 1: INTRODUCTION

Recent developments in robotics have resulted in significant advancements to autonomous navigation, especially for mobile robots operating in unstructured and cluttered environments. These environments pose challenges, including enabling robots to navigate efficiently through cluttered spaces with static and dynamic obstacles, narrow corridor spaces, and avoid getting trapped in local minima. Traditional motion planning approaches, such Dijkstra's algorithm, and A*, work well for pathfinding in static environments but frequently fails in dynamic environments. These approaches have limited adaptability to dynamic changes in the environment, which leads to generate suboptimal solutions and in worst case stuck in local minima and failed to reach the goal. Hence, highlighting the importance of adaptive measures in crowded, changing environments. Deep reinforcement learning (DRL) is the more promising one, which enables robots to capacity to learn and adapt to advanced surroundings via experience and losses limitation of the conventional algorithms.

The aim of this work is to develop strong motion planning algorithm to control in complicated environment. Main issues overcome are the weakness of path planning algorithms, which perform poorly in long-term optimization for real-time scenarios and susceptible to local minima. This research proposes a DRL-based method that utilizes information from LiDAR sensor and dense reward function. The overall goal is to enhance the safety and performance of autonomous systems in difficult, real-world conditions by allowing robots to travel more efficiently with fewer obstacles to overcome.

1.1 Background and Motivation

Motion planning is significant in robotics since by it, robots can move in the environment from a point of origin to a destination point while circumventing obstacles and maintaining smooth and effective movement. Motion planning has a wide range of applications from industrial automation[1] to personal assistance[2]. Robots employ motion planning in manufacturing to perform accurate movements for applications like

welding and assembly [3]. Medical assistive robots apply motion planning to navigate disabled patients around their home. In addition, motion planning is applied in autonomous cars, autonomous vehicles[4]and drones [5]to plot their path and prevent them from colliding with one another in order to make a safe and efficient trip. Motion planning methodologies can be broadly categorized into learning-based and classical. The classical approaches cover very broad sets of techniques, the most efficient of which are combinatorial [6], graph-based [7], sampling-based [8], artificial potential fields (APF) [9], and bio-inspired heuristics [10]. Learning-based methods, on the other hand, have chiefly drawn upon machine learning [11] techniques, such as learning by demonstration (LbD) [12], reinforcement learning (RL) [13], and deep learning-based motion planning methods.

Graph-based methods such as A* [14]and Dijkstra's [15], and sampling-based methods such as RRT and RRT* [16], suffer from various limitations that prevent them from operating in dynamic as well as complex environments. One of the core limitations is that they cannot adapt to dynamic changes in the environment since they are primarily geared for static environments. These methods build suboptimal paths and cannot proceed smoothly as environmental change. In addition, these algorithms have the potential to become trapped in thin corridors and restrict the quality of how they can plan good routes in congested worlds. They also have enormous computational complexity, particularly in the case of big or complicated worlds, which leads to longer planning time and has negative impacts on real-time response applications.

Classical approaches make use of heuristics in guiding their searches, and these heuristics may lead to inefficiency in case they are poorly defined. They also suffer from inadequacies in dealing with high-dimensional space and therefore have deteriorating performance when the problem space is higher-dimensional. Classical approaches lack flexibility since they work on rules fixed beforehand and do not have the capability to learn from experience or shift strategies during the process [17]. The last thing to consider is that typical algorithms tend to need complete foreknowledge of the world, which is typically unachievable in real-world contexts in which information can be incomplete or erroneous.

Learning from techniques promise potential for motion planning but with tremendous drawbacks. They will ask for enormous training sets, which are challenging and time-consuming to obtain, and are of poor generalization, leading to their poor performance when they are tested in new environments. Furthermore, the algorithms can overfit training conditions and decrease their general performance in varied environments. To tackle such issues, Deep Reinforcement Learning (DRL) bridges deep learning and reinforcement learning in a manner that makes agents learn to form optimal policies from sensory perception and enhance their capacity to adapt under uncertainty.

1.2 Deep reinforcement Learning

Deep Reinforcement Learning (DRL) is becoming more common in a large variety of applications since it can learn complicated behavior by trial and error in high-dimensional spaces with ease [18]. Autonomous robotics is its most common usage, where it is applied to tasks such as navigation, collision avoidance, and manipulation. DRL is used to learn driving through traffic, make decisions with sensory data, and avoid collisions. It has also found application in warehouse robots to enable robots to automatically perform operations such as picking, putting, and moving products. Additionally, DRL is applied in unmanned aerial vehicles (UAVs) for path planning [5] and in industrial automation for performing repetitive tasks with changing environments. Besides robotics, DRL finds application in gaming, healthcare [19] (for instance, robot-assisted surgery and automated diagnosis), finance, and energy management.

DRL is advantageous as it can be flexible, handle continuous and high-dimensional data, and learn to enhance performance from experience. Its most significant advantage is that programming explicitly for some tasks is not required as agents learn policies that are optimal when they are engaged with the environment. DRL's ability to learn from raw sensory input data like images or 2D LiDAR point cloud information makes it extremely flexible when facing vague or overly complex settings, which are difficult for standard algorithms. DRL is also more capable in another realm of handling sequential decision-making issues with immediate action affecting the future state. Policy gradient algorithms, which are a form of DRL algorithm, also enable agents to perform amazingly well on

discrete as well as continuous action spaces. This is a valuable skill for robotics tasks, like causing a differential drive robot to move. DRL agents in real-world robotics applications can improve their sensor noise handling and uncertainty in input data with trial-and-error learning, which is extremely useful considering the unavoidable existence of sensor noise.

Motion planning using Deep Reinforcement Learning (DRL) has numerous advantages in adaptability and responding to complex environments, Robot tends to move by sensing the environments. In interacting with the environment, the DRL agent makes actions that affect its environment and receives rewards or penalties depending on its performance as feedback. In robot navigation, an agent receive a reward for reaching the goal without running into obstacles, or receive a penalty for running into delaying obstacles or inefficient movement. This enables the agent to learn faster and adapt continuously dispatching its actions to maximize the total amount of rewards it will receive over time.

DRL agents, in contrast to machine learning (ML) and deep learning (DL) methods, generate and build their own experience by exploring the environment in a replay buffer, without requiring massive pre-existing data. The buffer saves past experiences for the agent to learn from and relive without the necessity of constant external data acquisition. This autonomous learned capability allows DRL to operate effectively in dynamic and unstructured environments and increasingly independent. This paper focuses on differential drive motion planning for robots. Deep Reinforcement Learning (DRL) in general has immense potential to empower robots to learn and accommodate independently in challenging and dynamic environments, holding tremendous potential for maximizing applications such as autonomous navigation and motion planning.

In short, Deep Reinforcement Learning (DRL) promises huge opportunities for robots to learn on their own and adjust in difficult, dynamic conditions. The approach has huge potential for maximizing real-world tasks such as autonomous navigation and motion planning in real-world environments.

1.3 Limitations of Deep reinforcement Learning

Reinforcement Learning (DRL) promises high robot navigation performances with high prospects but still possesses several challenges. Sample inefficiency, one of the largest challenges, is where an agent must heavily explore its environment to learn an optimal policy. This problem gets worse in simulations, which become more laborious as the world becomes more complicated [20]. Consequently, training model-free policies becomes more challenging in large or dynamic environments, especially when an enormous number of possible obstacles configurations exist in the environment. During real-world deployment, the agent's capacity to adapt to unseen environments is restricted by this lack of generalization.

To tackle the problem of generalization in DRL, proposed approaches include data augmentation and domain randomization. To address the generalization problem [21] et.all increase the data to include the wide range of scenarios in the training phase. Introducing randomness in simulations, domain randomization [22] helps agents prepare for real-world conditions by requiring them to adjust to various environments, thus improving their robustness. However, The implementation of these strategies necessitates extensive data variation and training iterations in order to minimize sample inefficiency. In addition, when implemented in the real world, the behavior encoded in a simulation may not meet expectations or could even result in failure.

New solution proposed [23] to increase the adaptability of the DRL algorithms to new and unseen environments, by introducing a dense reward function, new state representation and continuous action space. In the proposed study a dense reward function is added which using the map information during the training phase to enhance the robot’s ability to make informed decisions. Guiding the robot to explore and navigate in the environment and proceed directly to the nearby goal, ensuring it reaches the goal position efficiently without getting trapped in local minimums. Additionally, this reinforcement enhances the robot's ability to traverse unseen environments.

The dense reward function that has been suggested improves generalization and prevents getting stuck in local minima by promoting exploration in complex environments. However, there are still some limitations. The risk of overfitting the reward function to the environments encountered during training is a significant concern. This could reduce the ability of the robot to learn to adapt to new, novel environments with different obstacle configurations. If the robot is exposed to unexpected situations or diverse conditions, such overfitting could result in poor performance. Furthermore, in extremely dynamic or unstable environments where there are great variations in obstacle distribution and types, the reward shaping can become partially non-adaptive. although it keeps the robot from being trapped. For the robot's navigation strategy to be robust in real-world, dynamic environments, more adaptive reward function would be required. Below are the core limitations of robotic navigation using Deep Reinforcement Learning (DRL):

• **Sample Inefficiency:** It requires a huge number of training episodes for DRL

algorithms to learn efficient policies, thus making training laborious.

• **Challenges to Sim-to-Real Transfer:** Due to the "reality gap," simulation-trained models can perform very badly or even fail to function in the real environment.

Generalization Issues: Policies learned through DRL can behave in an unpredictable way in unseen and highly dynamic situations.

• **Instability or Jerky Movement:** Since DRL-trained robots perform suboptimal policies during exploration, the robots can be jerky or unstable in nearly all intricate situations.

• **Overfitting:** DRL models can be poor in new environments to limit their ability.

• **High computer Demand:** To fine-tune and train, DRL would demand much computer power.

The research proposed here tries to get over these limitations by improving generalization, preferring sim-to-real transfer, and enhancing the reward function as well as promoting steady movement and adaptive in unknown and changing environments.

1.4 Proposed Research

The goal of this work is to enhance the motion planning ability of the P3dX robot. It is a differential drive robot with a Velodyne sensor for environmental sensing. Leverage the high-density 3D point cloud data produced by the Velodyne sensor, the proposed method aims at enhancing the ability of the robot to navigate congested areas. Three popular deep reinforcement learning algorithms will be utilized in the research: Twin Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor-Critic (SAC), and Deep Deterministic Policy Gradient (DDPG). All three approaches will be contrasted based on their capacity for efficient exploration towards goals without running into obstacles, with a special emphasis on the impact of having more than one reward function on policy robustness and learning efficiency. The evaluation will consider the capacity of each algorithm to handle dynamic changes in the environment, thus overcoming the shortcomings of traditional navigation methods.

The research involves the utilization of a variety of various reward functions designed to promote a variety of navigation objectives such as obstacle avoidance, goal-distance minimization, and smoothness of motion while exploring. Algorithmic performance will be rigorously evaluated using a variety of significant performance metrics such as time to reach the goal, jerk (linear and rotational), path length, and success rate. Performance of all algorithms will be evaluated in three unseen conditions, e.g., local minima, narrow corridors, and sparse cases. Moreover, in the case of the dense reward function, the SAC algorithm performed very well as well, dealing with these difficult cases efficiently and achieving better performance than the other algorithms. Through the illustration of the significance of reward function design to improve overall performance, this in-depth study endeavors to provide valuable information regarding the effectiveness of various reinforcement learning methods to robot navigation.

1.5 Problem statement

1.5.1 Problem Statement

This research aims to develop a motion planning algorithm for safe and efficient navigation of a robot in cluttered environments with static and dynamic obstacles. Addressing limitations such as overfitting, sample inefficiency, the proposed approach will tackle sensor uncertainty and high-dimensional state spaces. By incorporating tailored reward functions, the algorithm will improve generalization, adaptability, and smooth locomotion in static and dynamic environments.

1.5.2 Research Objectives

Research goals of this research are given below

- Establish a real-time motion planning algorithm for differential drive robots navigating in dense environments with static and dynamic worlds.
- Come up with a reward function in high dimensional state space that promotes efficient collision-free motion and smooth motion
- Improve Deep Reinforcement Learning (DRL) models' generalization to new environments and reduce sampling inefficiency and risk of overfitting.
- Enhance the transfer learning between the sim and real environments by minimizing sensor uncertainty and learning to respond to changing, real-world scenarios.
- Compare SAC, TD3, and DDPG algorithms and observe how they perform in diverse scenarios and reward function combinations.

1.6 Summary

This chapter comprises background and use of motion planning algorithms, including classic solutions and their shortcomings in scenarios with many obstacles. Also, add their limitations for deep reinforcement learning solutions like sample inefficiency and generalizability. The solution suggested in this study is carefully named in this chapter to constrain the scope of this suggested work.

CHAPTER 2: LITERATURE REVIEW

This chapter provides a description of the most important advances in robotic motion planning, emphasizing the issues and solutions of navigation in dense and chaotic spaces. This chapter deals with the issue of the limitation of current methods and the increasing importance of deep reinforcement learning techniques, especially since it is dealing with flexibility and real-time decision-making. This chapter attempts to set the groundwork for understanding the limitation of present literature and how more sophisticated solutions are needed.

2.1 Motion planning

One of the most significant aspects of robotics is motion planning in which robots can travel through their environments by determining the most efficient path from the beginning point to the end point without colliding with any obstructions. This requires thorough knowledge of the robot's dynamics and its world properties. Motion planning is split into two components, Path planning are concerned with determining an efficient and stable route, and control concerns the fact that the robot can move according to the planned route with accuracy by making real-time adjustments in its movement. The problem in motion planning is attempting to find a balance between the accuracy and consistency of planned paths and the need for computational efficiency, especially in complex as well as dynamic environments. To address these challenges, several motion planning algorithms have been developed over years; each has unique benefits and addresses robotic navigation problems. Classical motion planning algorithms provide structured approach to determine a path that avoids collisions in known or static environment. To choose the best or most feasible path for the robot, these algorithms usually entail searching through a predefine space. They frequently struggle with dynamic environments, high-dimensional state spaces, and real-time execution demands, despite being proficient in simpler, well-structured scenarios. Some of the classical algorithms includes, graph-based algorithms, sampling based reactive based and optimization-based algorithms, listed in below sections.

2.2 Graph based algorithms

The environment is represented by graph-based algorithms as a network of interconnected nodes, where each node represents a potential robot location and edges provide the practical routes between these locations. These algorithms are frequently employed for global path planning in static scenarios because they explore paths in a structured manner. The A*, Dijkstra, and D* [24] algorithms are well-known in this field. Key graph-based motion planning algorithms are listed below:

2.2.1 A Star Algorithm

The A* algorithm is a graph search technique that combines a heuristic estimate of the cost from the current node to the target (h-cost) with the cost from the start node to the current node (g-cost). A* investigates the most promising routes first using this combined cost (f = g + h). Based on their f-cost, the algorithm investigates nearby nodes till it achieves its goal. In large or high-dimensional environments, A* [14]can be computationally costly, particularly when accurate pathfinding requires fine resolution. It uses a heuristic function that may not always be precise, which occasionally results in suboptimal routes. The algorithm struggles in dynamic environments and is better suited for static environments because it must recompute routes whenever surroundings change.

2.2.2 Dijkstra's Algorithm

A traditional graph search algorithm, Dijkstra's [15] approach expands the node with the lowest initial cost to explore every potential route. In contrast to A*, this method employs just actual distances (g-cost) to estimate the distance to the target instead of using heuristics. To determine the least costly route, Dijkstra's algorithm explores every node, starting with the initial node and continuing on outwards out. By making sure that every node is visited in ascending order of distance from the starting node to ensures the shortest path. In numerous scenarios, particularly in big or complex environments, Dijkstra's is slower than A* due to its absence of a heuristic. Similar to A*, it needs to be completely

re-explored if new obstacles are encountered, making it inefficient in dynamic or real-time environments.

2.2.3 D star Algorithm

An enhanced version of the A* algorithm, D* (Dynamic A*) was developed for scenarios in which the cost map changes over time. When new obstacles are met or the cost map is updated, it dynamically recalculates the paths. To contend with dynamic challenges, D* [24] adapts A* by effectively recalculating pathways without having to begin the search from scratch. It works well in scenarios where the map is either continuously changing or only partially known since it can instantly modify the cost of nodes as the environment changes. D* is more computationally intensive than A* or Dijkstra, but it can manage dynamic changes better. This is particularly true in large-scale systems that undergo consistent changes. Furthermore, it has trouble in situations that are extremely dynamic and undergo rapid and continuous change.

Graph-based algorithms like A*, Dijkstra, and D* work well in static environments but are computationally intensive and have trouble with dynamic, real-time path adjustments.

2.3 Sample based Algorithms

Sampling-based algorithms establish internal affiliation by randomly sampling the state space, they are widely used in motion planning to facilitate efficient navigation in challenging environments. Important sampling-based algorithms are covered in this part, such as the Probabilistic Roadmap Method (PRM), RRT*, and Rapidly exploring Random Tree (RRT).

2.3.1 Rapidly exploring Random Tree (RRT)

One of the other well-known sampling-based planners is the Rapidly exploring Random Tree (RRT) [25]. It randomly samples point in the environment, interpolating

between the sampled points to the nearest tree node, and gradually constructs a tree that spans the state space from an initial point. RRT can be applied to online motion planning, and it has been proven that the cost of the solution returned by it is asymptotically optimal. However, traditional RRT possesses some shortcomings which may influence its performance in complicated instances, for example, low rate of convergence and high computational requirement.

2.3.2 Sparse-RRT* *Algorithm*

The Sparse-RRT* [26] method was created to address the constraints of conventional RRT with fewer computations needed without sacrificing efficiency. This form enhances path quality and accelerates convergence to optimal solutions by choosing nodes and sampling to optimize the tree formation. It states that experimental data, Sparse-RRR* solves satisfactorily in a very wide range of motion planning cases.

2.3.3 RRT Algorithm*

A better RRT algorithm that reduces the discovered paths via exploration is referred to as RRT*. It adds additional nodes, and it re-composes the tree to optimize the path. As the number of samples rises, RRT* [27]guarantees that the solutions become asymptotically optimal. Despite these improvements, RRT* still has limitations, including higher computing costs than conventional RRT and the need for a large number of samples to successfully traverse complex environments.

2.3.4 Probabilistic Roadmap Method (PRM)

Another sampling-based technique is the Probabilistic Roadmap Method (PRM) [28] , which generates a roadmap of viable routes by randomly selecting points in the configuration space and joining them to create a graph. PRM makes it feasible to pre-process and store the roadmap for future path planning, it is particularly useful in multi-query environments. PRM does have a few limitations nevertheless, such as the inability to adapt to changing conditions, where changing obstacles might render the roadmap ineffective, and the potential for large computing costs during the roadmap-building step.

2.4 Optimization-Based Algorithms

Optimization-based algorithms aims to find the optimal route by formulating motion planning as an optimization problem. These algorithms use mathematical techniques to reduce a cost function while satisfying constraints such as dynamic feasibility and obstacle avoidance. High-quality routes are typically produced by optimization-based methods.

2.4.1 Genetic Algorithm (GA)

Inspired by natural selection, genetic algorithms (GA) optimise motion planning [29] by gradually developing a population of potential solutions over many generations. In motion planning, GA starts with a starting population of paths that are assessed using a fitness function that represents each path's quality (e.g., safety, length). The algorithm generates new generations of routes that are increasingly optimised towards the goal by using operations like as crossover, mutation, and selection. --While GAs are flexible and can effectively search complicated solution spaces, it will most of the time take a lot of evaluations to be able to align to the best solution, which rises with more computation times. In addition, GAs could be plagued by local optima, which would eliminate their capacity to determine the best solution in scenarios with large constraints.

2.4.2 Particle Swarm Optimization (PSO)

A population-based optimization method known as particle swarm optimisation (PSO) [30] simulates the social schooling of fish and flocking of birds phenomenon. Characterized as a "particle," each potential solution updates its position in solution space relative to its own and neighbouring experience. PSO optimizes motion planning trajectories by finding the minimum cost function subject to obstacle avoidance and dynamic constraints. account. The cooperative behaviour of PSO allows it to converge to high-quality solutions at high speeds. PSO struggles to scale for high-dimensional problems and its performance relies heavily on its parameter settings, which may lead to premature convergence on poor routes.

2.4.3 Recurrent Spline Optimization (RSO)

Recurrent Spline Optimisation (RSO) is a method that iteratively refines motion trajectories using optimisation techniques after expressing them with splines. Flexible trajectory shaping is made possible by RSO [31], which can adapt to a variety of constraints and dynamic environments. In general, an objective function pertaining to time, energy efficiency, or path smoothness is minimised throughout the optimisation process. Although RSO may create flexible and smooth paths, but the intricacy of the spline representations and the optimisation procedure affect how computationally efficient it is. Furthermore, maintaining real-time performance might be difficult, especially in environments that are dynamic or extremely crowded and may require quick trajectory adjustments.

2.5 Reactive-Based Algorithms

Real-time motion planning is frequently accomplished by reactive-based algorithms, like the Artificial Potential Field (APF) [32], which produce immediate responses to the robot's environment. The APF algorithm guides the robot towards its target while preventing collisions by representing the robot's goal as an attractive force and obstacles as repulsive forces. Although APF works well in simple, real-time scenarios, it frequently has problems with dynamic environments and gets trapped in local minima, where the robot gets caught between opposing forces. Potential-field-guided learning is one variant that has been explored for more complicated scenarios, including warehouse navigation, to overcome these limits.

2.6 Learning Based Algorithms

Robots may adapt and learn from their interactions with dynamic surroundings because of learning-based algorithms, which are increasing in importance in the field of motion planning. These approaches fall into three general categories: deep learning (DL), machine learning (ML), Deep Imitation Learning (DIL), and deep reinforcement learning (DRL). Each has special benefits for resolving challenging robotics challenges.

2.6.1 Machine Learning (ML)

Motion planning has been employed alongside with machine learning techniques to identify obstacles, predict the best routes, and adapt to changes in the environment. It is very helpful for classification and regression tasks like obstacle detection and trajectory prediction to use supervised learning approaches, in which the model is trained on labelled data. Understanding the structure of various surroundings and clustering them together are made easier by unsupervised learning, which works with unstructured data. However, when it comes to high-dimensional areas or intricate decision-making processes, such real-time action planning, conventional machine learning algorithms are frequently constrained.

2.6.2 Deep Learning (ML)

A form of machine learning called "deep learning" uses multi-layered neural networks, or "deep networks," to extract complex features from huge amounts of high-dimensional data. DL algorithms can process sensor inputs like LiDAR, cameras, or depth sensors to comprehend an environment in real time for robotic motion planning. Complex tasks including obstacle identification, segmentation, and localisation can be handled using these techniques. For instance, vision-based systems frequently use Convolutional Neural Networks (CNNs) to comprehend their environment and make informed navigation decisions. One significant feature of deep learning is its ability to generalize successfully to different situations after being trained on diverse datasets [33]. However, DL models can be computationally expensive, needing significant amounts of training data and processing power. In addition, deep learning models are often not interpretable, which makes it difficult to comprehend the reasoning behind a model's decisions—a crucial aspect of safety-sensitive applications like autonomous navigation.

2.6.3 Deep Imitation Learning (DIL)

Another effective learning-from-experience-based motion planning technique is Deep Imitation Learning (DIL) [34], in which the robot is trained to mimic expert demonstrations. DIL acquires a model through observation of optimal behaviour given by human experts or other autonomous systems rather than DRL's trial-and-error method of learning. It is particularly useful for actions when the environment is too intricate for

exploration-based learning. By replicating expert trajectories, DIL may be employed in motion planning to teach robots how to drive through crowded circumstances. DIL models have the limitation that they are not necessarily good at generalization to circumstances quite different from the training circumstance because they are typically limited by the quality and number of expert demonstrations. It will be difficult for the robot to move around or make the correct decisions if it is placed in situations not reflected in the demo data.

2.6.4 Deep Reinforcement Learning (DRL)

The power of deep learning and reinforcement learning is combined in deep reinforcement learning (DRL), which uses either punishment or reward for an agent's behaviours to train it to make sequential decisions in each environment. DRL has become a potent tool in motion planning, allowing robots to find optimal routes and collision avoidance techniques autonomously without the need for pre-programmed instructions. Deep reinforcement learning has two major categories includes model free and model-based algorithms; this study focusses on model free learning techniques.

Some of the model free model includes Q learning [35], Robotic motion planning has advanced significantly because of Q-learning and Deep Q-Network (DQN) [36], its deep learning. Q-learning helps the agent choose actions that maximise its cumulative reward by storing the values of state-action pairings in a Q-table. However, the growth in the size of the Q-table makes traditional Q-learning insufficient for high-dimensional, continuous environments that are frequently encountered in robotics, and it struggles to scale to vast state spaces. By approximating the Q-function using a neural network, the Deep Q-Network (DQN) extended Q-learning and enabled the agent to manage more intricate and sizable state spaces. DQN did well in games such as Atari and showed success in situations with discrete actions. When used for continuous control tasks, such those needed for robotic motion planning, DQN still has several drawbacks. Converting continuous actions into a discrete set might lead to poor performance, which is a significant difficulty because DQN functions in a discrete action space. Furthermore, DQN suffers from ineffective exploration strategies that frequently result in suboptimal policies in

complex environments, and it is prone to instability during training because of overestimations of Q-values.

To overcome the instability problems with policy gradient approaches, Trust Region Policy Optimisation (TRPO) [37] was developed. It employs a KL-divergence constraint to maintain policy updates within a trust region and prevent large, destabilizing movements. TRPO is hard and computationally costly to implement second-order optimisation, although it improved stability and performance. Proximal Policy Optimisation (PPO) [38] introduces a clipped objective function that makes this easier and avoids the computational cost of TRPO to facilitate more stable and adaptable updates. PPO is less sample-efficient than off-policy methods, though efficient in nature, easy in design, and high in performance when bundled together in a strange way. As PPO is an on-policy technique, it is less sample-efficient than off-policy methods because it requires new data for every update. This increases the data collection and computation cost, particularly when new data collection is expensive or inefficient. Furthermore, PPO can be not optimal under scenarios of needing to have precise control and regular action spaces.

Algorithms like the Deep Deterministic Policy Gradient (DDPG) [39], specifically designed for continuous action space, were introduced to complete the work left undone by DQN. DDPG combines elements of Q-learning and policy gradient methods. It utilizes two neural networks: a critic network to estimate the Q-value of state-action pairs and an actor network for generating continuous actions deterministically.

With DDPG, discretisation of actions was no longer required, a significant leap forward for continuous control problems. The high sensitivity to hyperparameter adjustment and instability during training because of overestimation bias in Q-value estimates are two of DDPG's inherent drawbacks. Furthermore, the deterministic character of DDPG may result in inadequate exploration, particularly in settings with complex, multi-modal reward systems or in situations where the robot needs to adjust to novel or changing circumstances.

The Twin Delayed Deep Deterministic Policy Gradient (TD3) [40] was developed to improve the performance and stability of policy learning in continuous control situations

in response to the drawbacks of DDPG. By using clipped double Q-learning, which keeps two critic networks while choosing the minimal Q-value between the two critics to reduce overestimation, Hence, TD3 solves the problem of overestimation bias. Target policy smoothing is yet another innovation brought by TD3 to the target action with an aim of preventing overfitting regarding local peaks in the Q-value function. Policy update delays, allowing for more stable learning by updating the actor network at a lower frequency than the critic networks, constitute the third major innovation. The innovations make TD3 more stable in continuous control tasks, which makes it a suitable option for robotic motion planning. Although TD3 reduces the overestimation bias and enhances stability, its deterministic policy can be limiting in cases where border exploration of the action space is needed.

Soft Actor-Critic (SAC) [41], leveraging stochastic policies for improved handling of noisy and dynamic environments, was proposed since deterministic policies in DDPG and TD3 are absent. Adding an entropy term to the reward function and not disrupting the exploration-exploitation trade off, SAC encourages exploration. For mitigating overestimation bias, it also employs clipped double Q-learning. For difficult challenges such as robotic motion planning in which the robot must move in cluttered and dynamic space and escape local minima, SAC is highly effective due to its capacity to keep exploring and learning in real-time.

2.7 Deep Reinforcement Learning and Motion Planning

Deep Reinforcement Learning (DRL) has received significant interest in robotic motion planning in recent years due to its capacity to deal with dynamic spaces high-dimensional state spaces, as well as hard decision-making issues. This part discusses early research studies and case studies that illustrate the successful application of DRL in robotic motion planning, paying attention to the methods utilized, the performance realized, and their significance in expanding the field.

Discrete action and Double Deep Q-Network (DDQN) algorithm were employed by Ruan et al. [42] to move through an environment from camera images alone without

colliding with obstacles. Discrete action, while performing well in some environments, can prove to be a performance bottleneck in a highly crowded environment, and its reward function does not consider local minimum issues. Chen et al. also employ a DDQN algorithm with discrete action. [43] for navigation. The method is based on distance rewards to hit targets and collision avoidance but includes an occupancy grid map as input, which increases runtime computing costs. Grando et al. [44] introduced a method for using the Deep Deterministic Policy Gradient (DDPG) algorithm to learn a policy that navigates a hybrid Unmanned Aerial Vehicle (UAV) to move toward a destination avoiding obstacles and present actions as well. The model utilizes the robot's states, LiDAR measurements, and target location distance to determine the best control actions for the UAV. The used reward function is too sparsely populated and would result in bad performance for complicated situations. Four-step raw depth images are input into a double SAC architecture in [45], where a secondary network handles obstacle avoidance and a primary network train the navigation policy.

A similar hierarchical method is used in [46], where a high-level policy guarantees safety while a low-level policy handles navigation towards the destination. Although hierarchical networks have drawbacks compared to single networks, including increased algorithm complexity, the need for additional training, a higher risk of overfitting, and more difficulty in tuning hyperparameters, they typically aid in enhancing learning efficiency and stability when the agent is faced with complex tasks. In the context of map exploration, [47] and [48] offer methods for utilising a trained policy navigation system to explore the space. The first one uses the TD3 algorithm and provides a reward function that solely looks at nonprogressive movement circumstances, collisions, and arrivals. The second one was trained using the DDPG, and a reward consisting of a safety clearance term, avoiding obstacles, and travelling in the intended direction was created. Local minima problems are common in map exploration assignments because the robot must go a long distance. The authors of [47] provide auxiliary algorithms to help the target destination avoid local minimum and boost the generalisation of the strategy, while [48]employs a safety clearance reward to stop the robot from being high attraction to the goal. The proposed reward function does not specifically address this problem, and the training techniques do not exhibit significant action exploration during agent learning.

[49] et.all address the local minima problem, by introducing a dense reward function, that incorporate map information during training, hence the agent makes more informed decisions and improved the navigation performance. The proposed solution suffers from generalization issues in unseen environments. While its solutions are quite acceptable in simpler settings, its performance degrades in more complex environments. Additionally, this method is not effective in cluttered and highly dynamic environments.

2.8 Summary

We present a thorough literature analysis on motion planning in this chapter, with an emphasis on its use in autonomous robotics. We examine conventional algorithms and talk about their drawbacks in dynamic settings, including sampling-based and graph-based approaches. The discussion then shifts to learning-based methods, specifically deep reinforcement learning (DRL), and looks at how DRL methods have been used to improve robotic navigation. The foundation for our suggested motion planning strategy is laid by identifying the developments and gaps in existing research through an analysis of numerous studies and their techniques.

CHAPTER 3: METHODOLOGY

This chapter describes the proposed methodology for autonomous navigation using deep reinforcement learning. Proposed approach integrates the robot operating system (ROS) Noetic and Gazebo simulator, where robot can learn to navigate autonomously while avoiding obstacles. Below in the block diagram of the proposed methodology, proposed method defines a state space that includes lidar data goal distance, heading angle, and previous actions. This state space helps the robot to assess the environment and make decisions accordingly and interact with the surrounding. Robot also gets reward and penalty based on the action it took it the environment. Hence the reward-based learning improves obstacle avoidance, effectively balancing exploration and exploitation, and efficient navigation towards its goal in dynamic and cluttered environment.

Proposed method uses Differential derive robot, and for the policy learning dense reward shaping is proposed, which avoids the robot to mitigate the local minima problem. A comparison is also made between Deep Deterministic Policy Gradient (DDPG), Twin

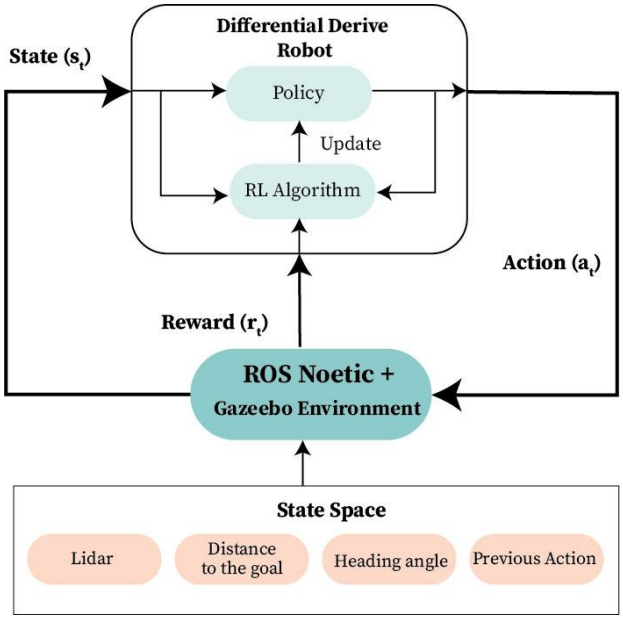


Figure 3.1 Block diagram of proposed methodology

Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor-Critic (SAC).

3.1 State Space and action space

In the proposed methodology, our robot is equipped with the velodyne sensor with a maximum measuring distance of 10 meters and laser reading is recording in 180-degree angular range in front on the robot. The laser data is divided into 21 groups and in each group minimum value of each group creates the state space. The second part of the state space is the goal state, goal state is represented as distance and heading angle. Euclidean distance between the robot current position and the goal position (x_{goal}, y_{goal}).

$$distance = \sqrt{\left(x_{goal} - x_{robot}\right)^2 + \left(y_{goal} - y_{robot}\right)^2} \tag{1}$$

The second dimension of the goal state is heading error, which is the error between robot heading and the directional vector points to the goal position. Heading angle given as

$$\Delta\varphi_t = \text{atan}\left((q^{<y>} - p_t{}^{<y>}), (q^{<x>} - p_t{}^{<x>})\right) - \psi_t \tag{2}$$

In the above equation ψ_t represents the heading angle of the robot, and heading angle is normalized such as:

$$\Delta\varphi_t = \begin{cases} \dfrac{\Delta\varphi_t - 2\pi}{\pi} & if\ \Delta\varphi_t > \pi \\ \dfrac{\Delta\varphi_t + 2\pi}{\pi} & if\ \Delta\varphi_t < -\pi \\ \dfrac{\Delta\varphi_t}{\pi} & otherwise \end{cases} \tag{3}$$

Once robot received the above-mentioned input, RL algorithm generates the action commands $a_t = \lceil v_t, w_t \rceil$ to interact the environment, for the linear velocity it generates the output in the range $v_t \in \lceil 0, 1 \rceil$ and angular velocity $w_t \in \left\lfloor -\frac{\pi}{2}, \frac{\pi}{2} \right\rfloor$

All above mention properties makes the state space, such as lidar, goal state and the previous action that robot took.

3.2 Policy Learning Algorithms

3.2.1 Deep Deterministic Policy Gradient Algorithm

DDPG is a model-free RL algorithm [50] [10], indicating it learns by interacting with the environment directly and doesn't require an environment model. It makes a substantial contribution in continuous action space and is based on actor critic architecture [16]. 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). A current and a target network are used in this approach 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 $(\mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, \mathrm{s}_{t+1}) \sim \mathrm{D}$ 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 added from the replay buffer for discounted target estimate, and the result is compared with the current Q network o compute the temporal difference $L(Q_\theta, D)$.

$$\mathrm{L}(\mathrm{Q}_\theta, \mathrm{D}) = \mathrm{E}_{(\mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, \mathrm{s}_{t+1})\sim \mathrm{D}} \left[\left(\mathrm{Q}_\theta(\mathrm{s}_t, \mathrm{a}_t) - \left(\mathrm{r} + \gamma(1-\mathrm{d})\mathrm{Q}_{\theta\mathrm{target}}\left(\mathrm{s}_t + 1, \pi_{\theta\mathrm{target}}(\mathrm{s}_{t+1})\right)\right)\right) 2 \right] (4)$$

where d is a binary indicator for episode termination and γ is the discount factor. the current policy π_θ generates actions in a continuous action space, maximizing the Q-value estimated by critic network Q_θ. The actor network aims to optimize the policy π_θ that maximizes the Q_θ.

$$L(\pi_\theta, D) = \theta_{max} E_{(\mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, \mathrm{s}_{t+1})\sim D}\left[\mathrm{Q}_\theta\left(\mathrm{s}_t, \pi_\theta(\mathrm{s}_t)\right)\right] \quad (5)$$

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.

3.2.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is the successor of the DDPG, TD3 reduces the overestimation biases problem of the DDPG. TD3 introduces the twin pair of critic networks and the target network. Minimum of both the twin networks contribute to compute the loss, but this leads to underestimation of the bias, the loss function given below:

$$\mathrm{Q}_{\theta\ target}\ \left(S_{t+1}, \pi_{\theta\ target}\ (S_{t+1})\right) = \min \begin{bmatrix} Q_{\theta\ target\ 1}\ \left(S_{t+1}, \pi_{\theta\ target}\ (S_{t+1})\right) \\ Q_{\theta\ target\ 2}\ \left(S_{t+1}, \pi_{\theta\ target}\ (S_{t+1})\right) \end{bmatrix} \quad (6)$$

$$L(\mathrm{Q}_{\theta}, D)\ =\ E_{(\mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, s_{t+1})\sim D}\ [(\mathrm{Q}_{\theta 1}\ (\mathrm{s}_t, \mathrm{a}_t)\ -\ (\mathrm{r}_t\ +\ \gamma(1-\mathrm{d})\ \mathrm{Q}_{\theta\ \mathrm{target}}\ (\mathrm{s}_t + 1, \pi_{\theta\ \mathrm{target}}\ (\mathrm{s}_t + 1))))^2\ +\ (\mathrm{Q}_{\theta 2}\ (\mathrm{s}_t, \mathrm{a}_t) - (\mathrm{r}_t\ +\ \gamma(1\ -\mathrm{d})\mathrm{Q}_{\theta\ \mathrm{target}}\ (\mathrm{s}_t + 1, \pi_{\theta\ \mathrm{target}}\ (\mathrm{s}_t + 1))))^2 \quad (7)$$

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. Policy robustness further improved by clipping the noise added in the action space, this ensures the noise regulated actions during 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.

3.2.3 Soft Actor Critic (SAC)

Soft Actor-Critic (SAC) combines exploration and learning efficiency in continuous action spaces by optimising 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 maximise the policy's entropy and expected return, which is represented by the objective function that follows:

$$J(\pi) = \sum_{t} E(s_t, a_t) \sim \rho\pi \left[r(s_t, a_t) + \alpha H\left(\pi(\cdot \mid s_t)\right) \right] \quad (8)$$

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

$$J(Q) = E_{(s_t, a_t, r_t, s_{t+1}) \sim D}[(Q(s_t, a_t) - (r_t + \gamma E_{a_{t+1}} \sim \pi[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} \mid s_{t+1})]))^2] \quad (9)$$

The policy update seeks to maximize the expected Q value and entropy.

$$J(\pi) = E_{st \sim D,\, a_t \sim \pi}\left[\alpha \log\left(\pi(a_t \mid s_t)\right) - Q(s_t, a_t)\right] \quad (10)$$

By directing SAC's policy towards a balanced exploration and reward-driven action selection, these equations strengthen the system's robustness in challenging circumstances.

3.3 Reward Function

Three different reward functions were created and tested in this study to evaluate their effectiveness as they're guiding the robot towards safe and effective navigation. Each reward function was designed to prioritize different motion planning aspects such as prioritizing goal reaching, path smoothness, obstacle avoidance. Different reward functions are listed below:

$$r(s_t, a_t) = \begin{cases} r_g & \text{if } D_t < \eta_D \\ r_c & if\ collision \\ v - |\omega| & otherwise \end{cases} \quad (11)$$

The reward $r(s_t, a_t)$ depends on the distance reward, if robot distance to the goal position at the current timestep D_t is less than a threshold η_D a position goal reward is applied. If the distance between the robot and the obstacle is less than a certain safe distance then the collision penalty a small negative reward is r_c is applied. A continuous reard function is also applied on the absence of above condition, an immediate reward is applied based on the current linear and angular velocities, this reward function emphasis to take less turn and take linear velocity as it will allow the robot to reach the goal position faster.

The second reward function is a modified version of the previous one, with an additional term introduced as follows:

$$r_{dense}(s_t, a_t) = \ r_{Gd} \tag{12}$$

where

$$r_{Gd} = \frac{G}{d} \tag{13}$$

In the above equation, G represents the increase in map information since the last measurement, The increased map information G is computed using data from the map occupancy grid created while the robot navigates through the environment. As the robot navigate in the environment, after each time step difference between the sum of all occupied cells (Oc) and free cells (Fc) in the grid in the current time step t and in the previous time step $t-1$ is calculated, given below.

$$G_t \ = \ (O_{ct} + F_{ct}) - \ (O_{ct} - 1 + F_{ct} - 1) \tag{14}$$

and d denotes the distance between the robot and the goal position, which, as a distance measure, cannot be zero in above equation. This new term in the reward function helps to mitigate the problem of local minima, by reinforcing the robot to explore new areas when the robot far away from the goal and gradually decreases this and emphasis robot to reach the goal position if its near the goal location. The reward function also avoids the robot from revisiting previously explored places which do not contribute any progress towards the goal position.

$$r_{dist} = \frac{\mathrm{d}_{t-1} - \mathrm{d}_t}{d_{total}} \tag{15}$$

In above equation d_t is the Euclidean distance in the current time step, d_{t-1} is the distance in the previous time step before the robot took a step d_{total} is the total distance at the start of the episode. This reward function gives the robot a positive reward function if robot progressing and taking steps toward the goal position, and a small penalty if it is going away.

$$r_a = v_t - \eta_a \times \text{abs}(\omega t) \quad (16)$$

Above is the action reward function, and it derives the UGV to approach the goal position as soon as possible by taking few numbers of steering commands.

3.4 Summary

This chapter presents the comprehensive methodology for the recommended approach, which includes the usage of model-free reinforcement learning algorithms, the specification of the state space. The reward functions aimed to regulate the robot's behavior are defined along with different algorithm. Together, these components provide the framework for effective motion planning in cluttered environment.

CHAPTER 4: EXPERIMENTAL RESULTS

The experimental results of the DRL algorithms, Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Soft Actor-Critic (SAC), are covered in this section. All algorithms are evaluated in unseen environments and strengths and weakness are discussed.

4.1 Evaluation Scenario

The experimental results from evaluating the proposed motion planning framework based on reinforcement learning are presented in this section. The system's performance is evaluated in a variety of simulated environments including spare and cluttered unseen environments. All deep reinforcement learning algorithms including SAC, TD3 and DDPG are evaluated in these two environments with different reward functions listed below

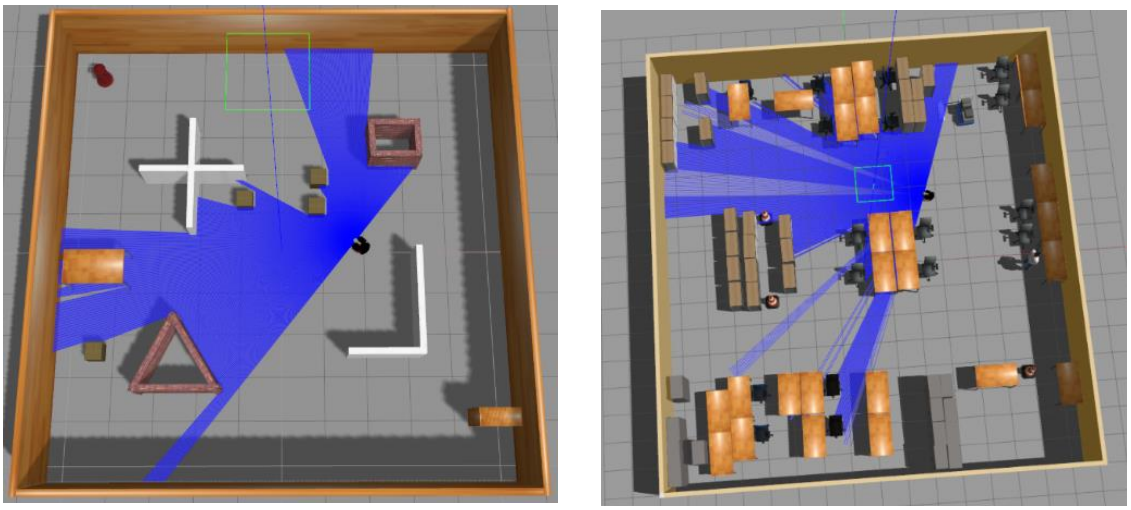
Table 4.1: List of Reward function used for comparison

Serial No	Reward Function	Reference
1	$r = \begin{cases} r_g & if\ D_t < \eta_D \\ r_c & if\ collision \\ v - \lvert\omega\rvert & Othervise \end{cases}$	Sparse Reward
2	$r_{dense(s,a)} = r_{Gd} + r_l + r_v$ $r_{Gd} = \frac{G}{d}$ $G_t = (O_{ct} + F_{ct}) - (O_{ct} - 1 + F_{ct} - 1)$	SLAM reward

3	$R = \begin{cases} 200.0 & if\ Target = True \\ -100.0 & if\ collision = True \\ r_{dist} + r_{smooth} + r_{vel} + r_{laser} + r_{map} & Otherwise \end{cases}$	Dense reward

4.2 Evaluation in Sparse Environment

All algorithms are trained on a same environment given below and evaluated in



(a) (b)

Figure 4.1 figure (a) is the training environment in gazebo, and (b) is the testing sparse environment with dimension (20*20) meters.

uns

in

algorithms evaluated in 64 same start and goal position and evaluated using the performance metric such as time to reach goal, path length, steps size, linear and angular jerk, and reward function in testing phase. Evaluation results of 64 independent episodes of all episodes are given below:

Table 4.2: Success rate comparison for different DRL algorithms in sparse environment

Algorithm	Successful Episode	Success Rate (%age)
SAC-Dense	64/64	100%
SAC-SLAM	63/64	98.44
SAC-Sparse	63/64	98.44%
TD3-SLAM	63/64	98.44
TD3-Sparse	63/64	98.44
TD3-Dense	58/64	90.62
DDPG-Dense	55/64	85.94
DDPG-SLAM	22/64	34.38%
DDPG-Sparse	53/6	82.81%

Above table shows that SAC outperformed in sparse environments by achieving 100 percentage success rates. Whereas, TD3-Sparse and TD3-Slam also generated competitive results but DDPG and TD3-Dense exhibit lower success rate. Other performance evaluation shown in figure. The results shows that SAC and TD3-dense has outperformed and achieved lowest steps to the goal and shortest path length and time. Whereas TD3-SLAM Performance on all evaluation metrics was moderate. DDPG and TD3-Sparse, on the other hand, showed slow convergence, requiring a lot more steps and longer paths to get to the destination. The time efficiency results demonstrate that SAC reached to the goal position earlier followed by TD3-Dense, TD3-SLAM, TD3-Sparse and then DDPG. These findings unequivocally demonstrate that the speed and general performance of navigation are greatly enhanced using dense reward structures.

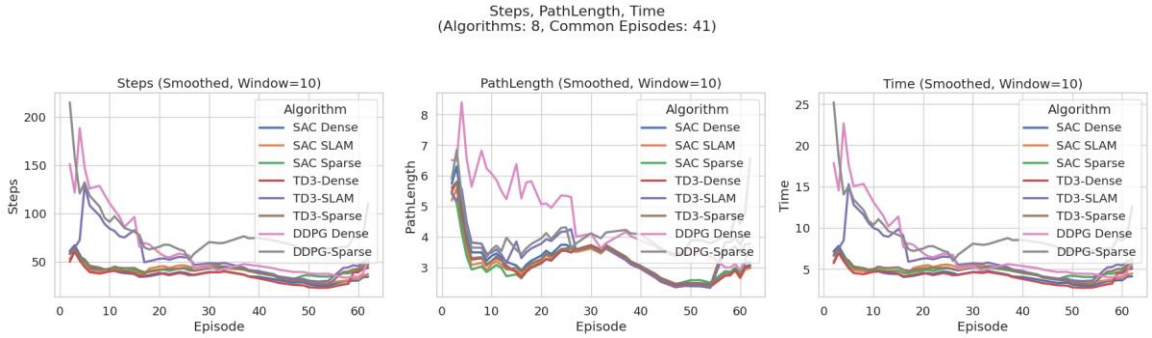


Figure 4.2 Comparison of SAC, TD3, and DDPG in cluttered environments based on number of steps, path length, and time.

The evaluation results across steps, path length, time, jerk, and reward metrics demonstrate that SAC with dense rewards consistently outperforms other methods in environments. SAC Dense achieves the shortest path lengths and the fewest steps across episodes by quickly reducing the number of steps per episode. In comparison to SAC SLAM, SAC Sparse, TD3 variants, and DDPG, it also records the lowest elapsed time to reach goals, indicating quicker and more effective navigation in sparse environment. On the other hand, TD3 variants demonstrate moderate rates of convergence, requiring longer paths and more steps to achieve goals than SAC variants, indicating lower efficiency in sparse environments. With high variance and higher step counts throughout the episodes, especially when rewards are sparse, DDPG variants exhibit the slowest performance, therefore indicating challenges with stable and effective navigation.

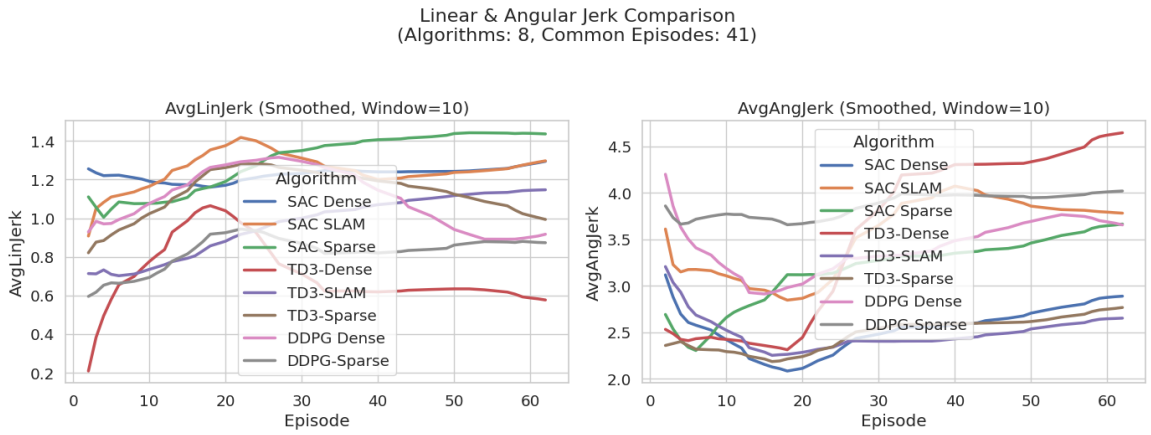


Figure 4.3 Analysis of linear and angular jerks for motion smoothness in sparse environments.

SAC Dense achieves the lowest linear and angular jerk in terms of trajectory smoothness, indicating more stable and smoother control while navigating. Although its jerk values are a little higher, SAC SLAM and SAC Sparse are still within reasonable bounds for steady motion. Variants of TD3 exhibit moderate jerk levels, which are considerably lower than DDPG but higher than SAC, indicating that TD3 can navigate sparse spaces with reasonably smooth trajectories. DDPG variants, on the other hand, show the highest linear and angular jerk values, especially when rewards are sparse. This is indicative of sudden changes in velocity and less stable control policies during testing and training.

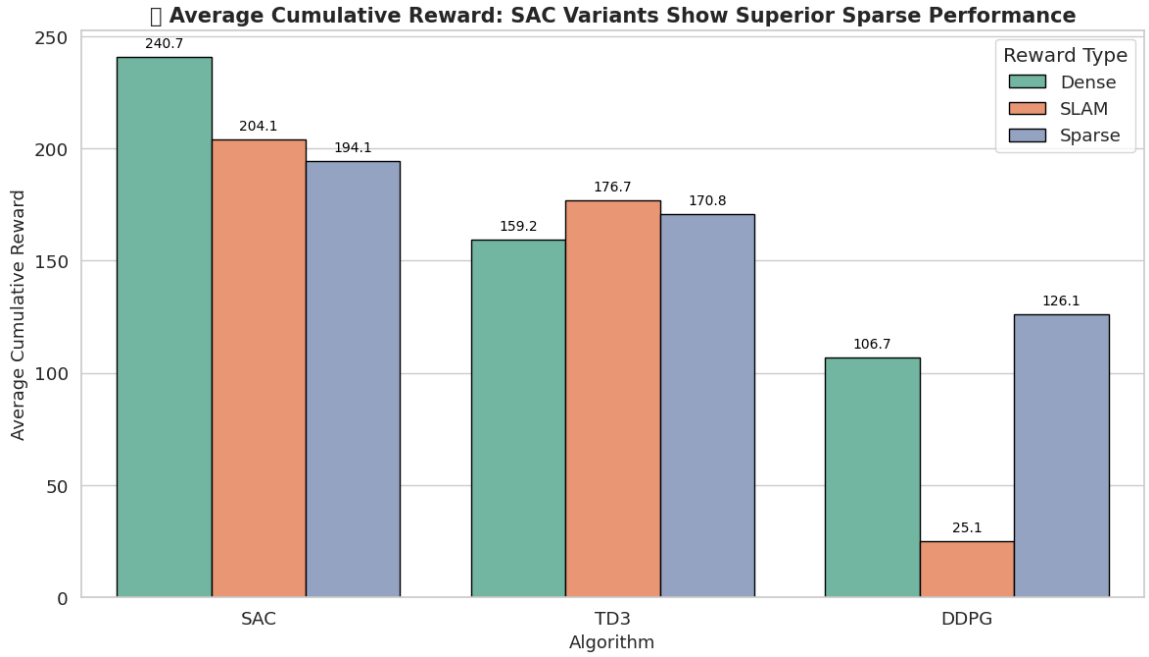


Figure 4.4 Average cumulative reward of 64 episodes in sparse environments

As evidence of SAC's superior learning and stability in sparse environment, SAC Dense achieves the highest average cumulative reward (240.7), followed by SAC SLAM (204.1) and SAC Sparse (194.1). With TD3 SLAM (176.7) marginally outperforming TD3 Sparse (170.8) and TD3 Dense (159.2), TD3 variants yield moderate cumulative rewards. This suggests that, although TD3 is capable of learning efficient navigation policies, it falls short of SAC in terms of performance, particularly when rewards are dense. Overall, DDPG variants perform the worst; DDPG Dense and SLAM exhibit much lower rewards (106.7 and 25.1, respectively), while DDPG Sparse attains a comparatively higher reward (126.1) but still performs worse than SAC and TD3. This demonstrates the stability and reward accumulation limitations of DDPG in environments that are sparse.

4.3 Evaluation in Cluttered Environment

SAC demonstrated strong reliability in complex scenarios, achieving the highest success rate of 87.5 percent in the cluttered environment shown in fig. A success rate of 71.88 percent was attained by TD3-Sparse, and 62.5 percent by TD3-SLAM. TD3-Dense achieved a moderate success rate of 46.88 percent. With a success rate of only 15.62 percent, DDPG demonstrated extremely poor performance. These findings demonstrate that SAC provides the best navigation performance in cluttered environments below table shows these results.

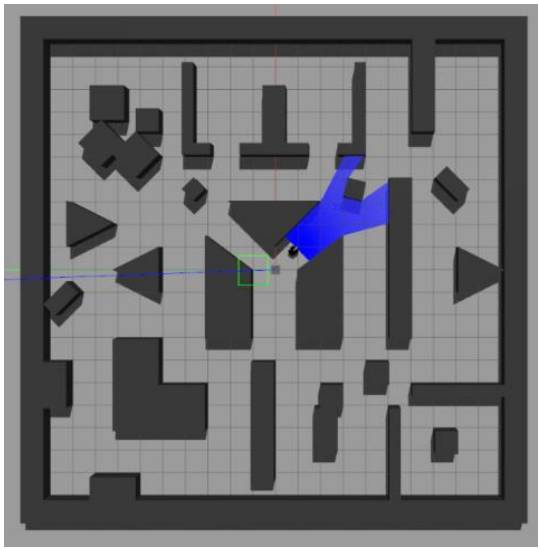


Figure 4.5 Testing cluttered environment

Table 4.3: Success rate comparison for different DRL algorithms in cluttered environment

Algorithm	Successful Episode	Success Rate (%age)
SAC-Dense	56/64	87.50
SAC-SLAM	48/64	75.00
SAC-Sparse	55/64	85.94
TD3-SLAM	46/64	71.88
TD3-Sparse	40/64	62.50

TD3-Dense	30/64	46.88
DDPG-Dense	10/64	15.62
DDPG-SLAM	29/64	45.31%
DDPG-Sparse	50/64	78.12

To ensure a fair comparison, all tested algorithms including SAC, TD3-Dense, TD3-Sparse, and TD3-SLAM evaluated in a cluttered environment for 64 episodes using same start and goal positions. The graphical results shows that SAC reach the goal position in shortest time compared to other alorithms. TD3-Dense perform well, but TD3-Sparse and TD3-SLAM demonstartes longer time to reach goal and longer path lengths. Hence SAC outperforms in cluttered environment as well and shows higher path optimization and time efficiency than other algorithms.

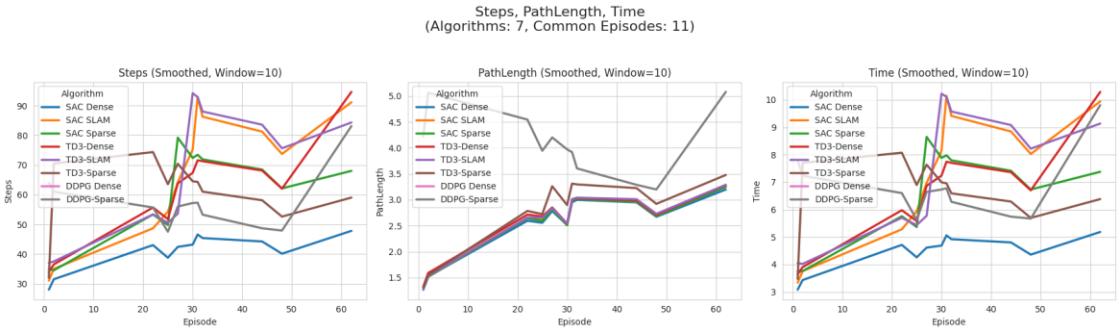


Figure 4.6 Evaluation of performance metrics in cluttered environments.

Figures show that the algorithms' navigation performance varies significantly. With the fewest steps and the quickest time to completion, SAC Dense proved to be the most effective approach. This suggests that SAC can learn policies that lead to clear and efficient navigation strategies when directed by a dense reward function with a good shape. Although they needed a few more steps and time than SAC Dense, SAC SLAM and SAC Sparse also demonstrated encouraging results. These findings imply that SAC retains its learning stability in the presence of scant or perception-based feedback. On the other hand, TD3 and DDPG variants had more difficulty, especially when rewards were sparse. Around

episode 30, TD3-SLAM and DDPG-Sparse showed a discernible increase in steps and time, most likely as a result of challenges adjusting to complex and cluttered environment.

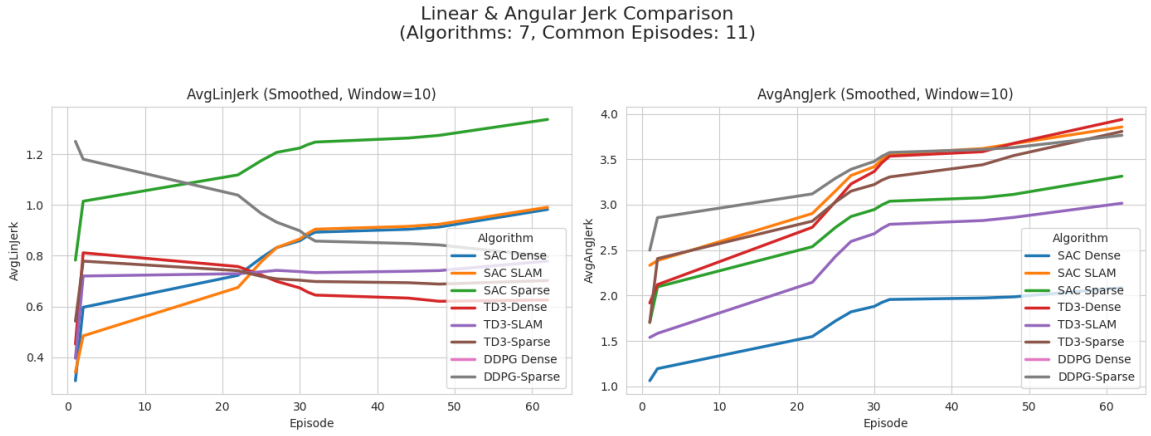


Figure 4.7 Linear and angular jerk analysis in cluttered environment.

Robot motion smoothness is shown in Figure 2 using average linear and angular jerk metrics. These measurements show how quickly the robot changes direction or speed, which has an immediate effect on safety and trajectory stability in confined spaces. Throughout training, SAC Dense maintained low jerk values, demonstrating its superior performance once more. This illustrates how the algorithm can learn smooth, controlled policies in addition to efficient ones.

Although SAC SLAM's slightly higher values than SAC Dense's could be due to noise or delay in SLAM-based state estimation, SAC SLAM also maintained a comparatively low jerk. Because of the exploratory behaviour brought on by the inconsistent feedback, SAC Sparse displayed higher jerk. However, compared to TD3 and DDPG, its performance stayed more consistent. Especially in sparse settings with little reward feedback, the TD3 and DDPG variants displayed noticeably higher linear and angular jerk values. These findings imply that TD3 and DDPG typically generate more abrupt and unreliable trajectories in the absence of dense or well-structured feedback.

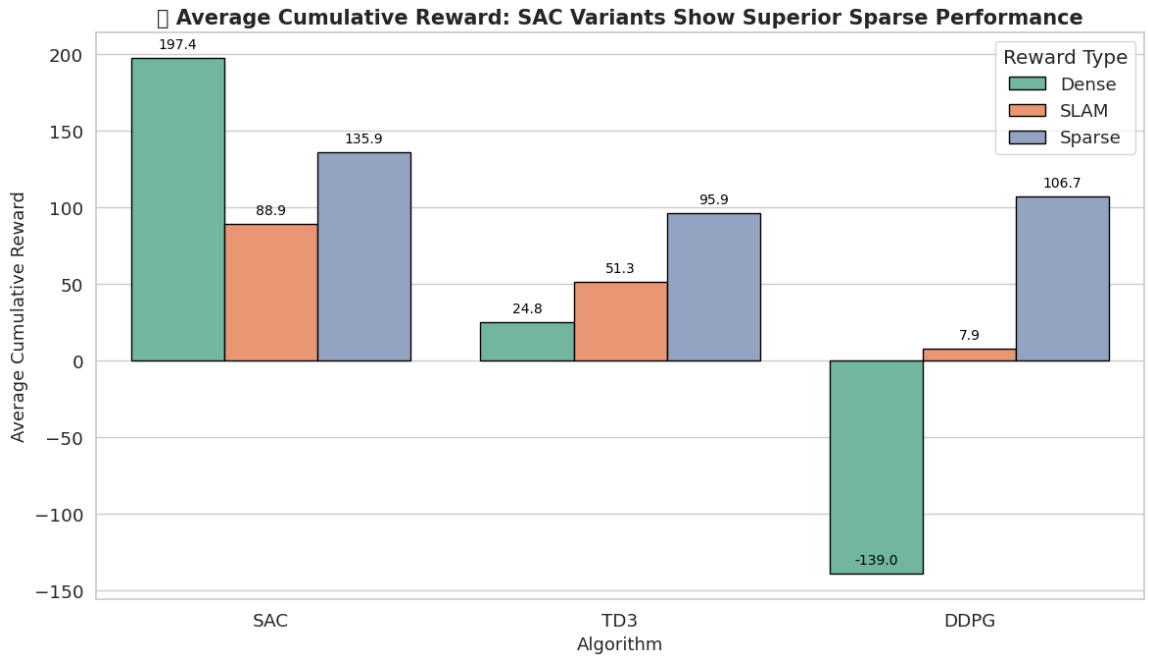


Figure 4.8 Cumulative reward function analysis of all DRL algorithms.

SAC performs better in learning across all reward settings, as shown by the average cumulative reward comparison in Figure 3. Strong policy convergence in an environment with structured feedback was demonstrated by SAC Dense, which received the highest reward of 197.4. Amazingly, SAC Sparse outperformed all TD3/DDPG variants and even SAC SLAM, achieving the second-highest reward of 135.9. This demonstrates that SAC can learn efficiently even in settings with few reward cues.

Sparse rewards generated the best results for this algorithm, and TD3 demonstrated a moderate level of learning performance. Nevertheless, in every reward category, TD3 was still unable to match SAC's performance. However, DDPG had a lot of trouble, especially when it came to dense rewards, where it was unable to converge and ended up with a negative reward of -139.0. DDPG only achieved satisfactory results (106.7) under sparse rewards, but it was still inferior to SAC Sparse.

These findings demonstrate that SAC's entropy-regularized framework makes it easier to conduct more consistent exploration and policy refinement, which makes it more appropriate for high-complexity and partially observable environments.

4.4 Summary

Deep reinforcement learning algorithms are evaluated over 64 episodes with consistent start and goal positions in both sparse and cluttered environments. SAC achieved best overall performance among all algorithms by generate highest reward and shortest paths. TD3-Dense trailed closely behind, exhibiting strong reward accumulation and effective navigation. In all metrics, TD3-SLAM performed moderately well, striking a balance between stability and exploration. However, the limitations of sparse reward structures were confirmed by the longer paths, slower execution times, and unstable rewards of TD3-Sparse and DDPG.

Higher complexity and more obstacles introduced in cluttered environment. SAC continued to perform at its highest rate, finding the most efficient paths and reaching goal position quickly. TD3-Dense also achieved competitive navigation performance but less than SAC. TD3-SLAM and TD3-Sparse demonstrate reduced effectiveness and struggled in cluttered environment. Results shows in both environments that SAC shows most robust and reliable algorithm.

CHAPTER 5: CONCLUSION

A motion planning framework based on Deep Reinforcement Learning (DRL) was introduced in this study for autonomous robots navigating in complex environments. The proposed approach used deep reinforcement learning algorithms, such as DDPG, SAC, and TD3, with an emphasis on improving performance by integrating SLAM and dense reward functions. All algorithms are evaluated in unseen sparse and highly cluttered environment to achieve efficient, collision free and smooth navigation. Based on the experimental data, SAC outperformed TD3 and DDPG significantly in maximizing rewards, motion smoothness, and trajectory effectiveness. While SAC using sparse and SLAM-based rewards also learned sturdily, SAC with dense rewards produced the optimal results. While DDPG failed to converge when encountering complicated scenarios, especially when it encountered dense rewards, TD3 learned moderately but had to bear difficulty in following smooth trajectories. Based on these results, SAC is an attractive option for navigating robots in unstructured real-world environments since it is robust and stable to learn.

5.1 Contribution

Here are the main contributions of this work:

1. In this thesis, a comprehensive comparison of SAC, TD3, and DDPG in an environment cluttered with obstacles is given highlighting the strengths and weaknesses of each algorithm with varying reward scheme.
2. For modelling more realistic and perception-based rewards, a new reward function has been proposed based on SLAM-based feedback. This redefinition enables closing the simulation to reality gap for robotic applications.
3. The work integrated linear and angular jerk analysis with normal performance metrics for the evaluation of the smoothness and safety of learned trajectories, which is an important aspect for real-world deployment.
4. The present work proposes more generalizable and scalable DRL methods by demonstrating that SAC is competitive even when operating in sparse feedback, unlike most of the prior work that completely depends on dense reward shaping.

5.2 Future Work

The Global Future work can investigate sim-to-real approaches such as domain randomization to generalize the learned models to real robots. Further, decision-making under partial observability can be enhanced by applying the framework to multi-agent or multi-robot systems or by incorporating temporal memory with recurrent networks such as GRU or LSTM.

REFERENCES

[1] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, and T. Tolio, "Motion planning and scheduling for human and industrial-robot collaboration," *CIRP Annals*, vol. 66, no. 1, pp. 1–4, Jan. 2017, doi: 10.1016/J.CIRP.2017.04.095.

[2] A. F. Valle, G. Alenyà, G. Chance, P. Caleb-Solly, S. Dogramadzi, and C. Torras, "Personalized Robot Assistant for Support in Dressing," *IEEE Trans Cogn Dev Syst*, vol. 11, no. 3, pp. 363–374, Sep. 2019, doi: 10.1109/TCDS.2018.2817283.

[3] S. Pellegrinelli, N. Pedrocchi, L. M. Tosatti, A. Fischer, and T. Tolio, "Multi-robot spot-welding cells for car-body assembly: Design and motion planning," *Robot Comput Integr Manuf*, vol. 44, pp. 97–116, Apr. 2017, doi: 10.1016/J.RCIM.2016.08.006.

[4] C. Katrakazas, M. Quddus, W. H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transp Res Part C Emerg Technol*, vol. 60, pp. 416–442, Nov. 2015, doi: 10.1016/J.TRC.2015.09.011.

[5] C. Yan, X. Xiang, and C. Wang, "Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 98, no. 2, pp. 297–309, May 2020, doi: 10.1007/S10846-019-01073-3/METRICS.

[6] I. Streinu, "Combinatorial approach to planar non-colliding robot arm motion planning," *Annual Symposium on Foundations of Computer Science - Proceedings*, pp. 443–453, 2000, doi: 10.1109/SFCS.2000.892132.

[7] D. Kularatne, S. Bhattacharya, and M. A. Hsieh, "Going with the flow: a graph based approach to optimal path planning in general flows," *Auton Robots*, vol. 42, no. 7, pp. 1369–1387, Oct. 2018, doi: 10.1007/S10514-018-9741-6/FIGURES/22.

[8] T. Fang and Y. Ding, “A sampling-based motion planning method for active visual measurement with an industrial robot,” *Robot Comput Integr Manuf*, vol. 76, p. 102322, Aug. 2022, doi: 10.1016/J.RCIM.2022.102322.

[9] S. S. Ge and Y. J. Cui, “Dynamic motion planning for mobile robots using potential field method,” *Auton Robots*, vol. 13, no. 3, pp. 207–222, Nov. 2002, doi: 10.1023/A:1020564024509/METRICS.

[10] F. Gul, I. Mir, D. Alarabiat, H. M. Alabool, L. Abualigah, and S. Mir, “Implementation of bio-inspired hybrid algorithm with mutation operator for robotic path planning,” *J Parallel Distrib Comput*, vol. 169, pp. 171–184, Nov. 2022, doi: 10.1016/J.JPDC.2022.06.014.

[11] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, “A Machine Learning Approach for Feature-Sensitive Motion Planning,” *Springer Tracts in Advanced Robotics*, vol. 17, pp. 361–376, 2005, doi: 10.1007/10991541_25.

[12] R. Ma, J. Chen, and J. Oyekan, “A learning from demonstration framework for adaptive task and motion planning in varying package-to-order scenarios,” *Robot Comput Integr Manuf*, vol. 82, p. 102539, Aug. 2023, doi: 10.1016/J.RCIM.2023.102539.

[13] R. Meyes *et al.*, “Motion Planning for Industrial Robots using Reinforcement Learning,” *Procedia CIRP*, vol. 63, pp. 107–112, Jan. 2017, doi: 10.1016/J.PROCIR.2017.03.095.

[14] A. K. Guruji, H. Agarwal, and D. K. Parsediya, “Time-efficient A* Algorithm for Robot Path Planning,” *Procedia Technology*, vol. 23, pp. 144–149, Jan. 2016, doi: 10.1016/J.PROTCY.2016.03.010.

[15] F. L. L. Medeiros and J. D. S. Da Silva, “A Dijkstra Algorithm for Fixed-Wing UAV Motion Planning Based on Terrain Elevation,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in*

Bioinformatics), vol. 6404 LNAI, pp. 213–222, 2010, doi: 10.1007/978-3-642-16138-4_22.

[16] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," *Proc IEEE Int Conf Robot Autom*, pp. 1478–1483, 2011, doi: 10.1109/ICRA.2011.5980479.

[17] M. G. Tamizi, M. Yaghoubi, and H. Najjaran, "A review of recent trend in motion planning of industrial robots," *Int J Intell Robot Appl*, vol. 7, no. 2, pp. 253–274, Jun. 2023, doi: 10.1007/S41315-023-00274-2/TABLES/5.

[18] M. Everett, Y. F. Chen, and J. P. How, "Motion Planning among Dynamic, Decision-Making Agents with Deep Reinforcement Learning," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3052–3059, Dec. 2018, doi: 10.1109/IROS.2018.8593871.

[19] Y. Dai, G. Wang, K. Muhammad, and S. Liu, "A closed-loop healthcare processing approach based on deep reinforcement learning," *Multimed Tools Appl*, vol. 81, no. 3, pp. 3107–3129, Jan. 2022, doi: 10.1007/S11042-020-08896-5/TABLES/13.

[20] Y. Zhang, W. Zhao, J. Wang, and Y. Yuan, "Recent progress, challenges and future prospects of applied deep reinforcement learning : A practical perspective in path planning," *Neurocomputing*, vol. 608, p. 128423, Dec. 2024, doi: 10.1016/J.NEUCOM.2024.128423.

[21] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J Big Data*, vol. 6, no. 1, pp. 1–48, Dec. 2019, doi: 10.1186/S40537-019-0197-0/FIGURES/33.

[22] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-September, pp. 23–30, Dec. 2017, doi: 10.1109/IROS.2017.8202133.

[23] V. R. F. Miranda, A. A. Neto, G. M. Freitas, and L. A. Mozelli, “Generalization in Deep Reinforcement Learning for Robotic Navigation by Reward Shaping,” *IEEE Transactions on Industrial Electronics*, vol. 71, no. 6, pp. 6013–6020, Jun. 2024, doi: 10.1109/TIE.2023.3290244.

[24] A. Vukolov, “D-Star-Based Optimized Trajectory Planner for Mobile Robots Operating in Dense Environments,” *Mechanisms and Machine Science*, vol. 165 MMS, pp. 294–301, 2024, doi: 10.1007/978-3-031-67295-8_33.

[25] L. Palmieri, S. Koenig, and K. O. Arras, “RRT-based nonholonomic motion planning using any-angle path biasing,” *Proc IEEE Int Conf Robot Autom*, vol. 2016-June, pp. 2775–2781, Jun. 2016, doi: 10.1109/ICRA.2016.7487439.

[26] A. Arab, K. Yu, J. Yi, and D. Song, “Motion planning for aggressive autonomous vehicle maneuvers,” *IEEE International Conference on Automation Science and Engineering*, vol. 2016-November, pp. 221–226, Nov. 2016, doi: 10.1109/COASE.2016.7743384.

[27] I. Noreen, A. Khan, H. Ryu, N. L. Doh, and Z. Habib, “Optimal path planning in cluttered environment using RRT*-AB,” *Intell Serv Robot*, vol. 11, no. 1, pp. 41–52, Jan. 2018, doi: 10.1007/S11370-017-0236-7/FIGURES/10.

[28] K. Cao, Q. Cheng, S. Gao, Y. Chen, and C. Chen, “Improved PRM for Path Planning in Narrow Passages,” *Proceedings of 2019 IEEE International Conference on Mechatronics and Automation, ICMA 2019*, pp. 45–50, Aug. 2019, doi: 10.1109/ICMA.2019.8816425.

[29] C. Lamini, S. Benhlima, and A. Elbekri, “Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning,” *Procedia Comput Sci*, vol. 127, pp. 180–189, Jan. 2018, doi: 10.1016/J.PROCS.2018.01.113.

[30] L. Zhang, Y. Zhang, and Y. Li, “Mobile Robot Path Planning Based on Improved Localized Particle Swarm Optimization,” *IEEE Sens J*, vol. 21, no. 5, pp. 6962–6972, Mar. 2021, doi: 10.1109/JSEN.2020.3039275.

[31] W. Xu, Q. Wang, and J. M. Dolan, “Autonomous Vehicle Motion Planning via Recurrent Spline Optimization,” *Proc IEEE Int Conf Robot Autom*, vol. 2021-May, pp. 7730–7736, 2021, doi: 10.1109/ICRA48506.2021.9560867.

[32] C. Sun, Q. Li, B. Li, and L. Li, “A Successive Linearization in Feasible Set Algorithm for Vehicle Motion Planning in Unstructured and Low-Speed Scenarios,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 4, pp. 3724–3736, Apr. 2022, doi: 10.1109/TITS.2020.3041075.

[33] M. F. Abdelwahed, A. E. Mohamed, and M. A. Saleh, “Solving the motion planning problem using learning experience through case-based reasoning and machine learning algorithms,” *Ain Shams Engineering Journal*, vol. 11, no. 1, pp. 133–142, Mar. 2020, doi: 10.1016/J.ASEJ.2019.10.007.

[34] K. Saleh, M. Attia, M. Hossny, S. Hanoun, S. Salaken, and S. Nahavandi, “Local Motion Planning for Ground Mobile Robots via Deep Imitation Learning,” *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, pp. 4077–4082, Jul. 2018, doi: 10.1109/SMC.2018.00691.

[35] J. H. Cui, R. X. Wei, Z. C. Liu, and K. Zhou, “UAV Motion Strategies in Uncertain Dynamic Environments: A Path Planning Method Based on Q-Learning Strategy,” *Applied Sciences 2018, Vol. 8, Page 2169*, vol. 8, no. 11, p. 2169, Nov. 2018, doi: 10.3390/APP8112169.

[36] L. Lv, S. Zhang, D. Ding, and Y. Wang, “Path Planning via an Improved DQN-Based Learning Policy,” *IEEE Access*, vol. 7, pp. 67319–67330, 2019, doi: 10.1109/ACCESS.2019.2918703.

[37] V. Arumugam, V. Alagumalai, and V. Sriniva, “Deep Reinforcement Learning based Path Planning with Dynamic Trust Region Optimization for Automotive Application,” Sep. 2024, doi: 10.21203/RS.3.RS-4948392/V1.

[38] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robots," *J Intell Manuf*, vol. 33, no. 2, pp. 387–424, Feb. 2022, doi: 10.1007/S10845-021-01867-Z/FIGURES/29.

[39] S. Deshpande, H. R, B. S. K. K. Ibrahim, and M. D. S. Ponnuru, "Mobile robot path planning using deep deterministic policy gradient with differential gaming (DDPG-DG) exploration," *Cognitive Robotics*, vol. 4, pp. 156–173, Jan. 2024, doi: 10.1016/J.COGR.2024.08.002.

[40] Z. Feiyu, L. Dayan, W. Zhengxu, M. Jianlin, and W. Niya, "Autonomous localized path planning algorithm for UAVs based on TD3 strategy," *Scientific Reports 2024 14:1*, vol. 14, no. 1, pp. 1–10, Jan. 2024, doi: 10.1038/s41598-024-51349-4.

[41] L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, and Y. Zhao, "Safe Reinforcement Learning with Stability Guarantee for Motion Planning of Autonomous Vehicles," *IEEE Trans Neural Netw Learn Syst*, vol. 32, no. 12, pp. 5435–5444, Dec. 2021, doi: 10.1109/TNNLS.2021.3084685.

[42] X. Ruan, D. Ren, X. Zhu, and J. Huang, "Mobile Robot Navigation based on Deep Reinforcement Learning," *Proceedings of the 31st Chinese Control and Decision Conference, CCDC 2019*, pp. 6174–6178, Jun. 2019, doi: 10.1109/CCDC.2019.8832393.

[43] G. Chen *et al.*, "Deep Reinforcement Learning of Map-Based Obstacle Avoidance for Mobile Robot Navigation," *SN Comput Sci*, vol. 2, no. 6, pp. 1–14, Nov. 2021, doi: 10.1007/S42979-021-00817-Z/FIGURES/13.

[44] R. B. Grando *et al.*, "Deep Reinforcement Learning for Mapless Navigation of a Hybrid Aerial Underwater Vehicle with Medium Transition," *Proc IEEE Int Conf Robot Autom*, vol. 2021-May, pp. 1088–1094, 2021, doi: 10.1109/ICRA48506.2021.9561188.

[45] K. Wu, W. Han, M. Abolfazli Esfahani, and S. Yuan, "Learn to Navigate Autonomously Through Deep Reinforcement Learning," *IEEE Transactions on*

Industrial Electronics, vol. 69, no. 5, pp. 5342–5352, May 2022, doi: 10.1109/TIE.2021.3078353.

[46] W. Zhu and M. Hayashibe, "A Hierarchical Deep Reinforcement Learning Framework With High Efficiency and Generalization for Fast and Safe Navigation," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 5, pp. 4962–4971, May 2023, doi: 10.1109/TIE.2022.3190850.

[47] R. Cimurs, I. H. Suh, and J. H. Lee, "Goal-Driven Autonomous Exploration through Deep Reinforcement Learning," *IEEE Robot Autom Lett*, vol. 7, no. 2, pp. 730–737, Apr. 2022, doi: 10.1109/LRA.2021.3133591.

[48] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning," *IEEE Trans Veh Technol*, vol. 69, no. 12, pp. 14413–14423, Dec. 2020, doi: 10.1109/TVT.2020.3034800.

[49] V. R. F. Miranda, A. A. Neto, G. M. Freitas, and L. A. Mozelli, "Generalization in Deep Reinforcement Learning for Robotic Navigation by Reward Shaping," *IEEE Transactions on Industrial Electronics*, vol. 71, no. 6, pp. 6013–6020, Jun. 2024, doi: 10.1109/TIE.2023.3290244.

[50] M. Shahid, S. N. Khan, K. F. Iqbal, S. Ali, and Y. Ayaz, "Dynamic Goal Tracking for Differential Drive Robot Using Deep Reinforcement Learning," *Neural Process Lett*, vol. 55, no. 8, pp. 11559–11576, Dec. 2023, doi: 10.1007/S11063-023-11390-2/TABLES/3.

LIST OF PUBLICATIONS

Journal Paper titled as “Robot Motion Planning using Deep Reinforcement Learning” for journal of engineering application of Artificial Intelligence ready for submission.