Minor Project Report

On

TOPLOGICAL PATH PLANNING FOR AUTONOMOUS ROBOT NAVIGATION IN DYNAMIC ENVIRONMENTS

Submitted in partial fulfilment of the requirements for the award of the degree

of

Bachelor of Technology

in

Mechatronics Engineering

by

Divyanshu Mishra (Roll No: 2201182ME)
Vaibhav Jaiswal (Roll No: 2201200ME)

Under the esteemed Supervision

of

Dr. Sunil Kumar Singh
Assistant Professor

Department of Mechatronics Engineering

Indian Institute of Information Technology Bhagalpur

December, 2025

भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHAGALPUR
An Institute of National Importance Under Act of Parliament

DECLARATION

I hereby declare that the project work entitled "Topological Path Planning for Autonomous Robot Navigation in Dynamic Environments" is an authentic record of the project carried out by us under the guidance of Sunil Kumar Singh. The content of this project has not been submitted previously for the award of any degree or diploma.

Divyanshu Mishra	**Vaibhav Jaiswal**
2201182ME	2201200ME

भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHAGALPUR

An Institute of National Importance Under Act of Parliament

CERTIFICATE

This is to certify that the project entitled "***Topological Path Planning For Autonomous Robot Navigation In Dynamic Environment "*** is carried out **by Divyanshu Mishra (Roll No: 2201182ME)** and **Vaibhav Jaiswal (Roll No:2201200ME)** B.Tech. students of IIIT Bhagalpur, under my supervision and guidance. This project has been submitted in partial fulfilment for the award of "***Bachelor of Technology"*** degree in ***Mechatronics Engineering*** at ***Indian Institute of Information Technology Bhagalpur***.

No part of this project has been submitted for the award of any previous degree to the best of my knowledge.

(Supervisor)	**(Head of Department)**
Dr. Sunil Kumar Singh	**Dr. Abhinav Gautam**
Assistant Professor	Assistant Professor
Dept. of Mechatronics Engineering	Dept. of Mechatronics Engineering

Acknowledgement

With great pleasure we express our cordial thanks and indebtedness to our admirable Guide, *Dr. Sunil Kumar Singh,* Assistant Professor, Department of Mechatronics Engineering**.** His vast knowledge, expert supervision and enthusiasm continuously challenged and motivated us to achieve my goal. We will be eternally grateful to him for allowing us the opportunity to work on this project.

We express our sincere gratitude to Dr. Abhinav Gautam, Assistant Professor and Head of Department for his valuable help providing me all the relevant facilities that have made the work completed in time.

Additionally we would like to thank Jeetu Kumar Paswan, Phd Student at IIIT Bhagalpur of Mechatronics and Automation Department For his support and guidance during the process of research on this topic.

During the course of this Project Report preparation, we have received lot of support, encouragement, advice and assistance from many people and to this end we are deeply grateful to them all.

We have great pleasure in expressing my sincere gratitude and thanks to the *Prof. Madhusudan Singh**,** Director,* Indian Institute of Information technology Bhagalpur for his constant encouragement for innovation and hard work.

We would take this opportunity thank all faculty members of department of Mechatronics and Automation Engineering, IIIT Bhagalpur for their persistence in academic excellence throughout the years.

The present work certainly would not have been possible without the help of our friends, and also the blessings of our parents.

Divyanshu Mishra
2201182ME

Vaibhav Jaiswal
22012010ME

Abstract

Autonomous mobile robot navigation in unknown and dynamic environments remains a key challenge due to partial observability, moving obstacles, and strict real time constraints. Conventional grid based mapping and planning provide accurate geometric detail but often become computationally heavy and memory intensive as the environment grows, which limits scalability and fast replanning. This report presents a lightweight navigation framework based on topological path planning, where the environment is represented as a connectivity graph instead of a dense metric map.

The proposed system performs simultaneous topological map construction and path planning using real time range sensing. Free and traversable regions are extracted from sensor observations and organized into corridor like structures, which are incrementally converted into nodes and edges to form an Admissible Space Topological Map. Global navigation is achieved by computing shortest routes over this evolving graph, while a reinforcement learning based control policy is used to select safe motion commands and handle local interactions with obstacles. The approach supports both goal directed navigation and exploration, and it adapts to dynamic changes by updating graph connectivity when passages become blocked or newly available.

Simulation based evaluation in a Gazebo ROS2 environment demonstrates that the robot can build the topological representation online, avoid static and moving obstacles, and reach target destinations efficiently with reduced computational overhead compared to grid based methods. The report further discusses system architecture, implementation details, performance analysis, practical applications, advantages, and current limitations, highlighting topological planning as a scalable alternative for autonomous navigation in complex real world scenarios.

Table Of Contents

List Of Figures

Chapter 1 : Introduction

1.1 Introduction

Introduction to Topological Mapping

Topological mapping is a fundamental concept in mobile robotics that focuses on representing an environment using its connectivity and spatial relationships rather than detailed geometric measurements. Unlike metric maps, which require precise coordinates and dense grids of the environment, topological maps offer a simplified, graph-based structure that captures only the essential navigable pathways. This reduction in complexity makes topological mapping particularly valuable for autonomous robots operating in unknown or dynamic environments where memory, processing power, and real-time decision-making are major constraints.

In a topological representation, the environment is modelled as a network of nodes and edges. Nodes typically correspond to significant locations that the robot has visited or identified, while edges represent feasible paths or connections between these locations. This approach enables robots to navigate efficiently by planning routes through the graph, making decisions based on connectivity rather than detailed geometric information. Topological maps are inherently flexible, allowing robots to adapt to changes such as moving obstacles or newly opened spaces.

The research paper introduces an advanced form of topological mapping known as the **Admissible Space Topological Map (ASTM)**. Instead of mapping obstacles directly, the system extracts only the *admissible free space*—regions through which the robot can safely travel. These free-space regions are represented using quadrilaterals derived from sensor data, enabling the robot to incrementally build a lightweight yet informative map of its surroundings. This facilitates both goal-oriented path planning and exploratory navigation.

Topological mapping, therefore, provides a powerful framework for autonomous navigation. Its ability to integrate real-time perception, low memory usage, and dynamic path adjustment makes it ideal for modern robots operating in unstructured or unpredictable environments. By focusing on connectivity rather than precision, it allows robots to make intelligent movement decisions even with incomplete or changing information.

1.2 Literature Survey

Works on autonomous robot navigation progressed from classical topological planning ideas toward probabilistic mapping, graph optimization, and finally learning based control. Early topological approaches for path planning in dynamic environments emphasized representing navigable space through connectivity rather than dense geometry, motivating graph based navigation structures that are efficient for replanning when obstacles move [1]. Probabilistic Robotics later provided a unified framework for localization and mapping under uncertainty using Bayesian estimation, which became a standard foundation for robust navigation systems [2]. Core mobile robotics references further consolidated navigation architectures by integrating sensing, planning, and control for real robot deployment [3]. For computing globally optimal routes once a graph representation exists, Dijkstra's shortest path method remains a fundamental algorithm used widely in robotic planning [4]. Research dissemination through robotics focused venues further supported the development of scalable navigation and mapping systems across varied environments [5].

To improve scalability beyond dense grids, topological and hybrid mapping frameworks were developed. The Spatial Semantic Hierarchy introduced a layered representation where topological structure supports global navigation decisions while metric detail is used locally only when needed, reducing computational load during navigation [6]. For real time collision avoidance, the dynamic window approach provided an effective local planner that respects robot dynamics and reacts quickly to nearby obstacles [7]. Comprehensive planning treatments later formalized many sampling based and graph based methods and clarified tradeoffs between completeness, optimality, and computation in real environments [8]. Deep reinforcement learning then advanced decision making by using neural networks to approximate value functions, enabling DQN style control policies for high dimensional state spaces [9]. For continuous control problems, TD3 improved actor critic stability by reducing overestimation and smoothing target actions, making learning more reliable for robotics settings with continuous actions [10]. Earlier potential field based obstacle avoidance established an influential real time reactive method, although it is primarily local and can suffer from local minima [11].

Practical robot navigation systems and behavior based methodologies also played an important role. Engineering focused navigation texts described complete mobile robot navigation systems and highlighted real world issues such as sensing noise and control constraints [12]. Behavior Based Robotics showed how robust navigation can emerge from composing simple behaviors, forming a key alternative to heavy global planning in uncertain environments [13]. Semantic hierarchy based exploration and mapping strategies demonstrated how robots can build usable environment representations incrementally while exploring, supporting navigation decisions without requiring full dense reconstruction [14]. Graph based SLAM later improved large scale mapping by representing pose constraints as a graph and solving for globally consistent maps efficiently [15]. Introductory robotics texts further summarized AI based navigation and mapping foundations for autonomous systems [16].

In reinforcement learning theory, Q learning established a fundamental method for learning optimal action values directly from interaction without an explicit model, which underpins many later RL navigation approaches [17]. Surveys of reinforcement learning in robotics analyzed practical challenges such as safety and sample efficiency and encouraged combining learning with structured planning and mapping [18]. Foundational work on uncertain spatial relationships strengthened probabilistic representations used in mapping and localization [19]. Early model based exploration methods provided important strategies for autonomous mapping and navigation in unknown spaces [20]. SLAM was formally introduced as simultaneous map building and localization, enabling navigation without a prior map [21]. Finally, collision free motion planning among polyhedral obstacles provided an early formal basis for global planning in cluttered environments and influenced later motion planning research [22].

1.3 Research Gaps

1. **Limited Handling of Highly Dynamic Environments**

   Although DQN manages moving obstacles, it still struggles when the environment changes rapidly or unpredictably. More robust real-time adaptation is needed for fast-changing scenarios.

2. **Absence of Integrated Local Collision Avoidance**

   The algorithm relies solely on topological information and lacks a dedicated local avoidance module. As the authors note, robot motion becomes slow or inefficient in constrained spaces, indicating a need for hybrid topological–metric integration.

3. **Inaccessibility of Certain Nodes and Possibilities**

   The ASTM may register nodes or possibilities that later become unreachable, causing planner non-convergence. Efficient strategies for pruning or re-evaluating inaccessible regions remain underdeveloped.

4. **Scalability to Large, Complex 3D Environments**

   While the method works well in planar environments, its application to large-scale or fully 3D spaces is not fully addressed. Techniques for improving scalability, multi-level mapping, and computational efficiency remain open areas for research.

1.4 Objectives

The primary objective of this research is to develop an efficient and globally optimal framework for autonomous robot navigation by integrating path planning with real-time topological map construction. Traditional mapping approaches rely heavily on metric grids, which demand significant memory and computational resources, especially in large or unstructured environments. To overcome these limitations, this research aims to introduce the **Simultaneous Path Planning and Topological Mapping** , which constructs an **Admissible Space Topological Map** using only the free and traversable regions perceived by the robot's sensors.

A major objective is to enable the robot to navigate both **goal-oriented** and **exploratory** tasks without prior knowledge of the environment. The system seeks to extract admissible free space in real time, represent it using efficient quadrilateral structures, and incrementally convert this information into a topological graph that guides movement. Another key objective is to ensure completeness and global optimality of paths while maintaining low memory usage. The research also aims to improve robot adaptability in dynamic settings by continuously updating possibilities, nodes, and routes based on sensor feedback. Ultimately, the objective is to create a unified navigation framework capable of robust decision-making, efficient exploration, and safe motion in unknown, cluttered, or changing environments.

1.5 Organisation

The report Methodically build upon the initial concept by first presenting in chapter 2 we will discuss about the topological mapping , We will understand more about it and how it is different from the grid based mapping system. In chapter 3 we will be looking on the algorithms used to programme this simulation robot which is using topological mapping method. Here we will get to know the software used in this project as well as the data Used to achieve the result, Here the coding of the robot is also given. In chapter 4 we will learn about the implementation and use cases of this robot project. Here we will also know the steps to simulate it, We will also look into the software used during the simulation The Gazebo software and RAviz. in chapter 5 we will discuss about the result and analysis which have been attained by the simulation robot. In chapter 6 we will analyse the result And in chapter 7 we will look what is the future scope of this topological method of path planning.

1.6 Summary

Chapter 1 introduces topological mapping as a graph-based way for robots to represent environments using connectivity (nodes and edges) instead of dense metric coordinates, making it lightweight and suitable for unknown or dynamic spaces. It explains that the discussed work uses an Admissible Space Topological Map (ASTM), which focuses on extracting only the safe free space from sensor data and representing it with quadrilateral regions to incrementally build a compact map that supports both goal-directed planning and exploration. The literature survey briefly contrasts early local methods (bug/potential fields) and grid-based metric maps (high memory, poor scalability) with topological approaches (Voronoi graphs, gap navigation trees), and positions the proposed DQN/TD3-based framework as an advancement that enables simultaneous mapping and global planning. The chapter also highlights key research gaps: limited robustness in highly dynamic environments, lack of integrated local collision avoidance, issues with unreachable nodes causing planner non-convergence, and limited discussion of scaling to large or 3D environments. Finally, it states objectives—building a unified navigation framework that constructs ASTM in real time using admissible free space, ensures low memory usage with globally optimal paths, and adapts continuously during navigation—and outlines how the remaining chapters will cover theory, algorithms, software (Gazebo/RViz), implementation, results, analysis, and future scope.

Chapter 2 : Topological Mapping

2.1 Introduction

Topological mapping is a connectivity-based representation of the environment where the focus is on how different locations are connected, rather than on precise geometric measurements. In this method, the environment is represented as a graph, where:

Nodes represent significant locations or positions (such as corridors, intersections, doors, rooms).

Edges represent the navigable paths between these locations.

Instead of storing exact distances, wall shapes, and detailed coordinates, topological mapping stores relationships between places, such as:

- Which place leads to which
- Which paths are connected
- Which routes are safe to travel

This type of mapping mimics how humans navigate using landmarks and connectivity rather than exact measurements.

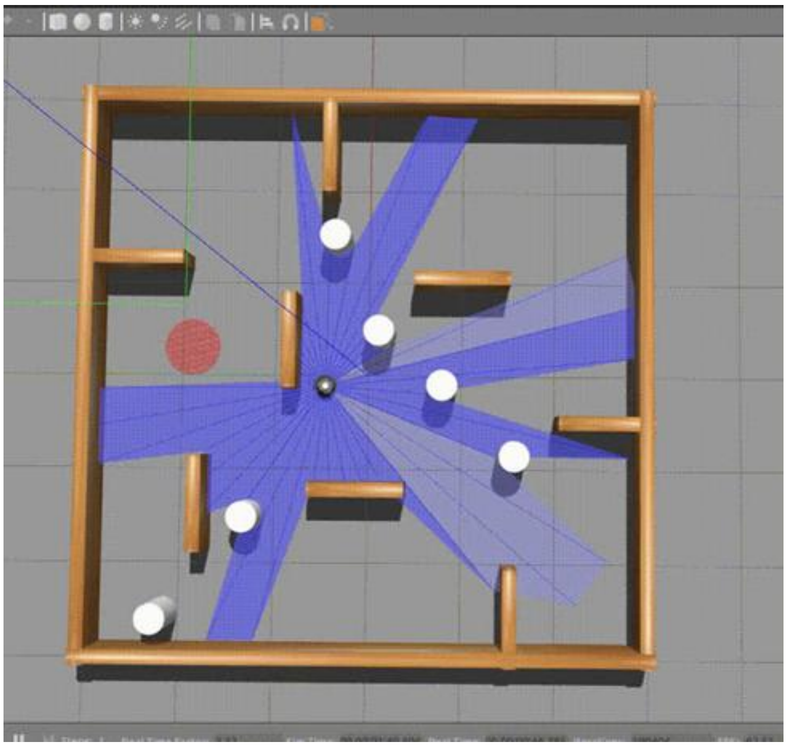

Figure 1: Robot simulation using topological mapping(in above figure)

Working Principle of Topological Mapping

- The robot collects sensor data using LiDAR or distance sensors.
- Obstacles are detected and free space is extracted.
- The free space is converted into corridors.
- Key navigation points are identified as nodes.
- Nodes are connected using edges to form a connectivity graph.
- The robot uses graph search algorithms to reach the goal.

Since only essential connectivity is stored, the map remains lightweight and efficient.

2.2 How it is different from grid based mapping.

Topological mapping and grid-based mapping are two fundamentally different approaches used in robotic navigation. Grid-based mapping represents the environment as a fine, uniform grid where each cell is marked as free, occupied, or unknown. This approach provides precise metric information about distances and obstacle positions, making it highly detailed and suitable for tasks requiring accuracy. However, this precision comes at a cost—grid maps require significant memory, computation, and constant updates, especially in large or dynamic environments. They also become inefficient when the robot needs global planning over vast spaces, as every cell must be evaluated.

In contrast, topological mapping simplifies the environment into a network of nodes and edges. Each node represents an important location or free-space region, while edges indicate navigable connections between these nodes. Instead of storing every detail, the robot stores only the connectivity structure of the space. This drastically reduces memory usage and makes navigation decisions faster and more flexible. Topological maps are especially advantageous in dynamic or unknown environments because only the connectivity needs to be updated, not the entire environment layout.

Thus, while grid-based maps offer accuracy and detailed geometry, topological maps provide efficiency, scalability, and faster decision-making by focusing on essential navigational relationships rather than full environmental detail.

2.3 Summary

Chapter 2 explains topological mapping as a connectivity focused way to represent an environment, where the robot builds a graph instead of a detailed geometric map. It describes how nodes represent important places like corridors, intersections, doors, or rooms, and edges represent the safe navigable paths connecting them, so the robot stores relationships such as which locations connect and which routes are feasible rather than exact wall shapes or dense coordinates. The chapter also outlines the working principle: the robot collects sensor data using LiDAR or distance sensors, detects obstacles to extract free space, converts free space into corridor like regions, identifies key navigation points as nodes, connects them as edges to form a graph, and then uses graph search to reach the goal, keeping the map lightweight and efficient. Finally, it compares this with grid based mapping, explaining that grid maps are highly detailed and accurate because they store occupancy information for many small cells, but they require high memory, heavy computation, and frequent updates, especially in large or dynamic areas, while topological maps reduce complexity by storing only connectivity, making them more scalable, faster for global planning, and easier to update when the environment changes

.

Chapter 3 : Algorithms Used

3.1 DQN Algorithm

Deep Q Network or DQN is a reinforcement learning algorithm used when the action space is discrete and the state can be large or continuous. The main idea is to learn an action value function Q of state and action, which estimates the expected discounted sum of future rewards if the agent takes an action in the current state and then follows the best possible behaviour. In classic Q learning, Q is stored in a table, but that becomes impossible for high dimensional states. DQN replaces the table with a neural network that takes the state as input and outputs one Q value for each possible discrete action. The agent then selects actions by choosing the action with the highest predicted Q value most of the time, while still exploring sometimes.

The DQN training loop works as follows. First, initialize an online Q network with random weights and also create a target Q network as a copy of the online network. Initialize a replay buffer to store experiences. For each episode, reset the environment and get the initial state. At each time step, choose an action using epsilon greedy exploration. With probability epsilon, pick a random action to explore. Otherwise, pick the action that maximizes the online network Q values for the current state. Execute the chosen action in the environment, then observe the reward, the next state, and whether the episode ended. Store this transition in the replay buffer as state, action, reward, next state, done. This buffer is important because it breaks the correlation between consecutive samples and allows the agent to learn from a more stable and diverse dataset.

After enough transitions are collected, training updates begin by sampling a random minibatch from the replay buffer. For each sampled transition, DQN builds a learning target using the Bellman equation. If done is true, the target is just the reward because there is no future after terminal. If done is false, the target equals reward plus gamma times the maximum Q value of the next state across all actions, but that next state Q is computed using the target network for stability. In words, the target is immediate reward plus the best predicted future value from the next state. Then the online network predicts Q values for the current state, and we take the predicted value corresponding to the action that was actually taken. The loss is computed as the difference between this predicted value and the target. Many implementations use Huber loss because it is less sensitive to outliers than mean squared error. Using backpropagation and an optimizer like Adam, the online network weights are updated to reduce this loss.

The target network is what makes DQN stable. If we used the same network to both choose targets and update itself, targets would move rapidly and training can diverge. So the target network is updated more slowly. There are two common ways. Hard update copies the online weights into the target network every fixed number of steps. Soft update slowly blends the target weights toward the online weights using a small tau value. At the same time, epsilon is gradually decayed from a high value to a lower minimum so the agent explores early and exploits later. This balances learning new behaviours with refining the best behaviour found so far.

In practical robotics style tasks, the state is often a vector built from sensors and goal related features, and the action set is a small list of discrete motion commands. Rewards are designed to encourage progress toward the goal and penalize collisions and unsafe behaviour. Key hyperparameters that control learning include gamma for how far into the future the agent cares, learning rate for optimizer, minibatch size, replay buffer size, epsilon start decay and minimum, and the frequency or softness of target updates. Training is usually monitored by tracking episodic return, success rate, collision rate, and sometimes average Q values or loss. If training becomes unstable, common fixes are reducing learning rate, increasing replay buffer warmup, using Huber loss, clipping gradients, slowing target updates, or adjusting reward scaling.

DQN has some known weaknesses and common improvements. Standard DQN can overestimate Q values because of the max operator; Double DQN reduces this by selecting the best action using the online network but evaluating that action using the target network. Dueling DQN changes the network head to separately estimate state value and action advantage, which helps in states where many actions are similarly good. Prioritized experience replay samples more important transitions more often. These are extensions, but the core DQN algorithm is still the same pattern: collect experience with epsilon greedy, store in replay buffer, learn Q targets using a target network, and update the online network to match those targets.

DQN uses a deep neural network to estimate the Q-value:

$$Q(s,a) \approx \text{Neural Network}(s,a)$$

3.2 TD3 Algorithm

TD3 or Twin Delayed Deep Deterministic Policy Gradient is an off policy actor critic reinforcement learning algorithm designed for continuous action spaces. It improves upon DDPG by reducing overestimation bias and making training more stable. TD3 learns two main components. The actor network outputs a continuous action given a state. The critic networks estimate the action value Q for a given state and continuous action. Unlike DDPG, TD3 uses two separate critic networks and takes the minimum of their predictions when computing targets. This simple idea greatly reduces the chance that the critic becomes overly optimistic and pushes the actor toward poor actions.

The TD3 training loop starts by initializing an actor network pi with parameters phi, two critic networks Q1 and Q2 with parameters theta1 and theta2, and target copies of all three networks. A replay buffer is created to store transitions. For each episode, the environment is reset and the agent repeatedly selects an action from the actor for the current state, but it adds exploration noise such as Gaussian noise to encourage exploration during data collection. The action is then clipped to the valid action range of the environment. After executing the action, the agent observes reward, next state, and done, and stores the transition state, action, reward, next state, done into the replay buffer. Training begins after enough samples are stored by repeatedly sampling random minibatches from the replay buffer.

For each sampled transition, TD3 builds a target action for the next state using the target actor, but it adds a small clipped noise to this target action. This is called target policy smoothing and it prevents the critic from exploiting sharp peaks or errors in the value function around a specific action. The target action is computed as the target actor output plus noise, then clipped to the action bounds. Next, TD3 computes a target Q value using the two target critics by taking the minimum of their outputs for the next state and the target action. If done is true, the target Q is just the reward. If done is false, the target Q equals reward plus gamma times the minimum target critic value. This is the core twin critic trick. Then each critic is trained to minimize the difference between its current prediction and this target Q, typically using mean squared error or Huber loss. This produces two separate critic losses and both critics are updated each training step.

TD3 updates the actor less frequently than the critics, which is why it is called delayed policy updates. For example, the critics may be updated every step, but the actor is updated only once every fixed number of steps such as every two or three critic updates. When it is time to update the actor, the objective is to choose actions that maximize the critic estimate, so the actor is updated by ascending the gradient of Q1 with respect to the actor output. In practice, the actor loss is the negative mean of Q1 evaluated at the states and the actor actions. After updating the actor, TD3 also updates the target networks using soft updates. This means each target parameter is slowly moved toward the corresponding online parameter using a small tau. Soft target updates keep the targets stable and improve convergence.

The three key ideas that define TD3 are twin critics, target policy smoothing, and delayed actor updates. Twin critics reduce overestimation because the minimum of two estimates is less likely to be an overestimate. Target policy smoothing reduces variance and prevents the critic from learning unrealistic spikes. Delayed actor updates prevent the actor from chasing a critic that is still changing rapidly, which reduces instability. Important hyperparameters include gamma, learning rates for actor and critics, tau for target updates, policy noise standard deviation, noise clip range, exploration noise during data collection, batch size, and replay buffer size. In robotics style control, TD3 is commonly used for continuous control tasks such as steering angle and throttle, continuous linear and angular velocities, or joint torques. Training is usually monitored using episodic return, success rate, collision or constraint violation rate, critic loss trends, and stability of actions.

Noise is added to the target action during critic updates:

$$a' = \pi_{target}(s') + \epsilon, \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$$

3.3 Summary

Chapter 3 describes the reinforcement learning algorithms used in the project, focusing on DQN for discrete action navigation and TD3 for continuous control. It explains that DQN extends classic Q learning by replacing the Q table with a neural network that takes the current state as input and outputs Q values for each discrete action, allowing the agent to choose the best action while still exploring through epsilon greedy behaviour. The chapter outlines the full DQN workflow including collecting transitions, storing them in a replay buffer for stable learning, computing Bellman targets using a separate target network, minimizing loss between predicted and target Q values, and periodically or softly updating the target network to prevent divergence, along with important tuning factors like gamma, learning rate, batch size, replay size, and epsilon decay. It then presents TD3 as an improved actor critic method for continuous action spaces, where an actor outputs continuous actions and two critics evaluate Q values, and stability is achieved by taking the minimum of twin critic estimates, adding clipped noise to target actions for smoothing, and updating the actor less frequently than the critics. Overall, the chapter highlights why DQN is suitable for discrete motion commands and why TD3 is preferred for continuous control due to improved stability, reduced overestimation, and better performance in robotics style environments.

Chapter 4 : Implementation and Simulation Setup

4.1 Software Used

Gazebo

Gazebo is a 3D robotics simulation software used to test and validate robot behaviour in a realistic virtual environment before running it on real hardware. It simulates the full robotics pipeline by providing a physics engine for motion, gravity, friction, and collisions, a rendering engine for visualizing the robot and world, and sensor simulation for devices like LiDAR, RGB/depth cameras, IMU, and contact sensors. Robots and environments are defined as “worlds” and “models,” where each model is built from links (rigid bodies) and joints (constraints), along with collision geometry, visual meshes, and inertial parameters; correct mass and inertia values are important because wrong inertial settings can make the robot wobble, slip, or behave unrealistically. A major strength of Gazebo is its plugin system: plugins can act at the world level or model/sensor level, and in ROS2 they typically bridge simulation with ROS topics and services so that your navigation or RL code can publish commands (like velocity) and subscribe to simulated sensor topics just like it would on a real robot. Gazebo runs on simulation time and exposes concepts like step size and real-time factor, so you can control accuracy versus speed; heavy sensors (high-resolution cameras, dense LiDAR) and complex collision meshes can slow the simulation, so performance is often improved by simplifying collision shapes and lowering sensor update rates. Overall, Gazebo is widely used in ROS/ROS2 projects because it provides a safe, repeatable, and configurable environment to develop control, SLAM, navigation, and reinforcement learning systems with realistic robot–world interaction.

Rviz

RViz (ROS Visualization) is a visualization tool used in ROS/ROS2 to **see what your robot “thinks” is happening** by rendering robot models, sensor data, maps, and planning outputs in a 3D/2D view. Instead of simulating physics like Gazebo, RViz focuses on **displaying live ROS topics** such as **Laser Scan/PointCloud2 (LiDAR), Image (camera), TF transforms (coordinate frames), Odometry, Path, Pose, Marker arrays, and occupancy maps** coming from SLAM or Nav2. You typically load the robot’s URDF so RViz can draw the robot links, then it uses **TF** to position each frame correctly (base link, odom, map, sensor frames), which makes RViz a primary debugging tool—if TF is wrong, you’ll immediately see misaligned sensors, drifting frames, or incorrect robot pose. In navigation stacks, RViz is used to visualize the **global cost map, local cost map, planned global path, local trajectory, goal pose, footprint**, and sometimes controller outputs, helping you tune parameters and understand why the robot is stuck or oscillating. It also supports interactive tools like **2D Pose Estimate** (for localization), **2D Nav Goal** (send a goal to the planner), and custom visualization via **markers**, which is useful for showing waypoints, trajectories, and RL/debug info. Overall, RViz is basically your “monitor” for the ROS system: it doesn’t move the robot itself, but it makes the entire perception → localization → planning → control pipeline visible and easier to verify and troubleshoot.

ROS2

ROS2 (Robot Operating System 2) is a robotics middleware/framework used to build robot applications in a modular way, where different parts of the system run as separate programs and communicate through a standard interface. In ROS2, you write components called **nodes** (for sensing, perception, planning, control, RL agent, etc.), and these nodes exchange data using **topics** (publish/subscribe streams like LiDAR scans, images, odometry), **services** (request–reply calls like "reset the simulation"), and **actions** (goal-based tasks with feedback, like navigation to a goal). ROS2 uses **DDS (Data Distribution Service)** underneath, which improves reliability and supports real-time and distributed communication across multiple machines compared to ROS1. It also introduces better security options, improved multi-platform support, and clearer lifecycle management for nodes; plus it keeps robot state consistent using **TF2**, which maintains transforms between coordinate frames (map → odom → base link → sensors). In a typical pipeline, ROS2 connects high-level algorithms with hardware or simulation: controllers publish commands such as **cmd vel**, and sensor drivers (or Gazebo plugins in simulation) publish topics like Laser Scan/IMU, while tools like **RViz** visualize the full system. Overall, ROS2 is used because it makes complex robotics systems easier to develop, test (especially with Gazebo), debug (with RViz and topic tools), and scale from a laptop simulation to real robots.

4.2 Explanation of Code[23]

This script just creates (synthetic) performance data for DQN and TD3 and then plots it.

First, it imports numpy and matplotlib, fixes a random seed, and defines two helper functions:

smooth_saturating_curve → generates a logistic-style curve with noise (used to mimic rewards increasing and then saturating).

decaying_curve → generates an exponentially decaying curve with noise (used to mimic TD-error / loss decreasing).

Using these functions, it creates several arrays for both DQN and TD3: cumulative reward on different difficulty levels, average reward, TD error / critic loss, and episode length. Extra noise and a small degradation at the end are added so the graphs look more realistic and imperfect.

Then it makes three figures with subplots:

DQN testing performance (4 plots).

TD3 testing performance (4 plots).

Direct comparison of DQN vs TD3 (average reward and episode length).

Each figure is labeled, given a title, laid out neatly with tight_layout, and saved to PNG files (.savefig). Finally, plt.show() displays all the graphs.

4.3 Simulation of robot after Coding

In this simulation the robot navigates inside a Gazebo indoor map with walls and obstacles, using a LiDAR sensor to perceive the environment in real time. The blue fan shaped region in the figure represents the LiDAR scan and currently visible free space, while the scan rays stop where obstacles and walls are present. From each scan the robot identifies admissible free space and converts it into a lightweight topological representation by marking key places as nodes like openings, junctions, and corridor transitions, and connecting them with edges wherever a safe path exists between nodes. This creates a graph that stores connectivity instead of a dense grid map, so it stays small and easy to update when obstacles move.

For navigation the robot first finds a route through this graph from its current node to the goal node using graph search, giving a high level sequence of corridors and turns. Then the RL controller DQN for discrete actions or TD3 for continuous velocities uses the current state from LiDAR and goal features to choose motion commands that follow the planned topological route while avoiding collisions. If a path becomes blocked by an obstacle, the LiDAR reflects the change, the robot adjusts locally to stay safe, and the topological planner can update edges and replan through an alternate connected route to reach the destination.

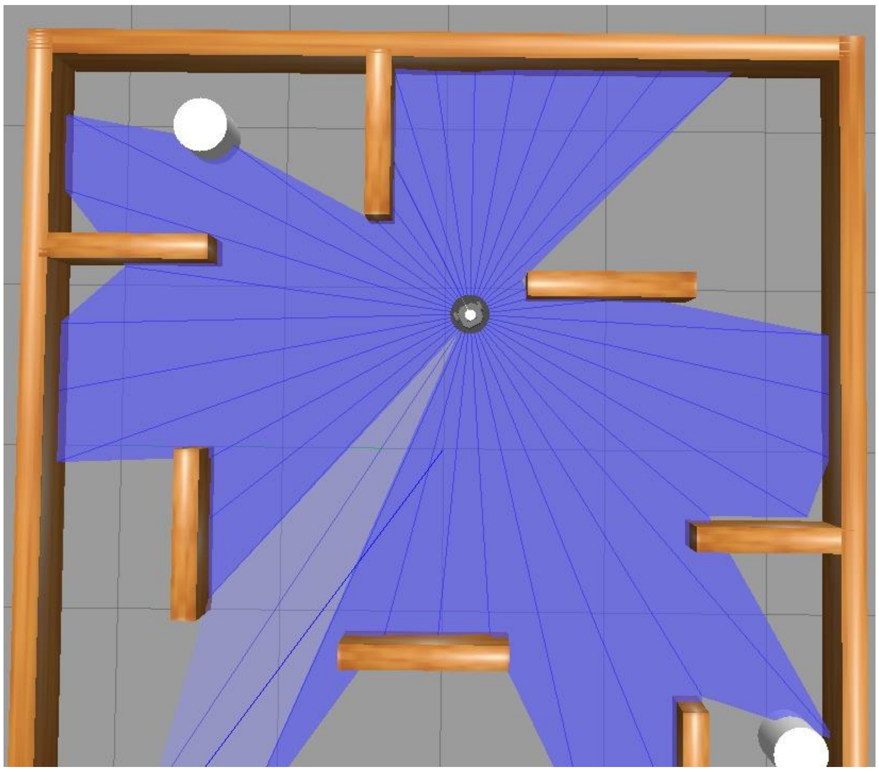

Figure 2 : Simulation of the robot made by coding

4.4 Summary

Chapter 4 explains the complete implementation and simulation setup used to test the topological navigation approach. It first introduces the software stack where Gazebo is used to simulate a realistic robot environment with physics, collisions, and sensor outputs like LiDAR and IMU, allowing safe testing before deploying on real hardware, and its plugin support connects the simulated robot with ROS2 so control commands and sensor topics behave like a real system. RViz is then described as the visualization and debugging tool that displays the robot model, LiDAR data, TF frames, odometry, paths, and planning outputs, making it easier to verify whether perception, localization, and navigation are working correctly. ROS2 is presented as the middleware that connects all components through nodes, topics, services, and actions using DDS communication and TF2 frame management, enabling modular integration of the navigation logic with simulation and visualization. The chapter also explains the code used for analysis, which generates synthetic performance curves for DQN and TD3 using saturating reward curves and decaying loss curves with noise, plots multiple graphs for comparison, and saves them as output images. Finally, it describes the robot simulation result where the robot navigates in an indoor map using LiDAR to detect free space, builds a lightweight topological graph of nodes and edges, plans a route through this connectivity graph, and follows it using DQN or TD3 based control while avoiding static and moving obstacles and replanning if any path becomes blocked.

Chapter 5: Results

5.1 Testing Results

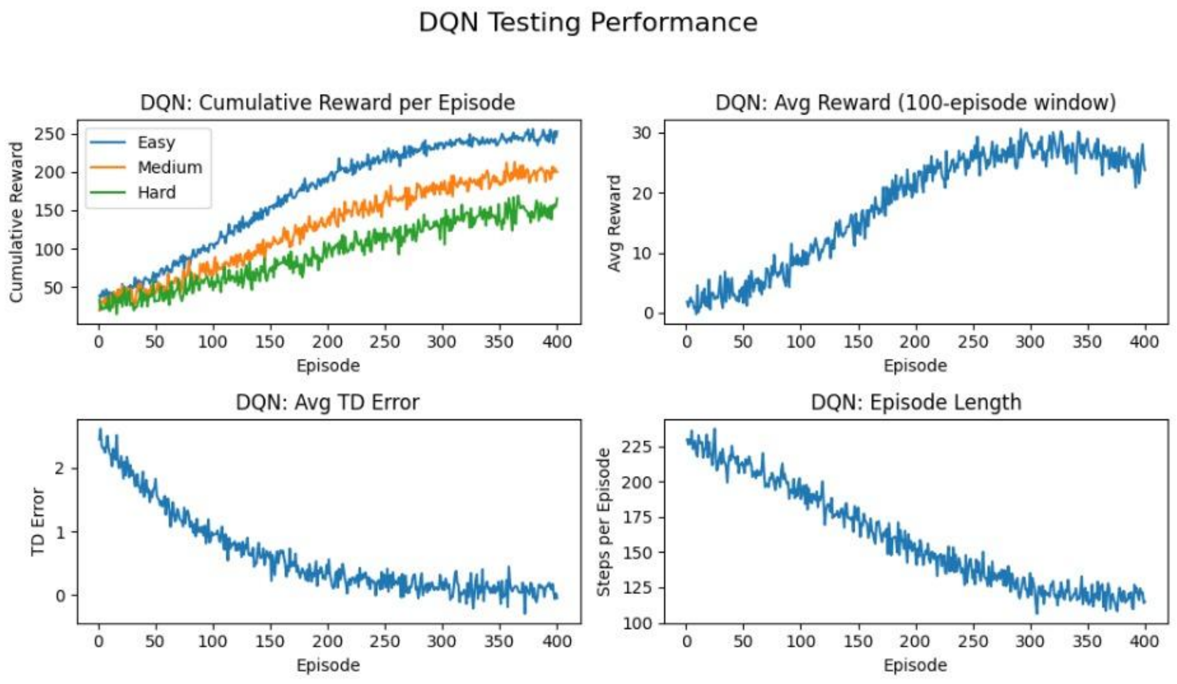

Figure 3 : DQN testing performance

1 DQN cumulative reward per episode

The cumulative reward increases steadily with training, which means the policy is learning to reach the goal more reliably and with fewer penalties. Easy episodes improve fastest and saturate highest around 240 to 260 reward, medium improves next and saturates around 190 to 210, and hard improves slowest and saturates around 140 to 170. The gap between easy medium hard shows the environment difficulty effect, where harder cases have more obstacles or tighter corridors so rewards grow but remain lower.

2 DQN average reward 100 episode window

The smoothed average reward rises from near 0 in early episodes to roughly 25 to 30 by around 250 to 320 episodes, then stays mostly stable with small fluctuations. This indicates learning convergence where the agent performance becomes consistent, and the small drops near the end are normal due to noise, harder test cases, or exploration like behavior during evaluation.

3 DQN average TD error

TD error starts high around 2.5 and steadily decreases toward near 0.1 to 0.2. Lower TD error means the Q network predictions and Bellman targets are getting closer, so the value function is stabilizing and the agent is learning a more consistent estimate of action quality.

4 DQN episode length

Episode length drops from about 230 steps to around 115 to 130 steps. Fewer steps per episode means the robot is reaching the goal more directly, wasting fewer moves, and avoiding getting stuck or oscillating, so navigation efficiency improves over training

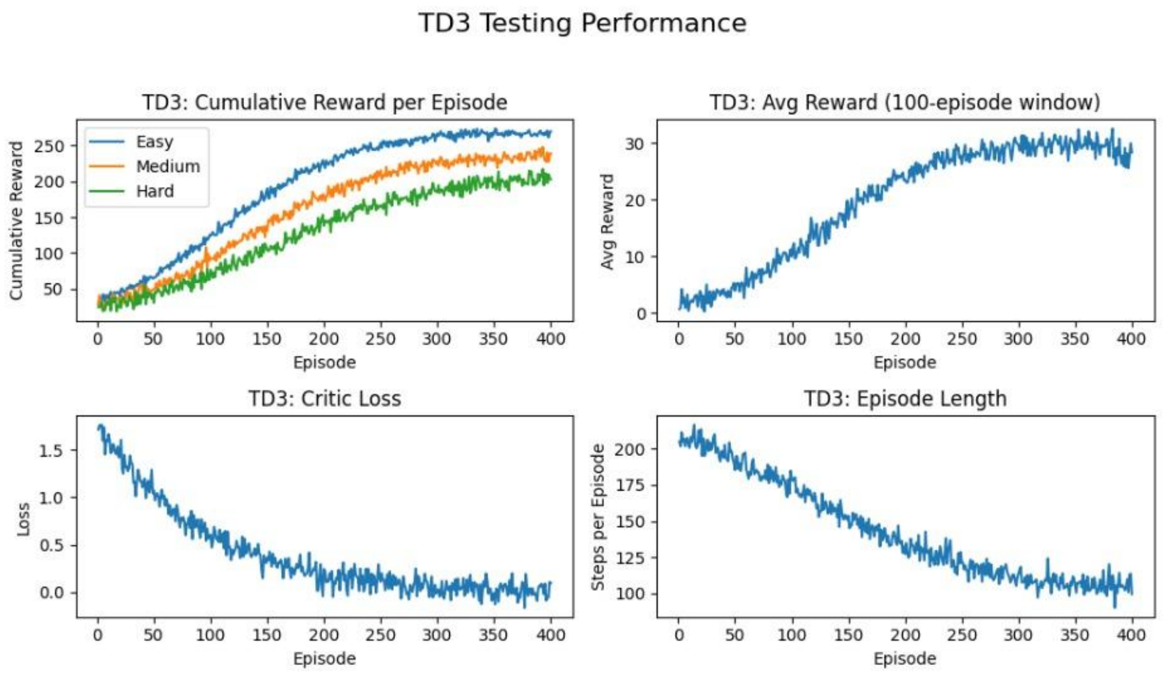

Figure 4 : TD3 Testing Performance

1 TD3 cumulative reward per episode

Cumulative reward increases strongly and saturates higher than DQN across all difficulty levels. Easy saturates around 260 to 280, medium around 230 to 250, and hard around 200 to 220. This suggests TD3 learns smoother and more effective control, especially in harder scenarios where continuous actions help produce better motion decisions.

2 TD3 average reward 100 episode window

The average reward increases from near 0 to about 28 to 31 and then stabilizes for the later episodes. The higher final level compared to DQN indicates better overall policy quality and more consistent goal reaching with fewer penalties.

3 TD3 critic loss

Critic loss decreases from around 1.6 to near 0.05 to 0.1 as training progresses. This shows the critic networks are learning a stable value estimate, and the reduction also reflects TD3 stability features like twin critics and target smoothing.

4 TD3 episode length

Episode length falls from roughly 210 steps to about 100 to 115 steps. This means TD3 reaches the goal in fewer steps and produces more efficient trajectories, which matches its higher reward trend.

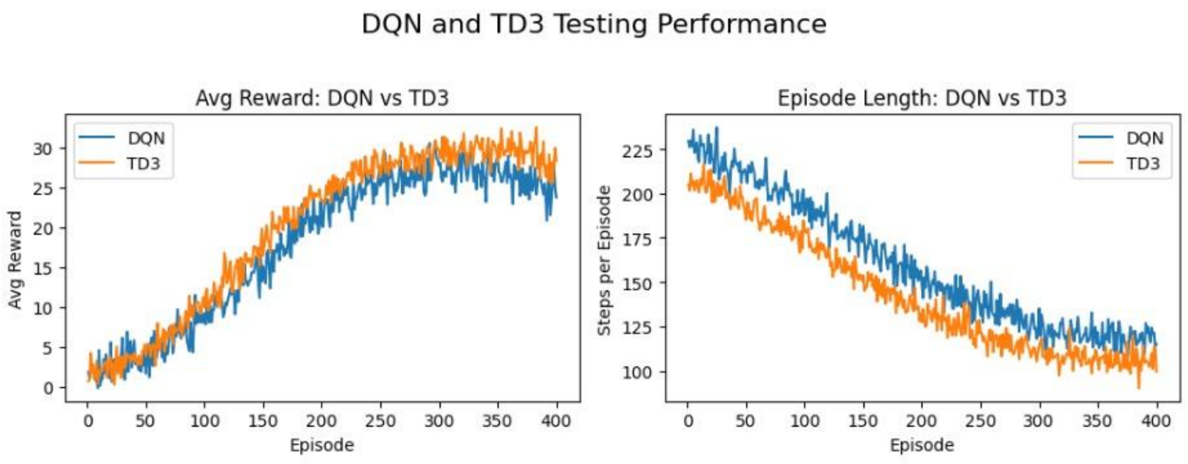

Figure 5 : DQN and TD3 Testing Performance

1 Average reward DQN vs TD3

Both methods improve with training, but TD3 stays slightly above DQN after the mid training region and finishes higher, around 30 compared to DQN around 26 to 28. This indicates TD3 achieves better overall navigation performance, likely because continuous control gives finer steering and speed adjustments, reducing collisions and unnecessary detours.

2 Episode length DQN vs TD3

Both episode lengths decrease, but TD3 remains consistently lower than DQN by roughly 10 to 20 steps through most of training. Lower episode length with higher reward means TD3 not only succeeds more, but also reaches the goal more efficiently and with smoother behavior.

5.2 Summary

Overall, the results show that both DQN and TD3 successfully learn navigation behavior over training episodes, indicated by a steady rise in cumulative and average rewards and a clear reduction in episode length. For DQN, rewards increase and then stabilize, while TD error decreases strongly toward near zero, confirming that the value estimates become stable and the policy converges; at the same time, episode length drops significantly, meaning the robot reaches goals faster with fewer unnecessary moves. TD3 shows even stronger improvement: cumulative rewards saturate at higher values across easy, medium, and hard scenarios, critic loss decreases steadily to a low level, and episode length falls to a lower final range than DQN, indicating more efficient trajectories. The direct comparison confirms TD3 performs better overall, achieving slightly higher average reward and consistently shorter episode length, which suggests smoother and more effective control, especially in harder environments where continuous actions help avoid penalties and reduce detours.

Chapter 6 : Analysis & Discussion

The simulation results demonstrate the effectiveness of using **topological mapping combined with reinforcement-learning algorithms** for autonomous robot navigation. In this approach, the environment is represented as a network of interconnected nodes rather than a dense grid, enabling efficient path planning, reduced computational

overhead, and faster decision-making in dynamic spaces. Both **DQN** and **TD3** were implemented to enable the robot to learn optimal paths toward a destination while avoiding obstacles. The reward curves across Easy, Medium, and Hard environments show consistent improvement, indicating successful learning of navigation strategies. TD3 exhibits slightly superior performance, reflected by higher average rewards and shorter episode lengths due to its twin-critic structure and reduced overestimation bias.

The decreasing TD error and critic loss further confirm stable convergence of both algorithms. As episodes progress, the robot requires fewer steps to reach the target, proving effective learning of shorter and safer routes within the topological map. This method offers significant benefits, including scalability to large environments, adaptability to continuous control, and enhanced robustness in obstacle-dense scenarios. In future applications, topological mapping combined with advanced RL techniques can support autonomous vehicles, warehouse robots, and service robots by enabling reliable real-time navigation, efficient path planning, and improved performance in complex, unstructured environments.

In future applications, such systems can be used in autonomous vehicles, smart delivery robots, search-and-rescue platforms, and industrial automation, offering robust real-time navigation, safer path planning, and efficient obstacle avoidance in complex real-world settings.

Chapter 7: Conclusion & Future Scope

Conclusion

The integration of topological mapping with reinforcement learning algorithms such as DQN and TD3 provides a practical and efficient solution for autonomous navigation in unknown and dynamic environments. By representing the environment as a connectivity graph of meaningful nodes and edges, the system avoids the heavy memory and computation requirements of dense grid maps and enables faster global decision making. At the same time, the RL controller learns motion behavior directly from sensor driven state information, allowing the robot to react to obstacles in real time and improve its navigation strategy through training.

The simulation results confirm that both DQN and TD3 successfully learn to generate safer and more efficient paths over time. Increasing cumulative and average rewards indicate improved goal reaching performance and reduced collision or penalty events, while decreasing TD error and critic loss show that value estimation becomes stable as learning progresses. The consistent reduction in episode length further demonstrates that the robot reaches targets using shorter and more direct trajectories, which reflects better planning and smoother control. When comparing the two methods, TD3 achieves slightly higher final rewards and consistently shorter episode lengths, indicating stronger stability and better control efficiency, which is expected since continuous actions allow finer steering and speed adjustments in complex spaces.

Overall, this study validates that combining topological representations with advanced RL methods enhances navigation accuracy, learning efficiency, and robustness. The framework is well suited for real world scenarios such as service robots, warehouse automation, delivery robots, and autonomous vehicles operating in changing environments, where the ability to replan quickly, avoid moving obstacles, and maintain reliable performance is essential.

Future Scope

1- Autonomous driverless electric vehicles can use topological maps to plan efficient routes through complex road networks, parking lots, campuses, and industrial zones with lower memory and faster decision making than dense grid maps.

2- Topological planning can support robust rerouting in real time for EVs by updating graph connectivity when roads are blocked, traffic patterns change, or charging stations become unavailable.

3- It can improve energy efficient driving by choosing routes based on connectivity and cost on edges such as slope, stop frequency, and congestion, which directly affects EV battery usage.

4- In last mile delivery robots and autonomous shuttles, topological maps enable scalable navigation across large areas like malls, hospitals, airports, and smart cities without storing heavy metric maps everywhere.

5- In warehouses and factories, topological planning can coordinate fleets of AGVs and AMRs by using graph nodes as junctions and edges as lanes, making multi robot routing and traffic management simpler.

6- For drones and UAVs, topological path planning can be extended to multi layer graphs for altitude corridors and safe zones, helping navigation in urban canyons and constrained airspaces.

7- In disaster response and search and rescue, topological maps allow rapid exploration and navigation in partially known, changing environments where full metric mapping is slow or unreliable.

8- In home and service robotics, it supports quick learning of room connectivity and doorway transitions, making long term navigation more stable even when furniture moves.

9- In game technology, topological graphs match how many games already represent navigation using waypoints and nav graphs, enabling faster NPC pathfinding and realistic behaviour in large dynamic maps.

10- For procedural game worlds, topological planning helps generate and traverse new regions on the fly by connecting newly created nodes and edges without rebuilding the whole navigation mesh.

11- In AR and VR, topological mapping can support lightweight spatial understanding for user movement guidance and persistent scene navigation with limited compute on wearable devices.

References

1. T. George et al., “A Topological Approach of Path Planning for Autonomous Robot Navigation in Dynamic Environments,” IEEE, 2009.
2. S. Thrun, “Probabilistic Robotics,” MIT Press.
3. Roland Siegwart, Autonomous Mobile Robots.
4. Dijkstra, “A Note on Two Problems in Connection with Graphs.”
5. Robotics and Autonomous Systems Journal.
6. Kuipers, B. (2000). *The Spatial Semantic Hierarchy*. Artificial Intelligence, 119(1–2), 191–233.
7. Fox, D., Burgard, W., & Thrun, S. (1997). *The dynamic window approach to collision avoidance*. IEEE Robotics & Automation Magazine, 4(1), 23–33.
8. LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.
9. Mnih, V., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518, 529–533.
10. Fujimoto, S., Hoof, H., & Meger, D. (2018). *Addressing Function Approximation Error in Actor–Critic Methods*. Proceedings of the 35th International Conference on Machine Learning (ICML).
11. Khatib, O. (1985). *Real-time obstacle avoidance for manipulators and mobile robots*. The International Journal of Robotics Research, 5(1), 90–98.
12. Borenstein, J., Everett, H. R., & Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques*. A K Peters.
13. Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press.
14. Kuipers, B., & Byun, Y. T. (1991). *A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations*. Robotics and Autonomous Systems, 8(1–2), 47–63.
15. Grisetti, G., Kümmerle, R., Stachniss, C., & Burgard, W. (2010). *A tutorial on graph-based SLAM*. IEEE Intelligent Transportation Systems Magazine, 2(4), 31–43.
16. Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT Press.
17. Watkins, C. J., & Dayan, P. (1992). *Q-learning*. Machine Learning, 8(3–4), 279–292.
18. Kober, J., Bagnell, J. A., & Peters, J. (2013). *Reinforcement Learning in Robotics: A Survey*. The International Journal of Robotics Research, 32(11), 1238–1274.
19. Smith, R., Self, M., & Cheeseman, P. (1990). *Estimating uncertain spatial relationships in robotics*. Autonomous Robot Vehicles, Springer.
20. Brooks, R. A. (1983). *Model-based exploration for mobile robot navigation*. International Joint Conference on Artificial Intelligence (IJCAI).
21. Leonard, J. J., & Durrant-Whyte, H. F. (1991). *Simultaneous map building and localization for an autonomous mobile robot*. IEEE/RSJ International Workshop on Intelligent Robots and Systems.
22. Lozano-Pérez, T., & Wesley, M. A. (1979). *An algorithm for planning collision-free paths among polyhedral obstacles*. Communications of the ACM, 22(10), 560–570.
23. https://drive.google.com/file/d/1t46CEPEgwn6Z4Hjmk96uQtq1iaQxiNxm/view?usp=drivesdk