Autonomous Navigation System for Swarm Robots in Dynamic Environments

1 Introduction

As industrial technology and artificial intelligence have advanced, robot swarms have greatly improved working conditions and lessened the strain on human labour, making them appropriate for a variety of job settings. Robot swarms have drawn a lot of attention because of their agility, maneuverability, and capacity to be used in a variety of challenging tasks, especially when it comes to autonomous navigation in warehouses. Robot swarms demonstrate the desired collective behavior based on the concept of self-organization, from local interactions with multiple agents through coordination and cooperation.

One of the essential technologies for creating smart robot swarms is path planning. Robots may use their sensors to gauge their surroundings and a variety of decision-making processes to create a sensible, workable, and secure path that will bring them to their destination without running into obstacles. Traditional path planning methods in robotics require precise localization and complete maps, limiting their adaptability in dynamic environments. Dynamic environments introduce complexities best addressed through adaptive learning-based methods and Reinforcement learning (RL), especially deep reinforcement learning (DRL), has emerged as a key strategy for this.

Reinforcement learning (RL) is recognized as a popular automatic design method, which focuses on learning collective navigation policies—which must allow agents in a homogeneous swarm to cooperatively explore an unfamiliar, dynamic, and changing environment while avoiding collisions with stationary and moving obstacles. Key capabilities of RL include collision-free navigation and scaling to an arbitrarily large number of agents. Deep reinforcement learning methods include Deep QNetwork (DQN), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC). Several variants and integrations of these approaches have been developed to improve computational efficiency, navigation in narrow channels, stability of search strategies, adaptability, and convergence speed.

2 Problem Statement

Indoor autonomous navigation poses unique challenges due to the absence of GPS and the reliance on manual setups like ArUco markers for navigation and pose estimation. This project aims to develop a scalable and adaptable navigation system for Autonomous Mobile Robots (AMRs) to operate in dynamic environments where obstacles move unpredictably and layouts frequently change. The solution will focus on collaborative path planning in multi-agent systems where each robot is assigned a unique target, while also focusing equally on real-time obstacle detection and avoidance, and memory persistence for continuous learning. Designed for deployment on Bharat Forge's ground robots, this system will enable optimized autonomous maintenance and inspection tasks across diverse industrial settings while ensuring scalability, efficiency, and adaptability to new environments.

3 Methodology

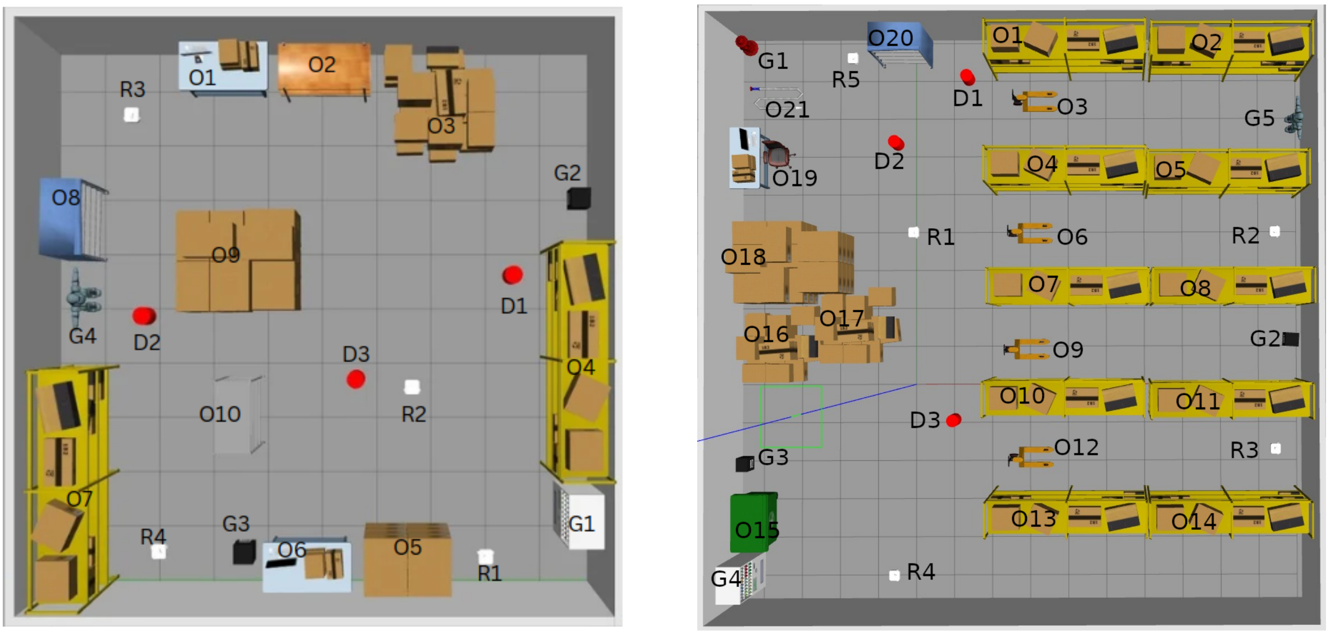
3.1 Solution Approach

The simulation framework developed in Gazebo and integrated with ROS 2 mainly evaluates the reinforcement learning (RL) algorithms in changing multi-agent environments. The setup has been tested in two distinct environments:

- $10m \times 10m$ layout with 4 robots, 10 static and 3 dynamic obstacles (Figure 1a);
- $15m \times 15m$ layout with 5 robots and the same number of dynamic obstacles (Figure 1b).

The environments feature static and dynamic elements, reflecting realistic warehouse settings. The framework adopts a modular approach, wherein each robot has an independent target, as opposed to the conventional leader-follower approach, thus allowing the number of robots to be scaled as needed, with configuration handled via a YAML file.
Each robot carries a 2D LiDAR sensor used for obstacle detection and a RealSense RGB-D camera as an extra sensor input. This LiDAR works with a maximum range of about 3.5 m and takes about 360 samples per scan. The odometry data are obtained by fusing onboard sensors and sent directly to the RL algorithm that steers the incremental robot

toward various goals while avoiding obstacles along the journey. It trains this model built on RL through 8000 episodes, and rewards are provided whenever targets are achieved, while penalties are also incurred due to collisions.



(a) $10m \times 10m$ Environment Containing 10 Static and 3 Dynamic Obstacles

(b) Another Larger $15m \times 15m$ Environment Containing 21 Static Obstacles

Figure 1: Environments in Gazebo
Keys: O (Static Obstacles), D (Dynamic Obstacles), G (Goals) and R (Robots)

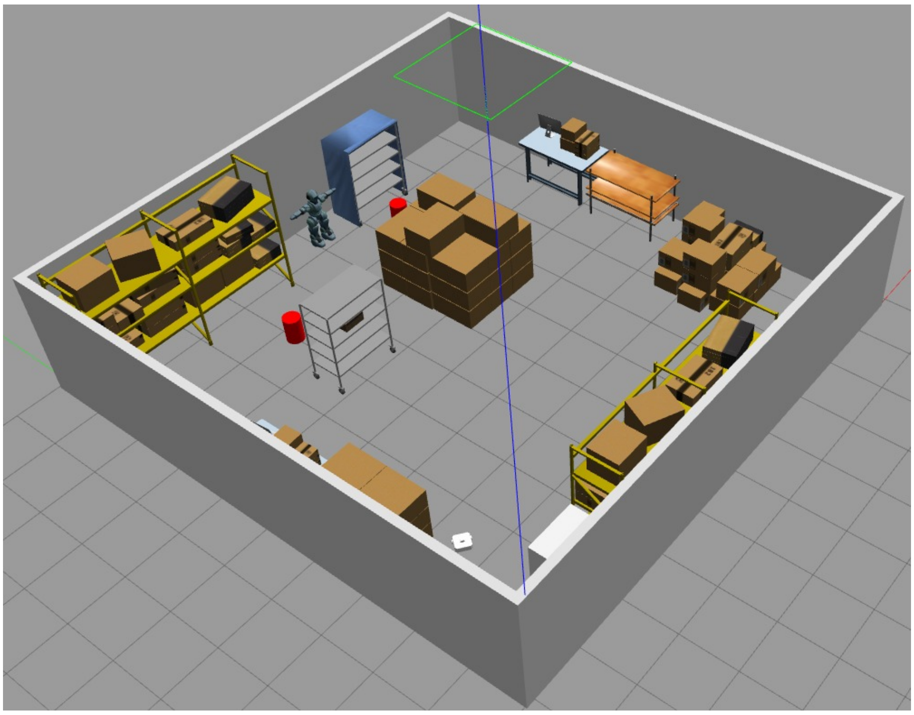


Figure 2: Isometric View of the Environment

3.2 System Architecture

The system (Figure 3) integrates:

- **Gazebo** for simulation of realistic environments.
- **ROS 2 Humble Hawskbill** for managing inter-module communication through topics and services.

- **RViz** and **Pygame** for real-time visualization of robot movement, obstacle detection, and system performance.

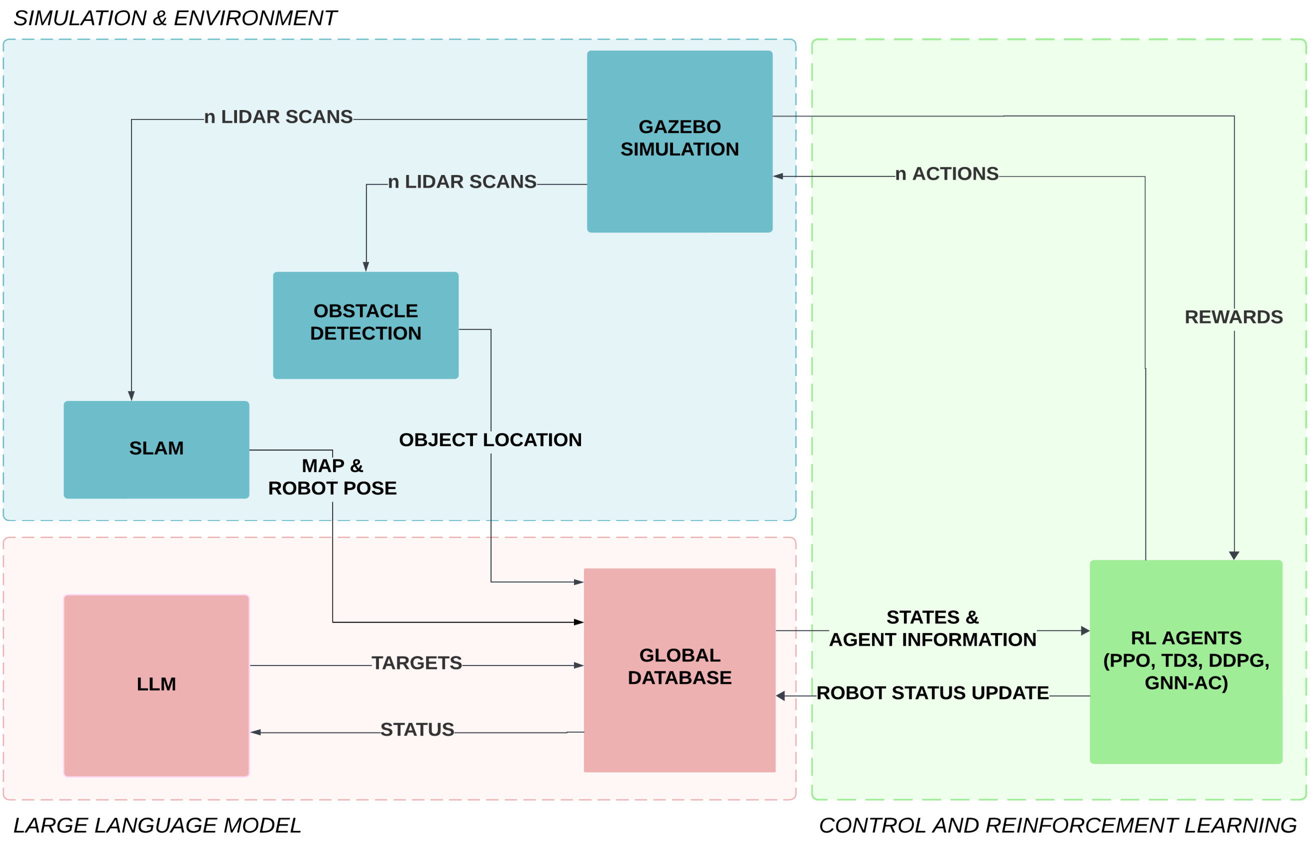


Figure 3: Illustration of Process Flowchart

3.2.1 Environment and Swarm Configuration

The simulation environments consist of:

- **Static obstacles:** Walls, furniture, and fixed installations.
- **Dynamic obstacles:** Other robots and moving elements.
- **Objects of Interest:** Objects to be targeted by the robots. Vending Machine, Charging Station, etc

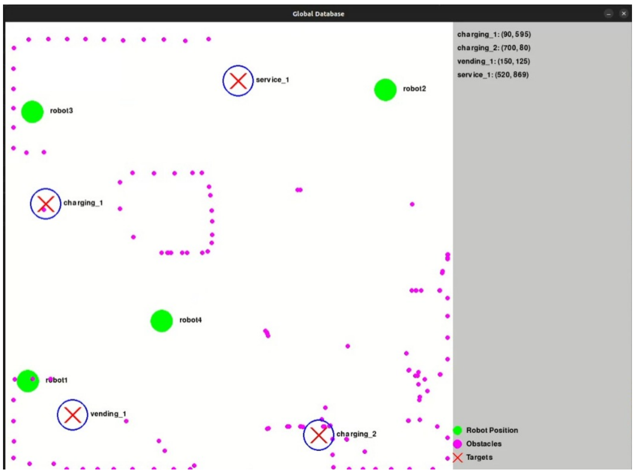
Robots are deployed at random positions in the environment, and their initial configurations are defined in a YAML file.

3.2.2 Obstacle Detection

LiDAR data from each robot is processed in real-time to identify both static and dynamic obstacles. The system employs a **decay mechanism** to distinguish between stationary and moving objects. The detected obstacles are transformed into a global coordinate frame and clustered using **DBSCAN** algorithm to minimize noise and ensure accurate localization. Each obstacle is recorded once per robot, and redundant detections are filtered out to maintain an optimal representation of the environment. The global obstacle map is updated constantly and used for navigation and task allocation.
Visualizations of the obstacle maps are provided through Pygame (Figures 4 and 5), displaying the robot trajectories and detected obstacle positions in real-time.

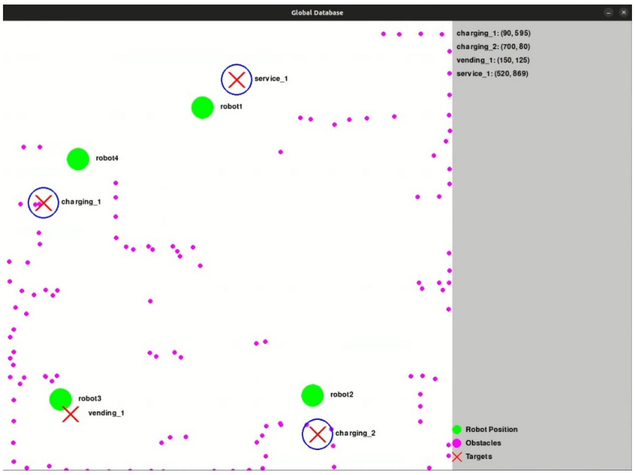
3.2.3 Simultaneous Localization and Mapping (SLAM)

SLAM Toolbox processes 2D LiDAR data from all robots to continuously update a global map (Figure 7) of the environment. Each robot contributes to this global map by merging its local observations (Figures 6a, 6b, 6c and 6d) with the global database, ensuring memory persistence and providing a comprehensive, up-to-date representation of the environment.



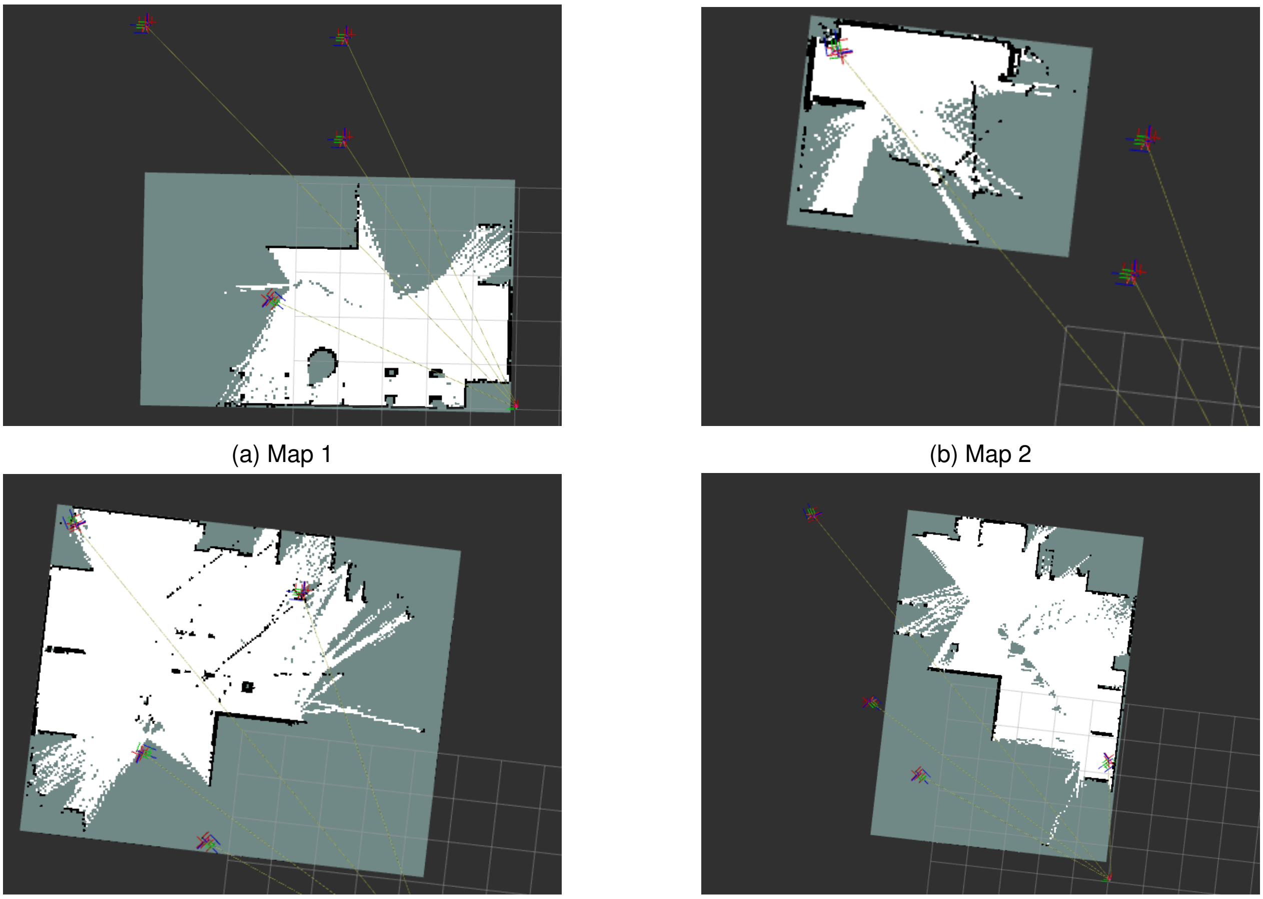
charging_1: (90, 595)
charging_2: (700, 80)
vending_1: (150, 125)
service_1: (520, 869)

Figure 4: Pygame Visualization of the Obstacle Map Initially



charging_1: (90, 595)
charging_2: (700, 80)
vending_1: (150, 125)
service_1: (520, 869)

Figure 5: Pygame Visualization of the Obstacle Map while Navigating



(a) Map 1

(b) Map 2

(c) Map 3

(d) Map 4

Figure 6: Individual Maps of the Bots

3.3 Reinforcement Learning Algorithm

Reinforcement learning (RL) operates without the need for pre-labeled data, unlike supervised and unsupervised learning. Instead, it relies on the interaction between an agent (the decision-maker) and the environment (real-world or simulated settings). At each step, the agent observes the current state of the environment and takes actions guided by a policy—a strategy that maps states to actions. Based on the action taken, the environment responds by transitioning to a new state and providing a reward that reflects the quality of the action. The goal is for the agent to learn a policy that maximizes long-term cumulative rewards through iterative trial and error. This process continues until the agent converges on an optimal strategy.

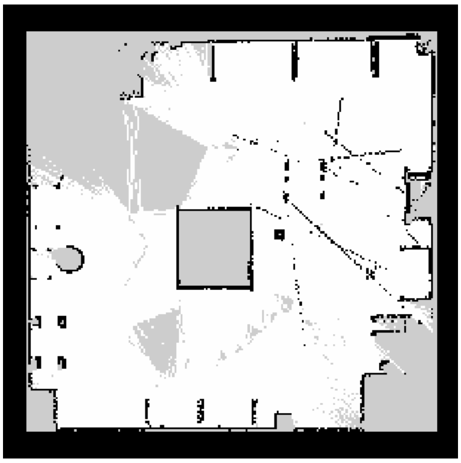


Figure 7: Combined Global Map after Merging Individual Maps

3.3.1 Comparison of Reinforcement Learning Algorithms for Path Optimization and Swarm Coordination

After conducting a series of assessments and evaluations, we identified the following five Reinforcement Learning algorithms as the most effective for navigation and swarm coordination. These algorithms—Twin Delayed Deep Deterministic Policy Gradient (TD3), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), GNN-Actor-Critic(Graph Neural Network-Actor-Critic) and Counterfactual Multi-Agent Policy Gradients (COMA)—were selected based on their performance across key parameters such as scalability, convergence speed, stability, sample efficiency, suitability for swarm behavior, and computational cost. The table below (1) summarizes the comparative performance of these algorithms:

Algorithm	**Scalability**	**Convergence Speed**	**Stability**	**Sample Efficiency**	**Suitability for Swarm Behavior**	**Computational Cost**
TD3	Moderate	High	High	Moderate	Moderate	Moderate
DDPG	Low	Low	Low	Low	Moderate	Low
PPO	High	Moderate	Very High	High	High	High
GNN-Actor-Critic	Very High	Moderate	High	High	Very High	High

Table 1: Comparison of TD3, DDPG, PPO, and GNN-Actor-Critic Algorithms Across Key Parameters

3.4 Main Framework

The navigation task (Figure 8) is formulated as a **Markov Decision Process (MDP)** represented by the tuple (S, A, S', P, R), where each element holds its standard interpretation.

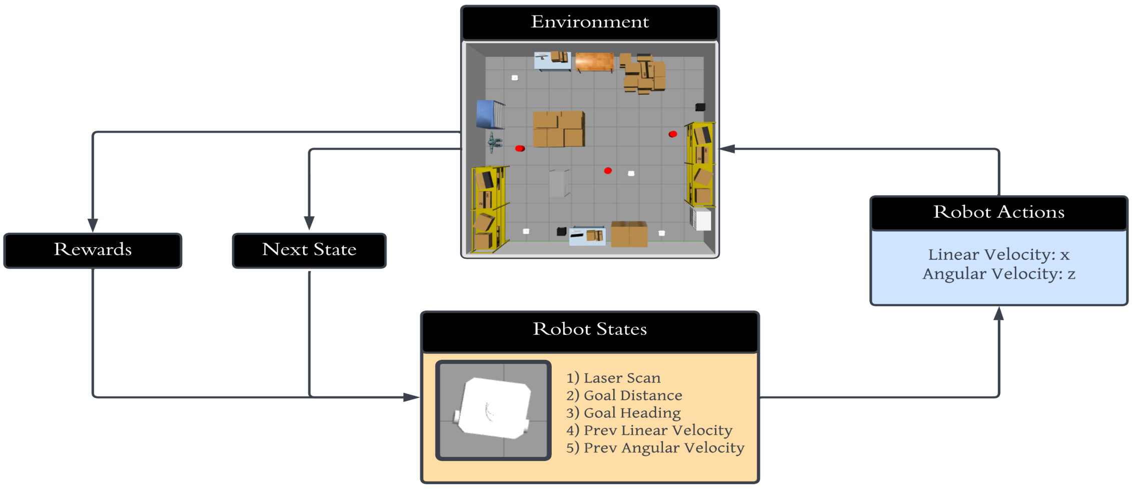


Figure 8: MDP Navigation Task

State Space: The components of the agent's state space are as follows:

- **Normalized Laser Scan Values**: Measurements from the LiDAR, scaled to a normalized range.
- **Normalized Goal Distance to Arena Size Ratio**: The distance to the target goal, normalized relative to the size of the navigation area.
- **Normalized Goal Heading Angle**: The angle between the agent's orientation and the direction to the goal, scaled to a normalized range.
- **Previous Linear Velocity**: The linear velocity of the agent during the prior time step.
- **Previous Angular Velocity**: The angular velocity of the agent during the prior time step.

Action Space: The action space consists of the agent's current linear and angular velocities. Linear velocity is bounded between $[-1, 1]$, while angular velocity ranges from $[-2, 2]$. These constraints are imposed to ensure safe navigation and to minimize the risk of the agent tumbling.

Reward Function: The reward function is carefully formulated to accelerate the convergence of reinforcement learning algorithms while maintaining safe and efficient navigation within a dynamic environment. The components of the reward structure are as follows:

- **Goal Alignment Penalty**: A penalty applied for deviations in the heading angle from the optimal trajectory toward the goal.
- **Excessive Angular Velocity Penalty**: Penalizes the agent for executing abrupt or excessively large angular velocity actions.
- **Proximity-to-Goal Reward**: A reward inversely proportional to the Euclidean distance from the agent to the goal, encouraging progress toward the target.
- **Obstacle Proximity Penalty**: A penalty incurred when the agent approaches obstacles, discouraging unsafe trajectories.
- **Excessive Linear Velocity Penalty**: Penalizes the agent for employing large linear velocities that may compromise safety or stability.

RL Algorithm: The primary RL navigation framework is implemented using the **Twin Delayed Deep Deterministic Policy Gradient** (TD3) algorithm. TD3 is an advanced off-policy RL algorithm that employs deep neural networks to approximate both the policy and value functions. The key reasons for selecting TD3 over alternative baseline algorithms are outlined below:

- **TD3 v/s DDPG:** Training the dynamic navigation environment with DDPG revealed a significant overestimation of Q-values, ultimately leading to policy degradation. TD3 addresses this issue through clipped double Q-learning, delayed updates to target and policy networks, and target policy smoothing, thereby improving stability and performance.
- **TD3 v/s PPO:** While PPO is a popular on-policy algorithm known for its ease of tuning across different environments, it is computationally intensive for large-scale navigation tasks or swarm scenarios. TD3's off-policy nature makes it more computationally efficient for such use cases.
- **TD3 v/s SAC:** Both TD3 and SAC have the best data efficiency. However, SAC trains a stochastic policy, which does not guarantee convergence to a single optimal solution. TD3, on the other hand, is deterministic and converges to the absolute optimal policy, which in the case of navigation task is the shortest path while avoiding obstacles.

Methodology: We intend to implement the **Decentralized Training and Decentralized Execution (DTDE)** approach. Initially, a single agent (robot) is trained within a dynamic environment. The trained networks are then deployed across multiple instances, corresponding to the number of robots. Goals are sampled randomly within the environment, and each agent is trained to navigate toward the goal by following the shortest and safest path.

3.5 Bonus Task: LLM Chatbot

In this, we utilized the Llama 3.1 (llama-3.1-70b-versatile) model, due to its generally high performance in function calling. A Groq API was used to effectively integrate this model. A user interface (UI) was developed using Streamlit, providing an accessible platform for user interactions.

3.5.1 Tool Calling Architecture

The system uses the data stored in a **JSON file**, which contains the coordinates of each robot and target. The JSON file also has the status of the targets assigned to each robot, along with handling whether the robots are idle or assigned to a target. This JSON file is also used by the **Pygame node** for real-time updates of robot coordinates. A central database is maintained within the JSON file to store the coordinates of all robots and targets, the status of each robot (idle or engaged), the status of each object (assigned or unassigned), and the status of the task (whether the objective is achieved, [”unknown”, ”success” and ”NA”]) when an object is assigned to a robot.
We also created 4 custom functions to help the LLM achieve robot-to-target assignments successfully:

1. To identify the nearest object of a specific type for a given robot.
2. To assign the nearest robot to the nearest object of a specified type.
3. To assign either:
   - The nearest robot to a specified object
   - A specific robot to a specified object
   - A specific robot to an object type.
4. To handle general queries such as robot statuses and to get information on how the robots are assigned to each object.

Example: User queries ”Assign vending machine to robot,” are interpreted using the Llama 3.1 model. The model outputs tool calls in dictionary format (For example, `['{"tool_name":"assign_general", "arguments":[{"argument_name":"object_type","argument_value":"vending"}]}']` is the output received). These outputs are parsed and mapped to corresponding implementations using a tool map. Once executed, the database is updated to reflect changes in robot status and task allocation.

3.5.2 Integration with the Environment

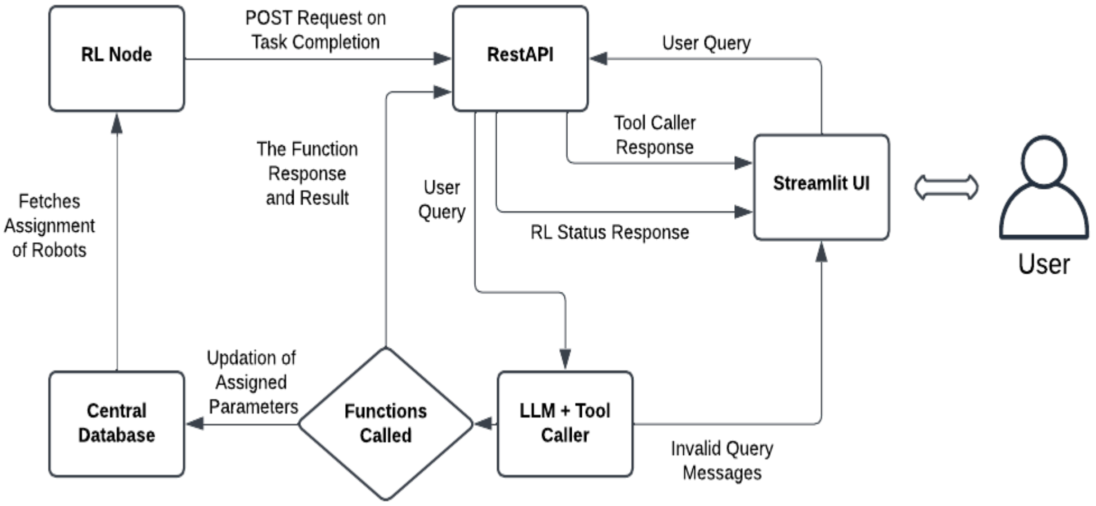


Figure 9: Integration of the Chatbot with the Algorithm

As shown in Figure 9 we have set up Rest API (Fast API) which is the access point to our tool caller. The chat UI is built in Streamlit, which sends a GET request to the API whenever the user sends any query. The response message from the tool caller is received and is displayed in the chat UI.
The statuses of robots and targets are recorded from the JSON file and are updated according to the user query when they are assigned accordingly. The RL node detects the changes in the main database and the movement of the robots is thus handled. On task completion, when each robot reaches its target, a POST request, with the robot ID and the target ID is sent to the API. This is also conveyed in the chat UI and the statuses of the robot and the target are also changed to reflect the completion of the task.

4 Evaluation Metrics

- **Obstacle Avoidance:** The robot swarm was systematically trained in the environment without any prior knowledge of its structure or layout. The swarm was designed to navigate toward the goal while avoiding both static and dynamic obstacles. This behavior was achieved by incorporating laser sensor data into the state representation and applying penalties for collisions during training.
- **Scalability:** A single agent was initially trained in a dynamically evolving environment, selecting random goal locations and learning the state-action visitation distributions as they converged. Post-training, it was scaled to multiple agents by running multiple independent instances, thereby ensuring decentralized training and execution.
- **Responsiveness:** The agent is trained to prioritize obstacle avoidance overreaching the goal location. This approach is essential to ensure safe reinforcement learning and navigation in unknown environments. To achieve this, a high weight is assigned to the collision penalty, in specific, an even higher weightage is provided to collision with dynamic obstacles.

5 Experiments and Results

From Figure 10, it is evident that the TD3 agent consistently receives positive rewards over the episodes, demonstrating steady progress. In contrast, the DDPG agent begins to overestimate and experiences a decline in performance, receiving diminishing returns after 700 episodes.

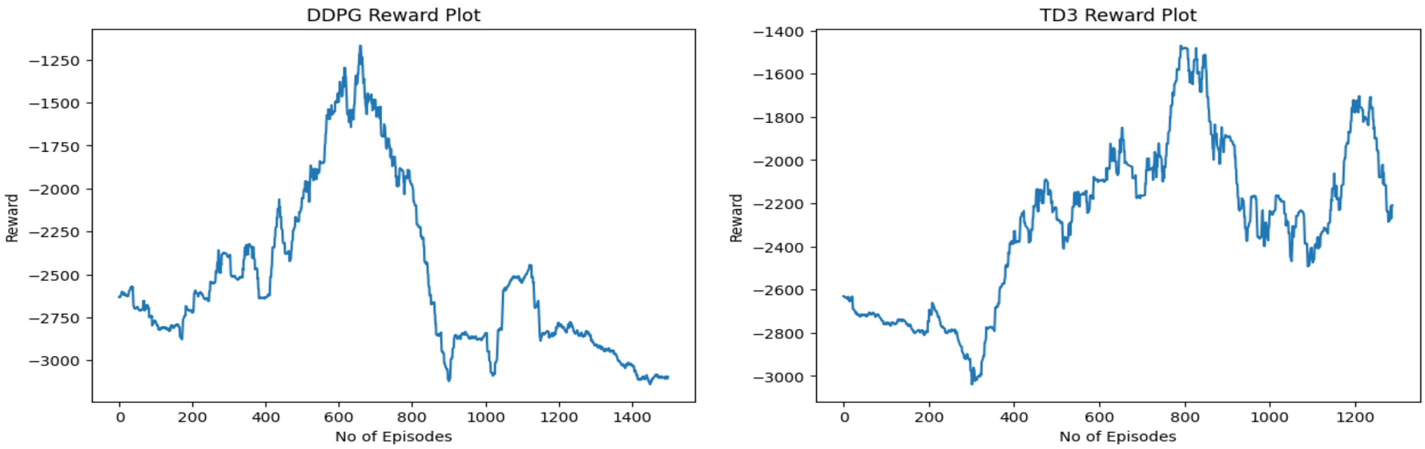


Figure 10: DDPG vs TD3 Reward Plot

6 Simulation Results

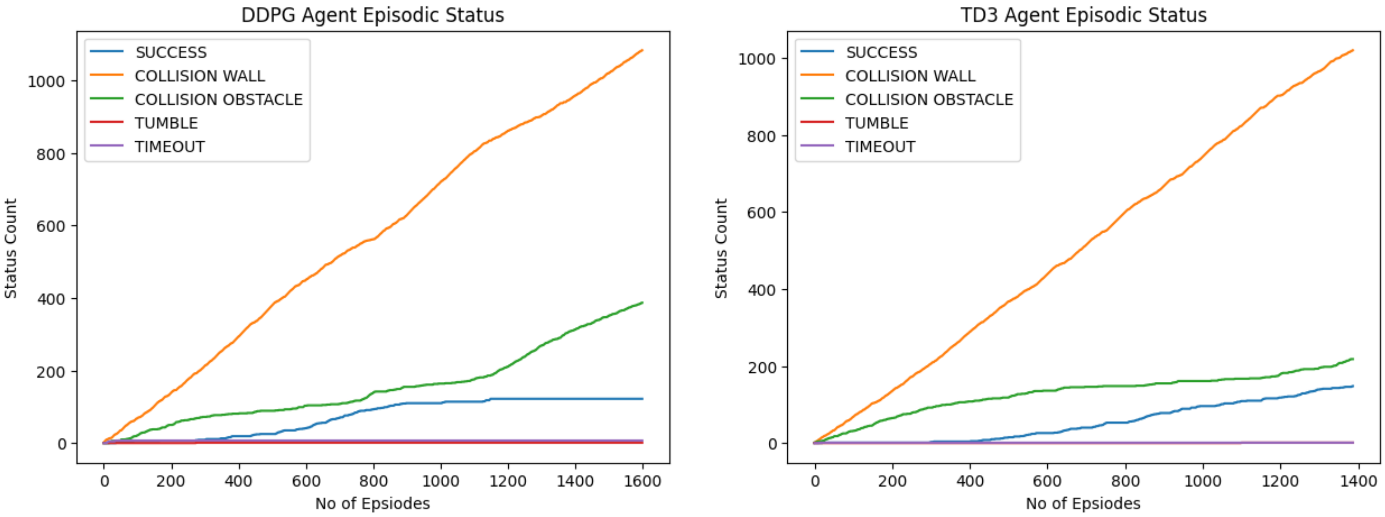


Figure 11: DDPG vs TD3 Epsiodic Status Plot

Figure 11 illustrates that after 1000 episodes, the DDPG agent exhibits a constant success count accompanied by a rising collision count. This behavior can be attributed to the overestimation of Q-values by the DDPG agent. In contrast, the TD3 agent shows a consistent increase in the success count and reduced collision count. It is expected to eventually solve the navigation task in subsequent episodes.

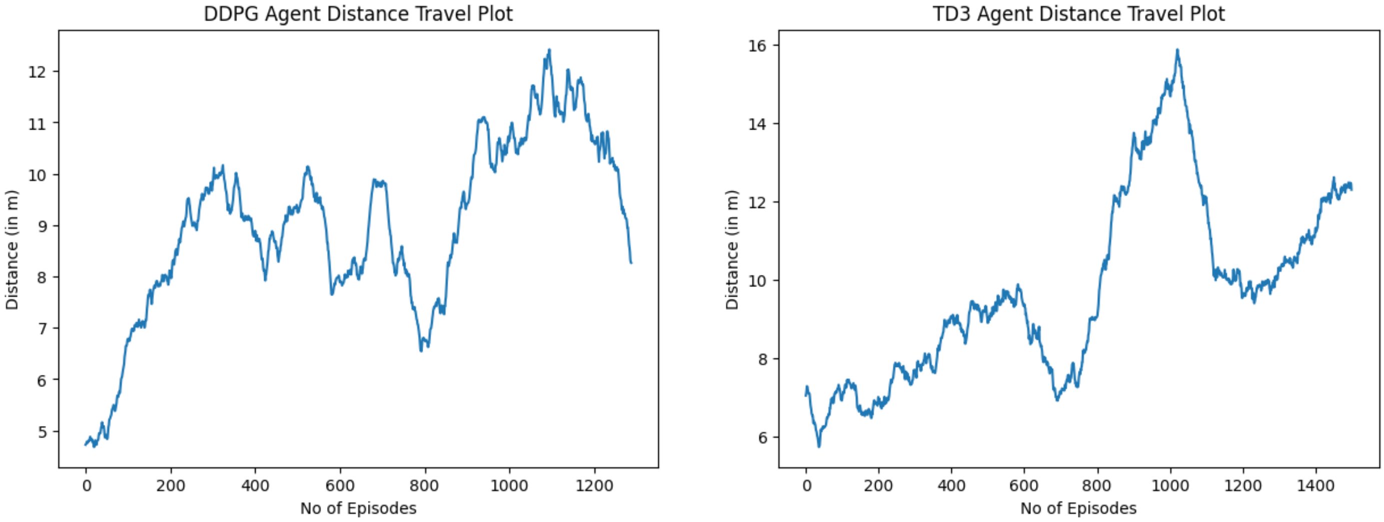


Figure 12: DDPG vs TD3 Agent Distance Travelled Plot

Figure 12 compares the distance traveled by agents trained using DDPG and TD3. From episodes 0 to 600, TD3 is shown to identify shorter paths to closer goal locations compared to DDPG. The sudden spike at episode 1000 occurs due to the selection of a distant goal location.

7 Future Pathway and Scalability Potential

The existing system exhibited promising scalability in larger environments and with increased swarm sizes. In environments larger than the initial 10 m × 10 m setup, such as the 15 m × 15 m layout, the system maintained efficient navigation and coordination. However, the computational load increased proportionally with the environment size, leading to a slight rise in collision rates under high-traffic conditions. When tested with larger swarms, the system successfully managed coordination and communication among multiple robots. The ROS 2 communication framework facilitated data exchange, but the overhead grew significantly with swarm size, affecting real-time performance. Future iterations could focus on optimizing communication protocols to minimize latency and improve data throughput. Moreover, the incorporated RL (DTDE) for the present system also offers considerable scalability for multi-agent systems, as the computational load is distributed across agents, making it suitable for large-scale systems. This decentralized approach minimizes communication overhead, allowing agents to be trained and deployed independently while enhancing adaptability and resilience in dynamic or partially observable environments. These inherent advantages were pivotal in its adoption as the core framework for this study.

Moreover, while TD3 and DDPG are efficient in environments with limited agent interactions and continuous action spaces, on-policy RL's like MAPPO, GNN-Actor-Critic and COMA, outperforms the former to address diverse challenges in dynamic multi-agent systems. MAPPO owing to its clipped surrogate objective mechanisms helps ensure stable and gradual policy updates, making it converge faster and scales well in continuous spaces. The GNN Actor-Critic framework stands out, representing episodes having high levels of inter-agent dependencies and interaction through its graph structure, dynamically adjusting the changing environment. COMA provides superior coordination in cooperative multi-agent environments by leveraging a centralized critic with counterfactual baseline advantages, which simplifies credit assignment among agents. By leveraging the Centralized Training with Decentralized Execution (CTDE) paradigm, these frameworks provide a more robust approach to navigation tasks compared to fully decentralized training and execution methodologies.

8 Conclusion

The simulation framework demonstrated depicts several strengths in navigating dynamic environments for multi-robot interaction. The system effectively avoided obstacles and showcased efficient adaptability to the changing environment.

It scaled efficiently across varying environment sizes and swarm configurations, maintaining consistent performance and coordination. Future work shall focus on several key factors like evaluation beyond simulated environments i.e., in real-world scenarios to enhance system performance and applicability. This transition will validate the robustness and adaptability of the framework in practical conditions in warehouses. Furthermore, MAPPO, GNN Actor-Critic and COMA models promise significant advancements in solving complex real-world problems involving dynamic, multi-agent environments. These algorithms have shown superior performance for dynamic environments in terms of navigation and stability.

However, certain limitations were observed. The system's real-time processing of sensor data and decision-making imposed significant computational overhead, which could become a bottleneck in more complex environments. Potential improvements should focus on enhancing system performance and scalability. Incorporating advanced sensors that balance reliability and efficiency could enhance obstacle detection accuracy and expand the perception range.

References

[1] Diego Arce, Jans Solano, and Cesar Beltrán. "A Comparison Study between Traditional and Deep-Reinforcement-Learning-Based Algorithms for Indoor Autonomous Navigation in Dynamic Scenarios". In: *Sensors* 23.24 (2023), p. 9672.

[2] Alexander Chernyavskiy, Alexey Skrynnik, and Aleksandr Panov. "Applying opponent and environment modelling in decentralised multi-agent reinforcement learning". In: *Cognitive Systems Research* (2024), p. 101306.

[3] Vo Duy Cong et al. "Path following and avoiding obstacle for mobile robot under dynamic environments using reinforcement learning". In: *Journal of Robotics and Control (JRC)* 4.2 (2023), pp. 157–164.

[4] Álvaro Díez and Fidel Aznar. "Deep Reinforcement Learning for Swarm Navigation Using Different Evolution Strategies". In: *Proceedings of the 2024 8th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*. 2024, pp. 14–19.

[5] Steve Macenski and Ivona Jambrecic. "SLAM Toolbox: SLAM for the dynamic world". In: *Journal of Open Source Software* 6.61 (2021), p. 2783.

[6] Siddharth Nayak et al. "Scalable Multi-Agent Reinforcement Learning through Intelligent Information Aggregation". In: (2023).

[7] Ramanjeet Singh, Jing Ren, and Xianke Lin. "A review of deep reinforcement learning algorithms for mobile robot path planning". In: *Vehicles* 5.4 (2023), pp. 1423–1451.

[8] Qingfeng Zhang et al. "Path planning of mobile robot in dynamic obstacle avoidance environment based on deep reinforcement learning". In: *IEEE Access* (2024).