Warsaw University of Technology

FACULTY OF
POWER AND AERONAUTICAL ENGINEERING

Institute of Aeronautics and Applied Mechanics

Master's diploma thesis

in the field of study Automatic Control and Robotics
and specialisation Robotics

Mobile Robot Navigation in Dynamic Environments

Ahmed Yesuf Nurye
student record book number 330148

thesis supervisor
dr hab. inż. Elżbieta Jarzębowska

Warsaw, 2024

To my parents.

Abstract

We present a framework for mobile robot navigation in dynamic environments using Deep Reinforcement Learning (DRL) and the Robot Operating System (ROS). Traditional navigation methods, which rely on complex, multi-module systems for mapping, localization, planning, and control, often lack the real-time adaptability required for unpredictable environments. To address this, we propose a streamlined approach that directly maps raw sensor inputs to control actions using the TD7 algorithm, an augmentation of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm with state-action embeddings. These embeddings are crucial for predicting the next state of the environment, enabling the system to better model environmental dynamics and significantly improving navigation performance in dynamic settings.

Extensive simulations were conducted in three distinct environments with varying levels of complexity—ranging from simple obstacle-free spaces to scenarios with static obstacles and dynamic actors. The results demonstrate that our DRL-based approach consistently outperforms baseline methods, particularly in environments with higher complexity.

Keywords: Mobile Robot Navigation, Deep Reinforcement Learning, TD3, TD7, ROS, Gazebo.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Prof. Elżbieta Jarzębowska, for her invaluable guidance, support, and encouragement throughout this research. Her expertise and insights have been crucial in shaping this work. I also extend my sincere thanks to the professors and instructors who have contributed to my academic growth during my studies.

My heartfelt appreciation goes to my family and friends for their unwavering support throughout this journey. Your belief in me has been a constant source of strength. I am also deeply thankful to the Polish National Agency for Academic Exchange (NAWA) for providing the scholarship that made this research possible.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

Chapter 1

Introduction

The ability of robots to navigate autonomously in environments characterized by dynamic elements, such as moving obstacles and evolving terrains, is crucial for the practical deployment of robotic systems in real-world scenarios [5, 84]. Autonomous navigation in such dynamic environments is a cornerstone of modern robotics, with applications spanning diverse areas, including exploration, search and rescue missions, logistics and warehousing, and service robots in homes and offices [68, 69].

This thesis addresses the challenge of autonomous navigation in dynamic environments by formulating it as a reinforcement learning (RL) problem. We propose a novel framework for mobile robot navigation that leverages a deterministic policy gradient algorithm, building upon the Twin Delayed Deep Deterministic (TD3) policy gradient algorithm [21] and its enhanced version, which incorporates state-action embeddings, referred to as the TD7 algorithm [24]. The resulting framework functions as a map-less motion planner, which processes a 24-dimensional input representing the shortest range findings, executed linear and angular velocity commands, and goal-relative information—specifically, the relative heading and distance to the target goal—and outputs continuous linear and angular velocity commands. The motion planner is trained end-to-end, enabling it to directly map a state to continuous velocity commands that the robot executes.

To validate the effectiveness of the proposed navigation framework, extensive testing and comparison against a baseline method are conducted in three distinct environments, each designed to emulate the varied challenges that a mobile robot

might encounter in real-world applications.

1.1 Motivation

Mobile robots have demonstrated remarkable efficiency in controlled environments, such as factories and warehouses, where they perform repetitive tasks with high precision [27, 28]. These successes, however, underscore a significant challenge when considering the deployment of robots in dynamic, real-world settings [1, 83]. Despite substantial advances in robotics, the widespread use of robots in homes, hospitals, and urban environments remains limited. The primary obstacle is the unpredictability and complexity of these real-world settings, where human behavior and rapidly changing conditions create a highly variable and uncertain landscape [59].

In such environments, robots must not only navigate around obstacles and adapt to evolving terrains but also interact seamlessly with humans, who often behave unpredictably. These tasks demand a level of adaptability and learning that exceeds the capabilities of traditional robotic systems. Therefore, developing robust navigation frameworks capable of handling uncertainty and variability in real-time is essential for the successful deployment of robots in dynamic settings. Motivated by these significant challenges, this thesis aims to contribute to the solutions that will enable the next generation of robots to achieve true autonomy, effectively bridging the gap between the controlled environments of factory floors and the unpredictability of the real world.

1.2 Thesis Objective

The primary objective of this thesis is to develop a DRL-based framework for mobile robot navigation in dynamic environments. By directly mapping raw sensor inputs to control actions through a single, unified model, this approach aims to streamline the navigation process and improve the robot's adaptability in complex, dynamic scenarios.

- Develop a DRL-based framework in ROS for mobile robot navigation, which directly maps raw sensor inputs to control actions.

- Address the limitations of traditional navigation methods by employing a unified model, simplifying the navigation process and reducing the system's complexity.
- Validate the proposed framework through comprehensive simulations in diverse environments, measuring performance against varying levels of complexity and obstacle densities.

1.3 Thesis Contribution

We propose a DRL based approach to mobile robot navigation in dynamic environments. Unlike traditional navigation frameworks, which depend on a series of complex and interdependent modules for mapping, localization, planning, and control, our approach simplifies this process by directly mapping raw sensor inputs to control actions through a unified model. This simplification not only reduces system complexity but also enhances adaptability to a wide range of real-world scenarios.

Traditional methods require extensive prior knowledge of the environment and involve multiple heavyweight modules that need careful integration and tuning, which can make them brittle and less effective in environments that change unpredictably or where real-time adaptability is crucial. In contrast, our DRL-based approach excels in such settings due to its inherent adaptability. By treating navigation as a reinforcement learning problem, the system learns optimal behaviors directly from interaction with the environment, without requiring prior maps or predefined rules. This makes the framework highly flexible and capable of handling diverse scenarios, such as exploration, delivery services, and robot companionship, without extensive reprogramming or reconfiguration. Additionally, we employ the TD7 algorithm, a state-of-the-art reinforcement learning method shown to outperform other algorithms in OpenAI Gym environments [24]. To the best of our knowledge, this is the first application of the TD7 algorithm to mobile robot navigation.

In summary, this work makes two key contributions: First, we introduce a DRL-based navigation framework for dynamic environments that does not require prior information. Second, we have made the implementation open-source[1].

[1] https://github.com/anurye/DRL-for-Mobile-Robot-Navigation-Using-ROS2.git

1.4 Thesis Outline

In this section, we present a brief summary of the organization of the thesis. Chapter 2 reviews existing research related to our problem, positioning our work within the current field. Chapter 3 introduces our methodology, starting with a brief recap of reinforcement learning, then detailing our simulation environment and system architecture. We explain how we frame navigation in dynamic environments as a reinforcement learning problem and describe our solution using deep reinforcement learning, specifically an actor-critic network based on TD7 algorithm. Chapter 4 presents our experimental results, discusses findings, and compares them with baseline method. Finally, Chapter 5 concludes the thesis with a summary of findings and suggestions for future research.

1.5 Tools Used for Implementation

This work utilizes a range of tools and dependencies, detailed comprehensively in the project documentation[2]. Below is a summary of the primary tools employed in the implementation:

Robot Operating System

The latest version of the Robot Operating System, *ROS2 Humble* [58], serves as the backbone of this work, providing the communication infrastructure between the simulation environment and the agent.

Gazebo

Gazebo [42] is employed to simulate the robotic agent and its environment, offering a high-fidelity, physics-based simulation platform.

[2] https://github.com/anurye/DRL-for-Mobile-Robot-Navigation-Using-ROS2/blob/main/README.md

Chapter 2

Related Works

Chapter organization: We begin with an overview of crowd navigation techniques in Section 2.1, reviewing both classical methods like the social force model and contemporary DRL based approaches. Next, we explore the fundamental components of navigation that are common across various methods. Section 2.2 reviews the state of the art in simultaneous localization and mapping (SLAM). In Section 2.3, we cover classical search-based and probabilistic algorithms for global path planning, as well as algorithms for local path planning. Finally, Section 2.4 focuses on research that employs DRL for mobile robot navigation.

2.1 Crowd Navigation Background

In this section, we present works that have been proposed for navigation in crowded environments. One of the earliest methods for describing pedestrian dynamics, which can also be applied to crowd navigation, is the social force model [32]. This model describes the motion of pedestrians as if they were subject to social forces, with various components including: acceleration towards the desired velocity, distance from other pedestrians and boundaries, and a term modeling attractive effects. The reciprocal velocity obstacles for real-time multi-agent navigation, shown in Figure 2.1, introduce a local reactive collision avoidance method that implicitly assumes other agents use similar collision avoidance strategies to generate safe and oscillation-free motions [86]. However, there is the freezing robot problem [83], which simple models

like the two mentioned above encounter, limiting their application in highly dynamic environments.

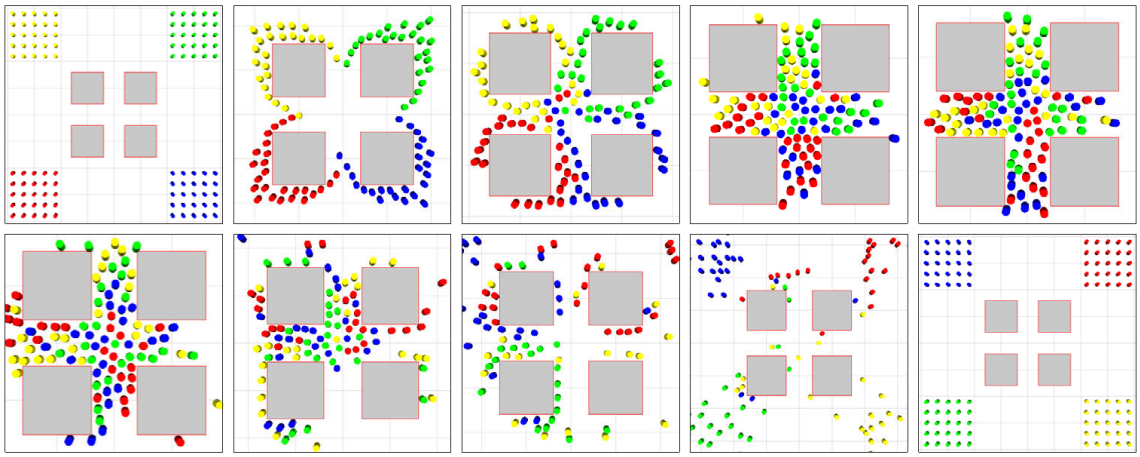


Figure 2.1: Four groups in opposite corners of the environment exchange positions in the Narrow Passage scenario, the zoom level varies between stills [86].

CADRL [14] introduces a decentralized multi-agent collision avoidance algorithm based on DRL, which addresses the limitations of existing algorithms in dynamic environments. The approach involves training a pair of agents to navigate around each other to learn a value network that encodes the expected time to goal and then generalizing this network in a principled way to handle multi-agent scenarios. Crowd-Robot Interaction [13] presents an approach to improve robot navigation in crowded environments by explicitly modeling interactions between humans and the robot as well as between humans themselves using a self-attention mechanism.

2.2 Simultaneous Localization and Mapping

SLAM is a fundamental process in robotics where a mobile robot simultaneously constructs a map of an unknown environment while deducing its own location within that map [18, 64]. The ability to perform these tasks concurrently is crucial for autonomous navigation, especially in environments where prior knowledge of the surroundings is unavailable or incomplete. SLAM algorithms are broadly categorized into two primary approaches [57]: Bayes-based filter methods and graph-based methods.

The earlier generation of SLAM algorithms predominantly relied on the Bayes-based filter approach, which utilizes probabilistic models to estimate the robot's location and map the environment iteratively [82]. This approach is rooted in the use of recursive state estimation techniques, such as the Extended Kalman Filter (EKF) and Particle Filter, to handle the inherent uncertainty in sensor measurements and robot motion. These algorithms represent the environment as a continuous probability distribution and update this distribution over time as the robot gathers more data. While these methods have been effective in various scenarios, they often struggle with scalability and computational efficiency, particularly in large, complex environments [57].

In contrast, more recent advancements in SLAM research have led to the development of graph-based methods, which have become the dominant approach in modern SLAM systems [57, 81]. Graph-based SLAM constructs a graph where nodes represent robot poses and landmarks, and edges represent spatial constraints derived from sensor observations. The core problem then becomes one of optimizing this graph to find the most likely configuration of poses and landmarks, given the observed constraints. This formulation allows for more efficient handling of large-scale environments and offers greater flexibility in incorporating various types of sensor data and constraints. Additionally, graph-based methods can effectively manage loop closures, situations where the robot revisits a previously mapped area, which is critical for reducing drift and improving the accuracy of the generated map.

Based on the primary sensor type used, SLAM implementation are typically classified into LiDAR SLAM and Visual SLAM [35, 91].

2.2.1 LiDAR SLAM

LiDAR SLAM algorithms rely on LiDAR sensors, which provide high-precision distance measurements through laser beams, making them particularly effective in environments where visual cues are insufficient.

Among the notable LiDAR SLAM approaches are GMapping, which uses a particle filter to build grid maps from 2D LiDAR data [29, 30], and HectorSLAM, which combines LiDAR scan matching with a 3D navigation filter based on EKF state estimation, suitable for high-speed mapping [44]. Graph-based methods like

Google Cartographer and KartoSLAM maintain a graph of robot poses and features, enabling efficient loop closure detection and global map optimization, particularly useful in large-scale environments [33, 46]. Another approach, TinySLAM, is designed for resource-constrained environments, offering a lightweight solution with reduced computational overhead [77].

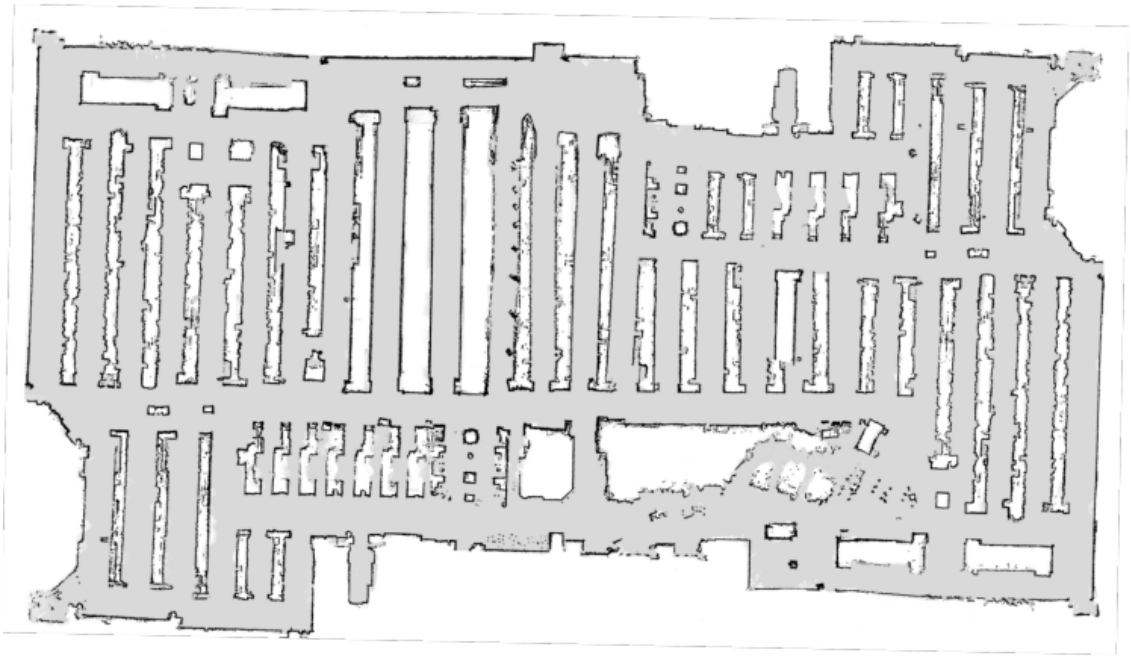


Figure 2.2: Retail store map created using SLAM Toolbox [57].

Figure 2.2 shows an example map of a retail store generated using SlamToolbox with data from a 2D-LiDAR sensor. Despite the ability to provide high precision data, LiDAR sensors are usually accompanied by hefty price tags making LiDAR SLAM an expensive solution [64].

2.2.2 Visual SLAM

Visual SLAM (vSLAM) uses images acquired from relatively inexpensive cameras and other imaging sensors enabling it to be implemented at a low cost. As a resut vSLAM has received an increasing attention over the past decades [37]. Unlike LiDAR-based SLAM, which relies on precise distance measurements, vSLAM extracts visual features

from the environment to perform localization and mapping. This approach allows for rich environmental perception, enabling detailed scene understanding and the potential for lower hardware costs, albeit with greater sensitivity to lighting conditions and texture variability.

One of the pioneering systems in this field is ORB-SLAM [63], a feature-based monocular SLAM system designed to operate in real-time across diverse environments. ORB-SLAM utilizes ORB (Oriented FAST and Rotated BRIEF) features consistently across all SLAM tasks—tracking, mapping, relocalization, and loop closing—thereby enhancing both efficiency and reliability. Building on the success of ORB-SLAM, ORB-SLAM2 [62] was developed to extend the system's capabilities to support stereo and RGB-D cameras. This extension addressed several inherent limitations of monocular SLAM, such as scale ambiguity and scale drift, by leveraging the additional depth information provided by these cameras. Further advancing the field, ORB-SLAM3 [12] was introduced as the first system capable of performing visual, visual-inertial, and multimap SLAM with monocular, stereo, and RGB-D cameras, using both pinhole and fisheye lens models. This system incorporated inertial measurements to improve the robustness and accuracy of the SLAM process, especially in scenarios with rapid camera motion or poor visual feature availability. The addition of multimap capabilities also allowed ORB-SLAM3 to handle long-term mapping tasks by managing multiple maps simultaneously, which is particularly useful in applications requiring map reuse or in environments that change over time.

While these systems demonstrated robust performance in static environments, real-world applications often involve dynamic elements that can degrade SLAM performance. To address this, DynaSLAM [7] was developed as an extension of ORB-SLAM2, specifically designed to handle dynamic environments. By leveraging Mask R-CNN for semantic segmentation, DynaSLAM can identify and segment dynamic objects, removing them from the SLAM pipeline to prevent them from affecting the map, Figure 2.3. However, the reliance on computationally intensive deep learning models like Mask R-CNN poses challenges for real-time performance, particularly on devices with limited processing power.

Recognizing the limitations of DynaSLAM, DynaSLAM II [8] was developed to provide a more integrated approach to handling dynamic environments. DynaSLAM II incorporates tightly coupled tracking of multiple dynamic objects, improving scene

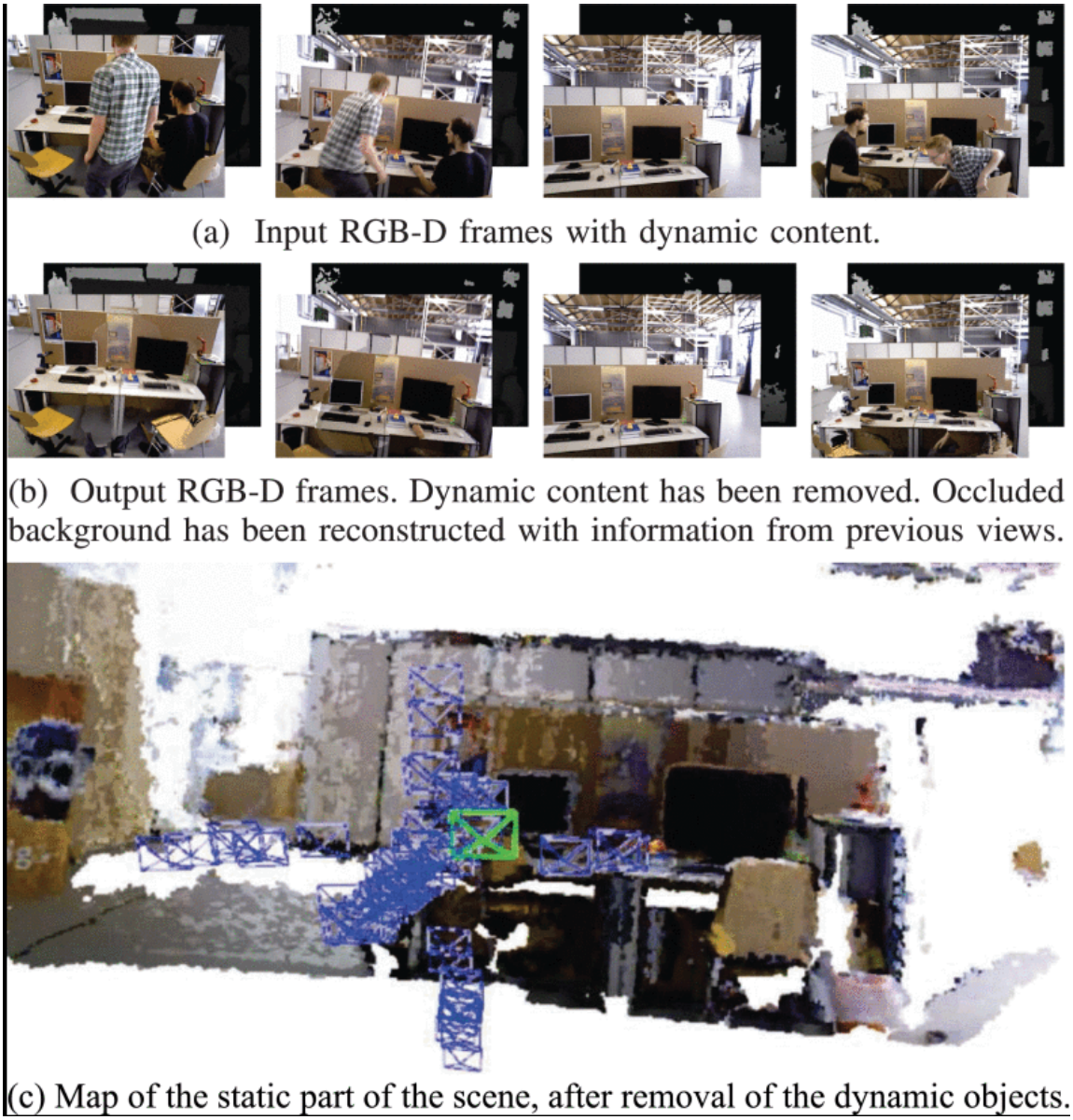
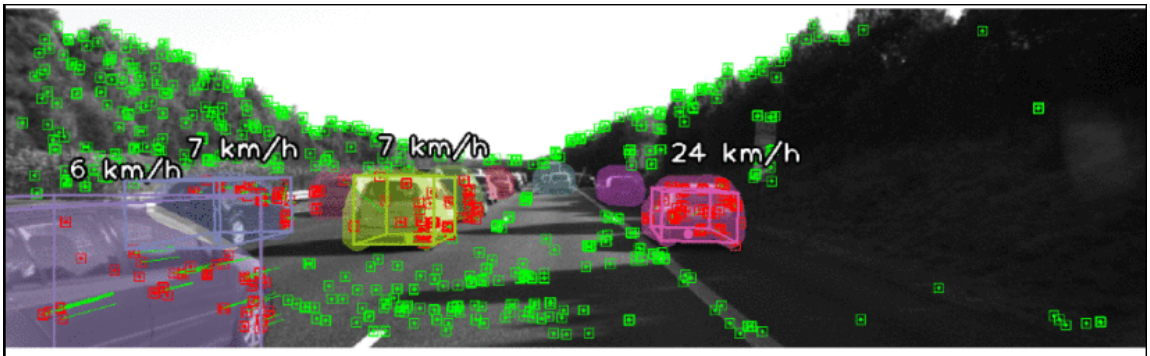
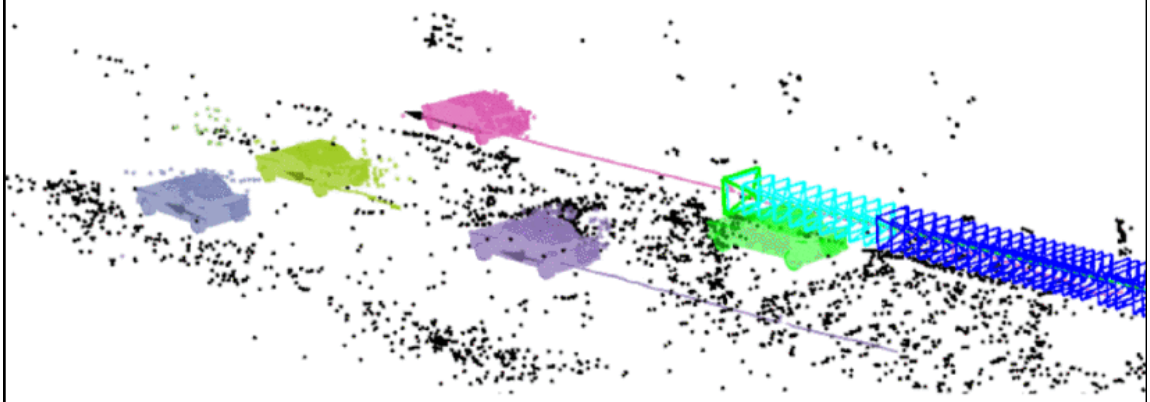


Figure 2.3: Overview of DynaSLAM results for the RGB-D case [7].

understanding and camera tracking accuracy in environments populated with moving objects, Figure 2.4. Unlike its predecessor, which focused on removing dynamic elements, DynaSLAM II integrates the tracking of these objects into the SLAM process, allowing for mutual improvements in tracking accuracy and map consistency. More research is being done to leverage Deep Neural Networks and their ability to learn from large amounts of training data to improve SLAM [89].



(a) The 3D bounding box and the speed of the objects are inferred in the image. Static and dynamic key points are in green and red respectively.



(b) Joint estimation of the camera ego motion (green car), the sparse static 3D map (black points) and the trajectories of the dynamic objects. The cyan key frames allow to optimize the map dynamic structure, whereas the blue ones only optimize the camera pose and the static structure.

Figure 2.4: An example output of DynaSLAM II [8].

In summary, almost all the theories and implementations of current SLAM approaches are built on the static world assumption, by treating any moving objects

in the environment as outliers to the static model and are intentionally ignored by tracking and mapping [79, 89]. This idealised setup, therefore, can only handle a small amount of dynamic elements and disqualifies itself from many real-world applications as environments, especially where humans are present, change constantly.

2.3 Path Planning

Planning is one of the fundamental and most studied problems in robotics [19]. The basic motion planning problem is a geometric problem of finding a collision-free path for a robot among rigid static obstacles [48]. Several extensions of the basic problem have been studied, where, for instance, kinematic constraints (such as joint limits and linkage configurations) and dynamic constraints (such as forces, torques, and inertia) limit the robot's motions, multiple robots have to be coordinated, and moving obstacles have to be considered. Therefore, the objective of path planning is to guide the robot from an initial starting point to a target goal while adhering to the robot's motion constraints [10]. Often, path planning is divided into two stages [53, 56]: global path planning, which involves generating an overall route considering the entire environment, and local path planning, which focuses on real-time navigation and obstacle avoidance based on immediate sensor data.

2.3.1 Global Path Planning

In global path planning, all the environmental information is known to the robot prior to commencing motion [11]. This includes its size, shape, current position and orientation, representation of the environment with locations of static obstacles, and target goal pose. Methods for global path planning includes:

Graph-based Approaches

Graph-based path planning algorithms are foundational techniques that represent the environment as a network of nodes and edges, where the objective is to find the shortest path between points of interest. One of the most widely known algorithms in this category is Dijkstra's algorithm [17], which guarantees the discovery of the optimal path from a given source node to every other node in the graph. Despite

its optimality, Dijkstra's algorithm is computationally intensive, particularly in large graphs, due to its exhaustive nature—requiring the evaluation of all possible paths, which can be prohibitively slow in complex or expansive environments.

To address these computational challenges, the A^* algorithm [31] extends Dijkstra's algorithm by incorporating heuristics to guide the search process more efficiently. The use of heuristics allows A^* to focus on the most promising paths, significantly reducing the number of nodes that need to be explored. This results in faster performance while still maintaining optimality, provided that the heuristic is admissible (i.e., it never overestimates the cost to reach the goal). A^* is widely favored in many robotic applications due to this balance of efficiency and optimality.

However, the application of graph-based approaches is not without challenges. A significant limitation is the need for discretization of the workspace, where the continuous environment is represented as a finite graph. This discretization can lead to performance degradation, particularly in high-dimensional spaces where the resolution of the graph may either be too coarse to capture important details or too fine, leading to an exponential increase in computational requirements. Consequently, while graph-based methods like Dijkstra's algorithm and A^* are powerful tools, their applicability in high-dimensional and complex environments often necessitates careful consideration of the trade-offs between computational efficiency and path quality.

Sampling-based Approaches

Sampling-based planning algorithms have become foundational in robotic path planning, particularly for high-dimensional configuration spaces where deterministic methods struggle due to computational complexity [19]. These approaches rely on randomly sampling the configuration space, which allows them to provide fast, albeit sometimes sub-optimal, solutions for complex problems. Among the most well-known algorithms in this category is the Rapidly-exploring Random Tree (RRT) [49], which incrementally builds a tree by expanding towards randomly selected points in the space. Despite its efficiency, a significant limitation of RRT is that it often fails to produce the shortest or most efficient paths.

To address some of these limitations, several variants of RRT have been proposed. RRT-Connect [47], for instance, enhances the basic RRT by incorporating a "Connect"

heuristic, which aggressively attempts to join two trees—one rooted at the start and the other at the goal configuration. This approach is particularly effective in solving single-query path planning problems in high-dimensional spaces. Another notable variant is the Informed RRT* [25], which improves upon the RRT* algorithm by focusing the search on regions of the configuration space that are more likely to yield better solutions. After an initial solution is found, Informed RRT* leverages a heuristic to constrain sampling to an ellipsoidal subset of the space, thereby accelerating convergence towards an optimal solution. This targeted approach significantly reduces computational overhead and enhances the quality of the final path.

In summary, while sampling-based approaches like RRT and its variants offer significant advantages in terms of computational efficiency, ongoing research continues to address their inherent limitations, such as sub-optimality and convergence speed, to make them more applicable to real-world robotic applications.

2.3.2 Local Path Planning

In real-world environments, where uncertainty is a pervasive characteristic, mobile robots must possess the capability to locally re-plan their paths in response to unforeseen obstacles and changes. This need arises because the robot often operates with incomplete information about its surroundings, making it necessary to adopt reactive strategies that enable it to make real-time decisions based on its perception of the environment [11]. Local path planning, therefore, plays a critical role in ensuring that the robot can navigate safely and effectively while tracking a global path towards its destination.

Bug Algorithms

Bug algorithms represent a class of local path planning techniques that do not rely on prior knowledge of the environment [60]. Instead, these algorithms use local sensing to detect and react to obstacles as they are encountered. Algorithms such as Bug1 and Bug2 navigate by moving directly towards the goal until an obstacle is encountered, as shown in Figure 2.5. Upon encountering an obstacle, the robot follows the obstacle's boundary until it finds a clear path to the goal. These algorithms are simple and easy to implement but can be inefficient in environments with complex or densely

packed obstacles.

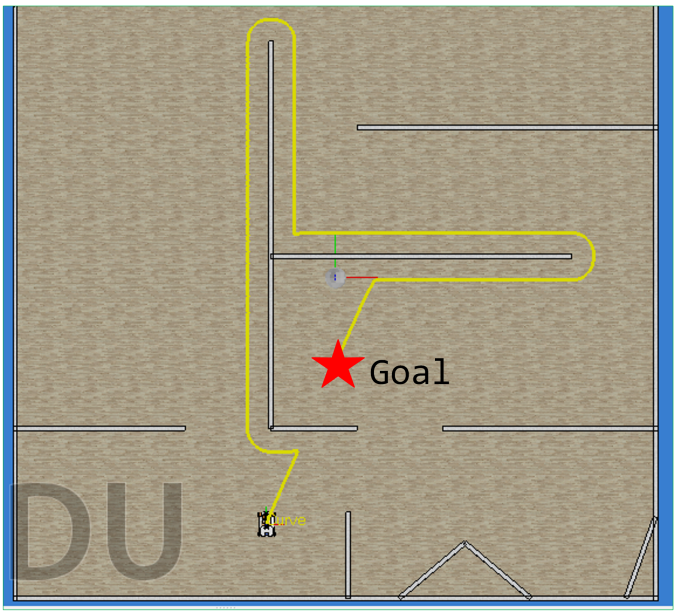


Figure 2.5: Bug2 algorithm demonstration from our implementation, showing the robot's path around obstacles.

Vector Field Histogram

The Vector Field Histogram (VFH) is a method that processes data from the robot's range sensors to create a two-dimensional grid representation of the environment [9]. This grid is then used to compute a histogram that reflects the presence and density of obstacles around the robot. By analyzing this histogram, the robot can identify safe directions for movement while avoiding obstacles.

Artificial Potential Fields

The Artificial Potential Field (APF) method is a widely-used approach in local path planning, where the robot is influenced by virtual forces that guide its motion. Obstacles generate a repulsive force, pushing the robot away, while the goal generates an attractive force, pulling the robot towards it [39]. The robot's movement is determined by the resultant force vector from these fields. While APF is intuitive

and easy to implement, it can suffer from issues such as local minima, where the robot can become stuck in a position where the forces cancel each other out.

Dynamic Window Approach

The Dynamic Window Approach (DWA) is a velocity-based local path planning algorithm that calculates the optimal collision-free velocity command to reach the target goal [20, 54]. DWA operates by constraining the robot's velocity space to a feasible range, based on sensory input and the robot's kinematic constraints. By evaluating possible velocity commands within this window, DWA selects the command that optimizes the balance between reaching the goal and avoiding obstacles. This approach is particularly effective in dynamic environments where the robot must continuously adapt its velocity to navigate safely.

2.4 Deep Reinforcement Learning in Mobile Robot Navigation

Deep reinforcement learning is widely used in various applications spanning from playing games [61, 74, 75], health care [90], and natural language processing [85, 87] to autonomous driving and robotics [2, 41]. DRL has gained significant attention in mobile robot navigation due to its ability to represent complex environments and learn from experience [91]. Figure 2.6b illustrates the interaction process between the agent and the environment within a DRL-based navigation system. In this framework, the DRL agent replaces the traditional modules for localization, mapping, and local path planning. In scenarios where structurally continuous obstacles become excessively complex, the agent may become trapped in local minima [91]. In such cases, the DRL-based navigation framework might be enhanced with additional global information through global path planning methods.

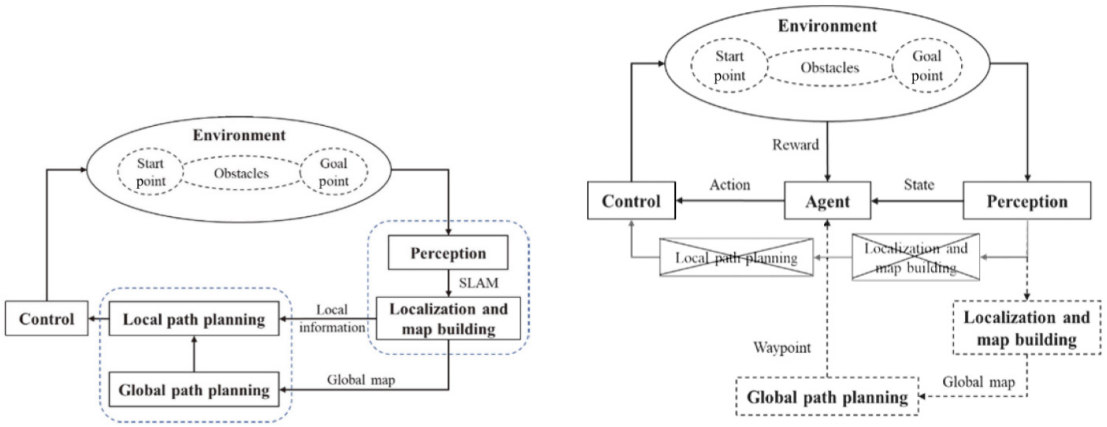


Figure 2.6: Comparison of traditional and DRL based navigation frameworks [91]. (a) traditional robot navigation framework. (b) DRL based navigation framework.

There are several approaches applying DRL for mobile robot navigation. There are methods that incorporates RL into traditional frameworks as a local planner [52, 66], this methods; however, are not fully end-to-end still requires external modules for things like mapping and global path planning. Next, we have methods that in general use RL end to end [67, 70, 92], . Finally, there are approaches to apply DRL in socially aware context [13, 14], which tries to encode social navigation norms directly into the policy.

In our work, we implemented end-to-end DRL to enable mobile robot navigation in dynamic environments. We utilized the encoders in the TD7 architecture to predict the next state, which is then used as input for both the policy and value functions.

Chapter 3

Methods

This chapter outlines the methodology employed in our study, organized as follows: Section 3.1 provides an overview of reinforcement learning, setting the groundwork for understanding the subsequent sections. Section 3.2 describes the system architecture, detailing the design and components of the embodied agent. Section 3.3 introduces the simulation environment used for training, including its setup and configuration. The formulation of mobile robot navigation as a reinforcement learning problem is discussed in Section 3.4, where we define the state space, action space, and reward function. Finally, Section 3.5 presents our solution approach, leveraging a deep reinforcement learning framework to address the navigation problem.

3.1 Reinforcement Learning Background

Reinforcement learning (RL) is learning what to do, how to map situations to actions, so as to maximize a numerical reward signal [80]. The essence of RL is learning through interaction, where the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the RL setup, Figure 3.1, an autonomous agent observes a state S_t from its environment at time step t. The agent interacts with the environment by taking an action A_t in state S_t. When the agent takes an action, the environment transition to a new state, S_{t+1}, and provides a scalar reward R_{t+1} to the agent as feedback. The goal of the agent is to learn a policy (control strategy) π that maximizes the expected return.

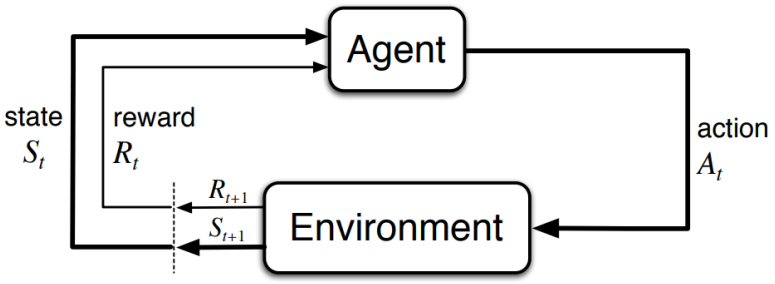


Figure 3.1: The agent-environment interaction [80].

Markov Decision Processes

Markov Decision Processes (MDPs) [6, 34] provide a formal framework for modeling decision-making problems in reinforcement learning. An MDP is formally defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ denotes the set of states.
- $\mathcal{A}$ represents the set of actions.
- $\mathcal{R}(s_t, a_t, s_{t+1})$ is the immediate reward function.
- $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ describes the transition dynamics, mapping a state-action pair at time t onto a distribution of states at time $t+1$.
- $\gamma \in [0, 1)$ is the discount factor, where lower values prioritize immediate rewards, and higher values emphasize future rewards.

In general, the policy π is a mapping from states to a probability distribution over actions $\pi : \mathcal{S} \mapsto p(\mathcal{A} = a \mid \mathcal{S})$. If the MDP is episodic, i.e., the state is reset after each episode of length T, then the sequence of states, actions, and rewards in an episode constitutes a trajectory or rollout of the policy. Every rollout of a policy accumulates rewards from the environment, resulting in the return $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. The goal of RL is to find an optimal policy, π^*, that achieves the maximum expected return from all states:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R \mid \pi] \tag{3.1}$$

It is also possible to consider non-episodic MDPs, where $T = \infty$. In this situation, the condition $\gamma < 1$ prevents an infinite sum of rewards from being accumulated.

A key concept underlying RL is the Markov property, the future is conditionally independent of the past given the present state. This means that any decisions made at s_t can be based solely on s_t, rather than $\{s_0, s_1, \ldots, s_{t-1}\}$. Although this assumption is held by the majority of RL algorithms, it is somewhat unrealistic, as it requires the states to be fully observable. A generalization of MDPs are partially observable MDPs (POMDPs), in which the agent receives an observation $O_t \in \Omega$, where the distribution of the observation $p(O_{t+1} \mid s_{t+1}, a_t)$ is dependent on the current state and the previous action [38].

Reinforcement Learning Algorithms

There are two main approaches to solving RL problems [3]: methods based on value functions and methods based on policy search. Additionally, there is a hybrid approach known as actor-critic, which employs both value functions and policy search methods.

Value Functions

Value function methods are based on estimating the value of being in a given state. The state-value function $V_\pi(s)$ for a policy π is the expected return when the agent starts in state s and uses policy π thereafter.

$$V_\pi(s) = \mathbb{E}[R|s,\ \pi] \tag{3.2}$$

The optimal policy, π^*, maximizes the expected return for every state. Correspondingly, the optimal state-value function, $V^*(s)$, can be defined as:

$$V^*(s) = \max_\pi V_\pi(s) \quad \forall s \in \mathcal{S} \tag{3.3}$$

In RL setting, the transition dynamics $\mathcal{P}$ is unavailable. Therefore, we construct another function, the state-action value function $Q_\pi(s,\ a)$, which is similar to V_π except that the initial action a is provided and π is only followed from the next state onward. The best policy, given $Q_\pi(s,\ a)$, can be found by choosing action, a, greedily at every state: $\operatorname{argmax}_a Q_\pi(s,\ a)$. Under this policy, we can also define $V_\pi(s)$ by maximizing $Q_\pi(s,\ a)$: $V_\pi(s) = \max_a Q_\pi(s,\ a)$.

This formulation forms the basis for several widely used RL algorithms, including Q-learning [88] and SARSA [71]. These algorithms iteratively update the Q values based on the agent's experiences, gradually improving the policy towards optimality.

Another major value-function-based method relies on learning the advantage function $A_\pi(s, a)$ [4]. Unlike producing absolute state-action values, as with Q_π, A_π instead represents relative state-action values. A_π represents a relative advantage of actions through the simple relationship $A_\pi = Q_\pi - V_\pi$.

Policy Search Methods

Policy search methods do not maintain a value function model, instead they directly search for an optimal policy π^*. Typically, a parameterized policy π_θ is chosen, whose parameters are updated to maximize the expected return $\mathbb{E}(R|\ \theta)$ using either gradient-based or gradient-free optimization [16].

When constructing the policy directly, it is common to output parameters for a probability distribution; for continuous actions, this could be the mean and standard deviations of Gaussian distributions, while for discrete actions this could be the individual probabilities of a multinomial distribution. The result is a stochastic policy from which we can directly sample actions.

Actor-Critic Methods

It is possible to combine value functions with an explicit representation of the policy, resulting in actor-critic methods. The "actor" (policy) learns by using feedback from the "critic" (value function). In doing so, these methods trade-off variance reduction of policy gradients with bias introduction from value based methods [45, 73].

3.1.1 Deep Reinforcement Learning

RL had some successes in the past [43, 65, 76]; however, previous approaches lacked scalability and were inherently limited to fairly low dimensional problems. These limitations exist because RL algorithms share the same complexity issues as other algorithms: memory complexity, computational complexity, and sample complexity [3, 78]. The rise of deep learning, relying on the powerful function approximation and

representation learning properties of deep neural networks, has provided us with new tools to overcome these issues.

Deep learning enables RL to scale to decision making problems that were previously intractable, *i.e.*, settings with high dimensional state and action spaces [3]. This capability allows RL to be applied to a wide range of problems, such as robotics, where control policies for robots can now be directly learned from raw sensor inputs in the real world, succeeding controllers that used to be hand-engineered or learned from low dimensional features of the robot's state [50, 51]. In general, DRL is based on training deep neural networks to approximate an optimal policy π^*, and/or the optimal value functions V^*, Q^* and A^*.

3.2 System Architecture

The concept of embodied agents is utilized to easily separate the specification and implementation phases. An embodied agent is an entity with a physical body (*e.g.,* robots) that perceives its environment through receptors and acts upon that environment through effectors, driven by an internal urge to attain a certain goal [93, 94]. An embodied agent a_j (where j is the designator of the agent, and in our case, $j = 1$ since we only have one agent) is decomposed into *five* subsystems: the control subsystem c_j, virtual receptors $r_{j,k}$, virtual effectors $e_{j,n}$, real receptors $R_{j,l}$, and real effectors $E_{j,m}$, where l, m, and n are the designators of particular subsystems of the agent a_j.

The internal structure of the agent is illustrated in Figure 3.2. The exteroceptor (Velodyne LiDAR) gathers range measurements, which are then processed by the virtual receptor to produce the environment state. Meanwhile, the interoceptor (Odometry sensor) captures the robot's odometry data, which is subsequently used by the virtual effector to generate the agent state. The control subsystem, *i.e.*, the DRL agent, utilizes data from both the virtual effector and virtual receptor, along with knowledge about the task to be executed, to produce control signals for the effectors. These signals, in turn, influence the environment, thereby closing the loop through the environment. The robot used in this study is the Pioneer P-3DX, a medium-sized, general-purpose differential drive mobile robot platform (see Figure 3.3). The differential kinematics is provided in Appendix A.

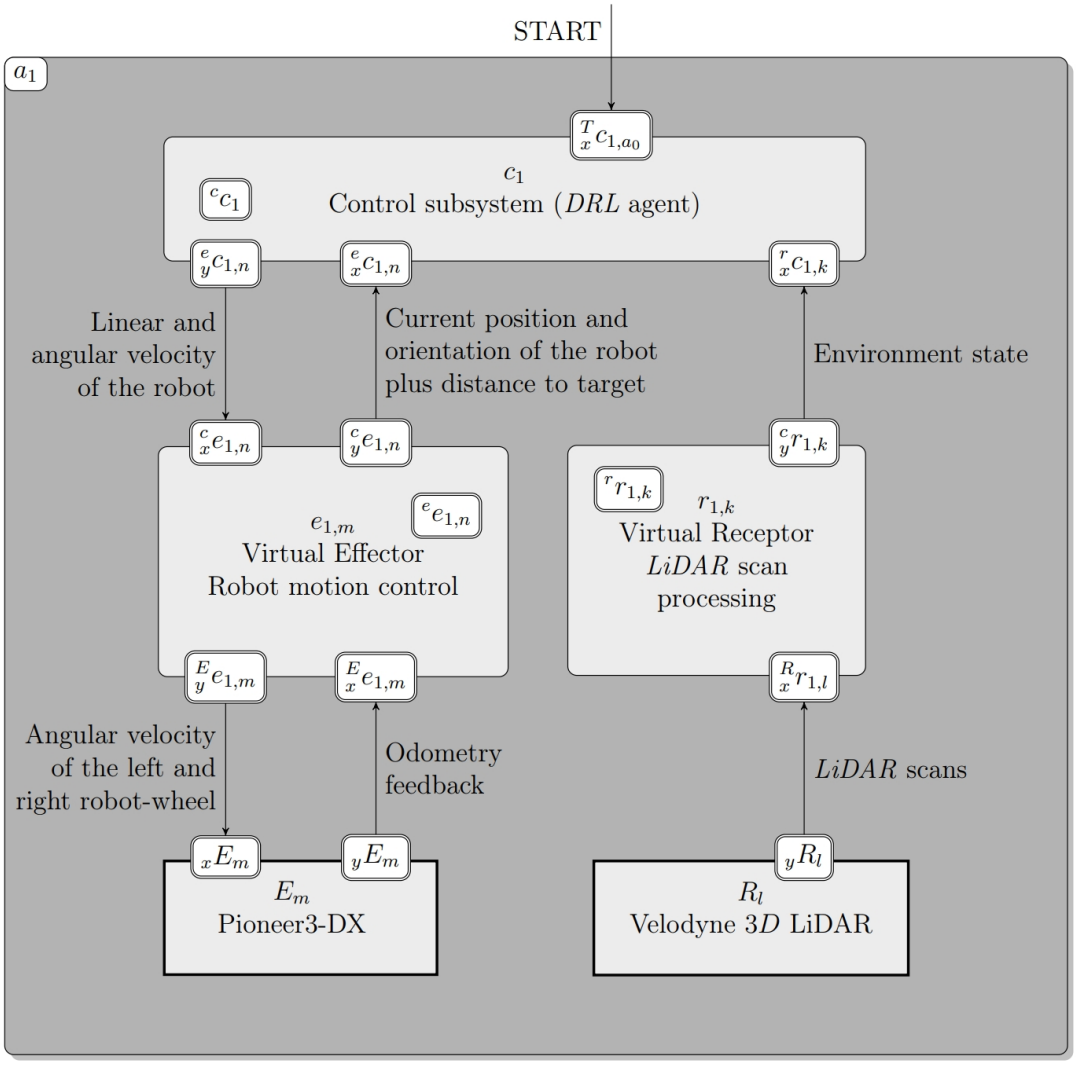


Figure 3.2: Internal structure of the agent.

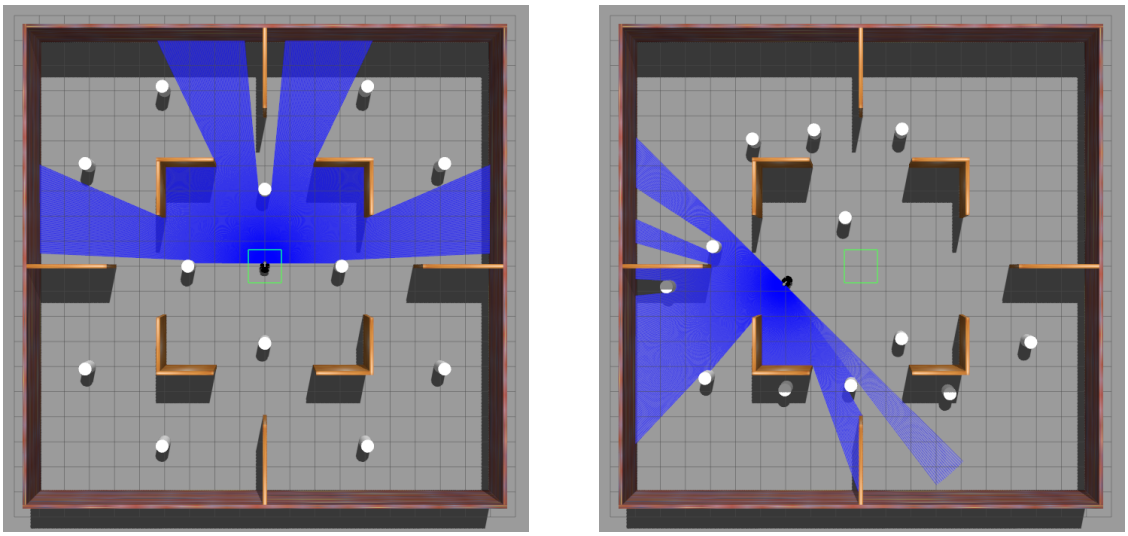
Figure 3.3: Pioneer-P3DX differential drive mobile robot.

3.3 Simulation Environment

We have built three simulation environments using Gazebo for training and testing the agent. The simulation environment used for training is shown in Figure 3.4. This environment consists of a 10m × 10m area, enclosed and divided by walls, and populated with obstacles. To enhance generalization, the positions of these obstacles are randomly altered upon each reset, *i.e.*, at the end of an episode. The robot is equipped with a 3D Velodyne laser scanner that has a 180° field of view (FOV). The laser scanner readings are divided into 20 bins, with the minimum distance from each bin selected as a representative of that bin, which is then used to construct the environment state. In addition to the laser scanner, the robot's odometry data is used to estimate its distance from and relative orientation to the target goal, forming the agent's state. The processed odometry and laser scanner data are combined to define the state space.

We have implemented a general-purpose node called `environment_node` to facilitate communication between the simulation environment and the agent using ROS topics, services, and actions, as well as to provide the necessary functionalities for performing DRL agent training and testing. This node provides the following main functionalities:

- Reset: The `reset` service server resets the world to its initial state. It then changes the starting position of the agent, sets a new target goal position, and finally shuffles the location of obstacles randomly.
- Step: The `step` service server publishes velocity commands, propagates the state for $\Delta t = 0.1$ seconds, pauses the simulation, and then retrieves the observed state after executing an action.
- Other functionalities: Two callbacks are particularly noteworthy. The first is the `environment_state` callback, where the laser range findings are processed to compute the environment state. The second is the `agent_state` callback, where the odometry data is used to calculate the distance and relative orientation of the agent to the target goal.



(a) (b)

Figure 3.4: Visualization of the Gazebo simulation environment used for training the agent. The images show the changes in the environment due to random repositioning of obstacles upon each reset: (a) initial configuration, (b) configuration after reset.

Having detailed the simulation environment and the system architecture of the embodied agent, we now move on to formulating the problem of mobile robot navigation as a reinforcement learning task. This formulation will utilize the simulation environment and the defined system architecture to establish the state space, action space, and reward function necessary for training.

3.4 Formulation of the Problem of Mobile Robot Navigation as a Reinforcement Learning Problem

In this section, we present the formulation of mobile robot navigation as a reinforcement learning problem. We start by defining the state and action spaces, and then we detail the reward function. We prioritize defining a reward function that is simple to mitigate the risk of unintended consequences. A less complex reward function helps avoid scenarios where the agent might exploit specific aspects or loopholes, leading to sub-optimal or unexpected behaviors.

3.4.1 State Space

The state space is a critical component that must encompass all relevant information about the agent's environment and its current status. As described in section 3.3, the environment is represented by laser scanner readings, which are grouped into 20 bins. Each bin captures the minimum distance to an obstacle within its field of view (FOV). This method of binning reduces the complexity of the raw sensor data while retaining essential information needed for navigation. Figure 3.5 illustrates how these laser scans are aggregated into bins to construct the environment state.

In addition to the environment state, the agent's state is defined by four values: the distance from the goal d, the relative heading to the goal (θ), and the linear (v) and angular (ω) velocities that were executed in the previous time step. These parameters are crucial as they provide the agent with information about its current trajectory and position relative to the target. Figure 3.6 depicts the formulation of the agent state, considering the agent's orientation with respect to the goal, its current distance from the goal, and the velocities it executed in the prior time step.

Combining these elements results in a state space ($\mathcal{S}$) with 24 dimensions. This state space is designed to be sufficiently rich to capture the necessary details of both the environment and the agent's status, enabling the reinforcement learning algorithm to make informed decisions during navigation.

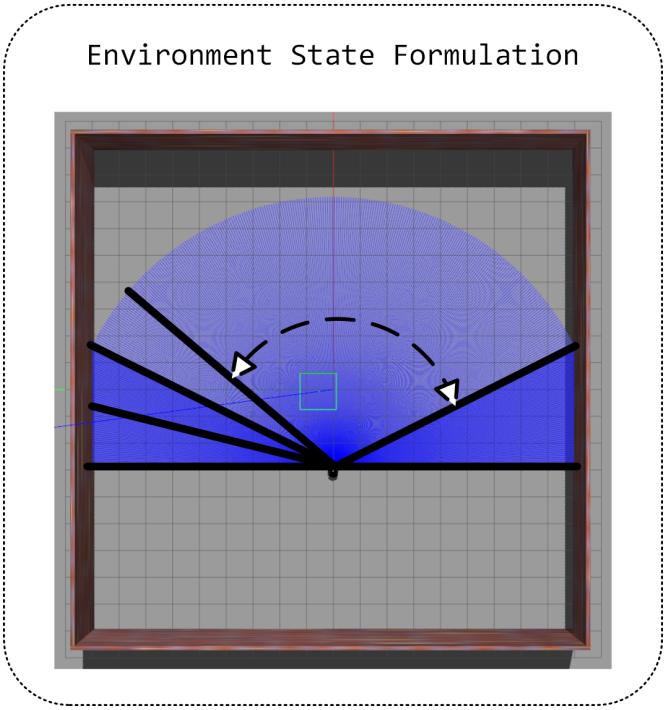


Figure 3.5: Environment state formulated by selecting the shortest range from each bin as a representative.

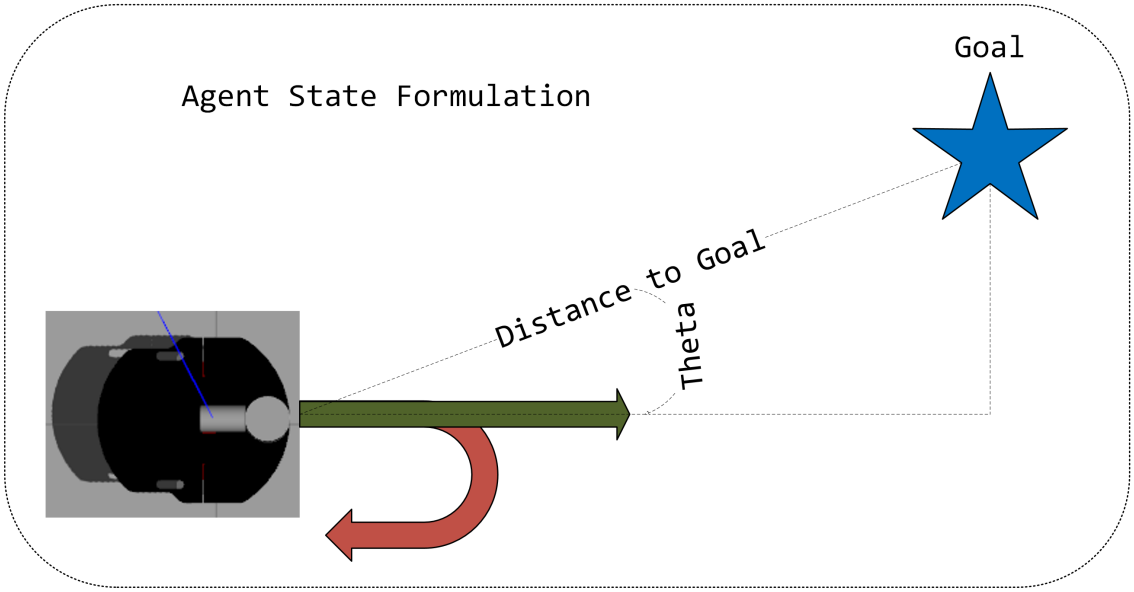


Figure 3.6: Agent state formulation

3.4.2 Action Space

The agent, Pioneer P-3DX differential drive mobile robot, can perform actions by altering its forward velocity (v) and angular velocity (ω), resulting in a continuous action space with 2 dimensions, as depicted in Figure 3.7. The range for these actions extends from $[-1, -1]$ to $[1, 1]$. For practical reasons, we have constrained the agent to only execute forward motions by clipping negative linear velocities. This restriction is due to the laser scan's 180° FOV, which makes backward movement impractical as it would leave the agent blind to obstacles behind it.

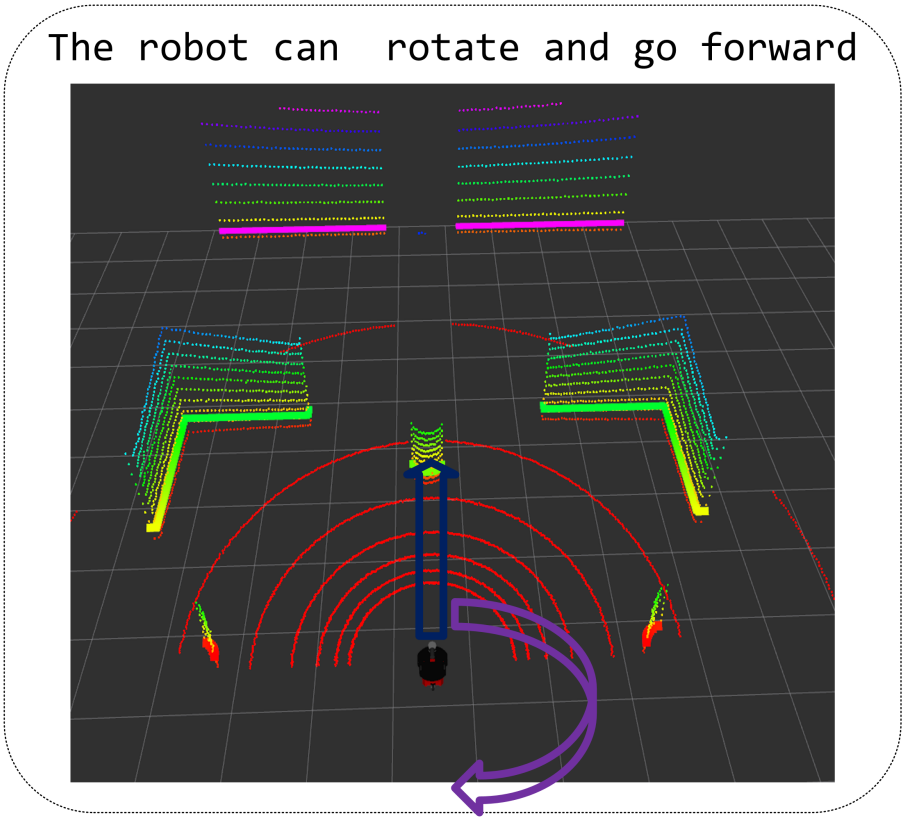


Figure 3.7: Action space formulation.

3.4.3 Reward Formulation

The reward function is designed to guide the agent in achieving its navigation goals by encouraging the selection of optimal paths while avoiding dangerous or sub-optimal

actions. To this end, the reward function is composed of three key components. The first component provides positive reinforcement for reaching the target goal. The second component imposes a penalty for collisions with obstacles. The third component offers an immediate reward to discourage the agent from coming too close to obstacles and from spending excessive time to reach the goal. These components collectively ensure that the agent navigates efficiently and safely towards its goal.

$$R = \begin{cases} r_t & \text{if } d_t < \text{GOAL\_THRESHOLD}, \\ r_c & \text{if } d_c < \text{COLLISION\_THRESHOLD}, \\ r_i & \text{otherwise} \end{cases} \tag{3.4}$$

Where r_t is the reward given when the agent reaches the target goal, r_c is the penalty incurred when the agent collides with an obstacle, and r_i is the reward for other intermediate cases.

$$r_i = \alpha(v - |\omega|) + r_g + c \tag{3.5}$$

r_i is directly proportional to the difference between v and $|\omega|$, which helps in achieving smooth motion. The term r_g represents the penalty for deviating from the global path. Furthermore, a small constant negative reward (c) is provided to encourage the agent to reach the target in a shorter time period.

$$r_g = \begin{cases} \sigma d_g & \text{if } d_g > \text{GLOBAL\_PATH\_THRESHOLD}, \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The specific values of each parameter used in the reward formulation are given in Table 3.1.

Components	Parameters	Value
r_t	r_t	100
r_c	r_c	-100
	α	0.5
$r_i = \alpha v - \beta \lvert\omega\rvert + r_g + c$	β	0.5
	c	-0.001
$r_g = \sigma d_g$ or 0	σ	0.0

Table 3.1: Reward parameters

3.5 Solution to the Problem of Mobile Robot Navigation in a Dynamic Environment

The proposed solution for mobile robot navigation in dynamic environments leverages a deep reinforcement learning approach, specifically employing an actor-critic network architecture. The approach is grounded in the TD3 algorithm [21] and its recent advancement, TD7 [24]. TD3 is recognized for its effectiveness in handling continuous action spaces and enhances training stability through key techniques such as twin Q-networks and delayed policy updates. Building on this, TD7 introduces state-action learned embeddings, which facilitate the modeling of environmental dynamics in a latent space. This enhancement enables the network to better comprehend and adapt to changes in the environment, thereby improving navigation performance in complex, dynamic settings.

3.5.1 Network Architecture

Encoder Network

To effectively capture dynamic actors in the environment, enabling the policy to make more informed actions, it is essential to predict the next state of the environment accurately. For this purpose, we employ a pair of encoders (f, g). The encoder $f(s)$ transforms the state s into a state embedding z^s, while $g(z^s, a)$ jointly encodes both the state embedding z^s and action a into a state-action embedding z^{sa}. This encoding process is designed to capture the relevant structures within the observation space and to model the transition dynamics of the environment effectively.

$$z^s \coloneqq f(s), \quad z^{sa} \coloneqq g(z^s, a) \tag{3.7}$$

Splitting the embeddings into state and state-action components allows the encoders to be trained with a dynamics prediction loss that depends solely on the next state s', independent of the next action or the current policy [24]. Consequently, the encoders are trained jointly using the mean squared error (MSE) between the state-action embedding z^{sa} and the embedding of the next state $z^{s'}$:

$$\mathcal{L}(f, g) \coloneqq (g(f(s),\ a) - |f(s')|_{\mathrm{x}})^2 = \left(z^{sa} - |z^{s'}|_{\mathrm{x}}\right)^2 \tag{3.8}$$

The encoders (f, g) are jointly trained to predict the next state embedding, which is decoupled from the training of the value function and policy. In Equation 3.8, the gradient stop operation for this decoupling is denoted by $|\cdot|_{\mathrm{x}}$.

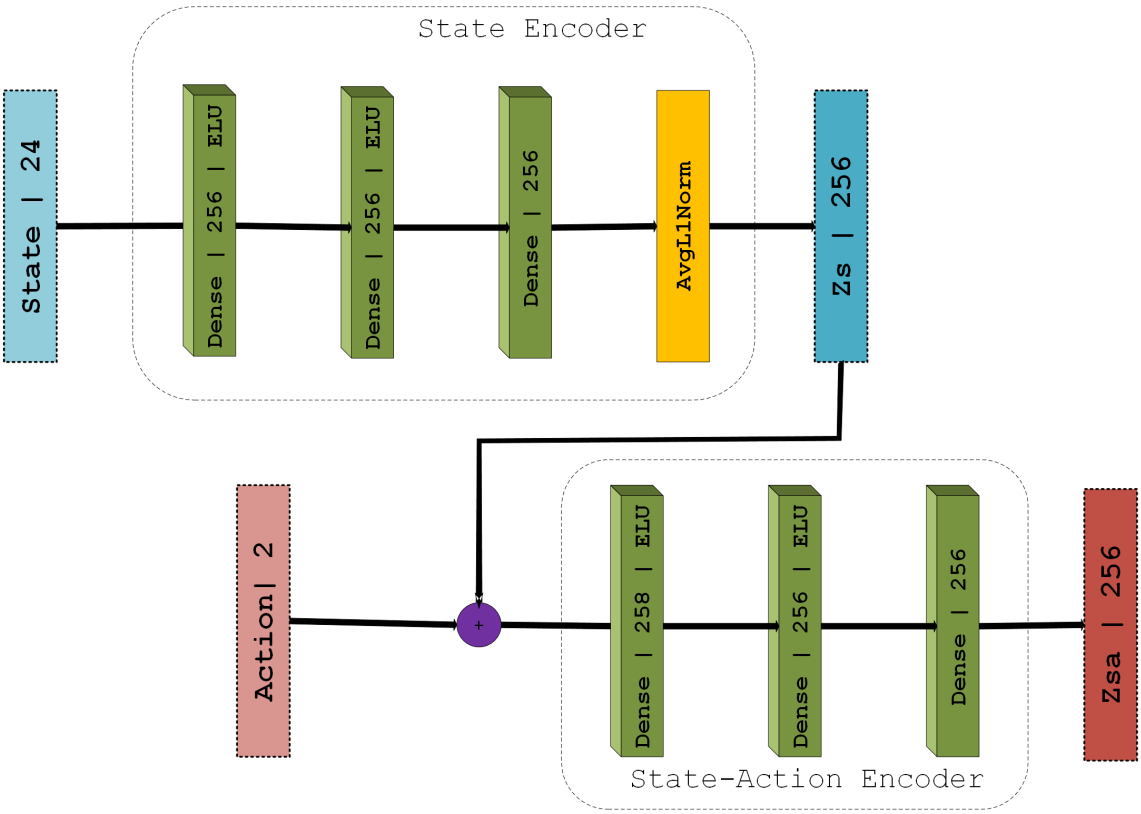


Figure 3.8: Encoder network architecture

In Figure 3.8 and the subsequent network architecture figures, fully connected dense layers are represented by green (■) cubes, normalization layers are represented by yellow (■) rectangles, and the concatenation operation is represented by purple (■) circles. The inputs and outputs of the network are depicted as rectangles with dotted boundaries. Additionally, embeddings are represented in the same color as their respective inputs but in a darker shade.

Actor and Critics Networks

The embeddings are designed to capture the underlying structure of the environment. However, they may not include all relevant information needed by the value function and policy, such as features related to the reward, current policy, or task horizon [24]. To address this, we concatenate the embeddings with the original state and action, as shown in Figures 3.9 & 3.10. This approach allows the value and policy networks to learn the necessary internal representations for their respective tasks.

$$Q(s,a) \to Q(z^{sa}, z^{s}, s, a), \quad \pi(s) \to \pi(z^{s}, s) \tag{3.9}$$

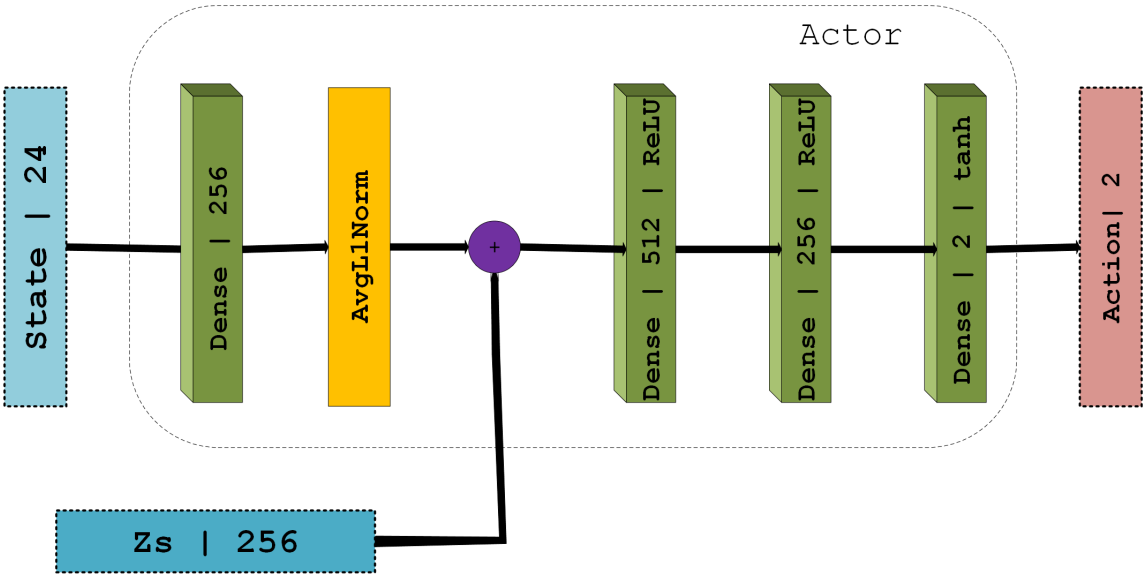
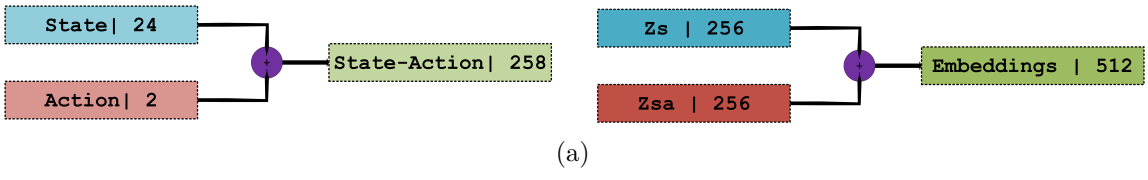
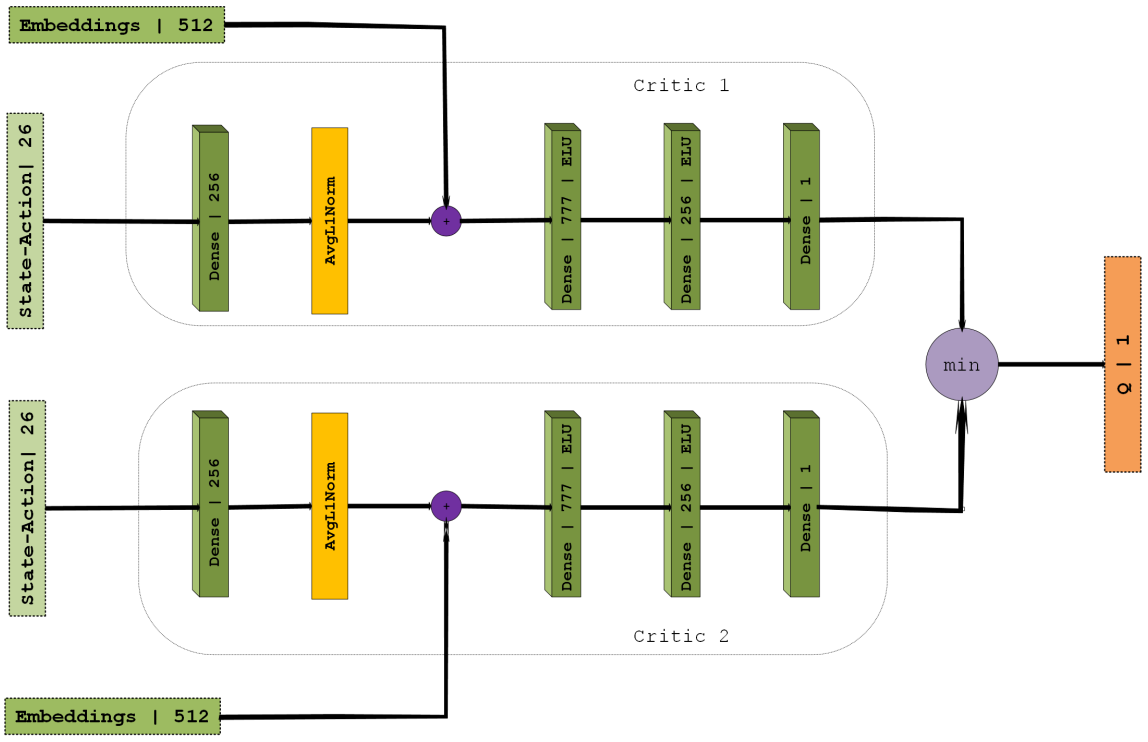


Figure 3.9: Actor network architecture



(a)



(b)

Figure 3.10: Critic network inputs and architecture. (a) The input to the critic network consists of the concatenation of the state and action (`State-Action`) and the concatenation of the state and state-action embeddings (`Embeddings`).

3.5.2 Agent Training

The encoders f and g are trained online and concurrently with the RL agent. However, they are decoupled in the sense that gradients from the value function and policy are not propagated to f and g. The following considerations were taken into account during the practical implementation:

Normalized embeddings: Minimizing distances in embedding space can lead to instability due to either monotonic growth or collapse into a redundant representation [26]. To mitigate this risk, the TD7 algorithm introduces AvgL1Norm, a normalization layer that divides the input vector by its average absolute value across each dimension, thereby maintaining the relative scale of the embedding constant throughout the learning process. Let x_i represent the i^{th} dimension of an N-dimensional vector x, then:

$$\text{AngL1Norm}(x) \coloneqq \frac{x}{\frac{1}{N}\sum_i |x_i|} \tag{3.10}$$

The AvgL1Norm is applied to the state embedding z^s and, following a linear layer, to the state and action inputs in the policy π and value function Q, ensuring they remain on a scale similar to the learned embeddings, as illustrated in Figures 3.8, 3.9, and 3.10. The input to the value function and policy is then defined as:

$$Q(z^{sa}, z^s, \text{AvgL1Norm}(\text{Linear}(s,a))),\ \ \pi(z^s, \text{AvgL1Norm}(\text{Linear}(s))) \tag{3.11}$$

Unlike the embeddings z^s and z^{sa}, these linear layers are learned end-to-end and can therefore be considered an integral part of the architecture of the policy and value functions.

Fixed embeddings: Inconsistent input can cause instability [24]. To mitigate this, the TD7 algorithm freezes the embeddings used to train the current policy and value networks. Consequently, at iteration $t+1$, the inputs to the current policy network π_{t+1} and value network Q_{t+1} uses embeddings z_t^s and z_t^{sa} from the encoders g_t and f_t at the previous iteration t. The value function and policy are updated as follows:

$$\begin{aligned} \pi_{t+1}(z_t^s, s) &\approx \underset{\pi}{\operatorname{argmax}}\, Q_{t+1}(z_t^{sa}, z_t^s, s, a), \text{ where } a \sim \pi_t(z_t^s, s) \\ Q_{t+1}(z_t^{sa}, z_t^s, s, a) &\approx r + \gamma Q_t(z_{t-1}^{s'a'}, z_{t-1}^{s'}, s', a'), \text{ where } a' \sim \pi_t(z_{t-1}^{s'}, s') \end{aligned} \tag{3.12}$$

The current value function Q_{t+1} is trained using the previous value function Q_t as a target network. The current embeddings z_{t+1}^s and z_{t+1}^{sa} are trained with Equation 3.8, with a target zs_{t+1}; therefore, without a separate target network. At every n steps, the iteration is incremented, and all target networks are updated simultaneously:

$$Q_t \leftarrow Q_{t+1},\ \pi_t \leftarrow \pi_{t+1},\ (f_{t-1}, g_{t-1}) \leftarrow (f_t, g_t),\ (f_t, g_t) \leftarrow (f_{t+1}, g_{t+1}) \tag{3.13}$$

Clipped Values: Extrapolation error occurs when deep value functions predict unrealistic values for state-action pairs that are rarely encountered in the dataset [22]. The inclusion of the state-action embedding z^{sa} expands the action input, making the value function more susceptible to over-extrapolating on unfamiliar actions [24]. The dimensionality of z^{sa} and the corresponding state-action input are crucial for maintaining stable value estimates. In online reinforcement learning, this error can be naturally corrected through feedback from interactions with the environment. Therefore, the key challenge is to stabilize the value estimates until these corrections can take place. The TD7 algorithm addresses this by monitoring the range of values in the dataset D, which are estimated over sampled mini-batches during training. It then constrains the target used in Equation 3.12 to fall within this observed range:

$$Q_{t+1}(s, a) \approx r + \gamma \operatorname{clip}\left(Q_t(s', a'), \min_{(s,a)\in D} Q_t(s, a), \max_{(s,a)\in D} Q_t(s, a)\right) \tag{3.14}$$

Hardware Specification Used for the Training

The training process was conducted using a Gigabyte G7-MF laptop, which is equipped with a 12th Gen Intel® Core™ i5-12500H processor featuring 16 cores. This setup also includes an NVIDIA GeForce RTX 4050 Laptop GPU and 16GB of RAM.

The training of the DRL agent was governed by the TD7 algorithm, which

is detailed in Algorithm 1 and Appendix B. As the training progressed, various metrics were monitored to evaluate the agent's learning performance. Figure 3.11 shows the average Q-value over time, illustrating how the agent's estimation of the expected return improves as it interacts more with the environment. The Q-value graph indicates that initially, the agent struggled to predict the correct Q-values, as evidenced by the low and fluctuating values. However, as training continued, the Q-values stabilized and gradually increased, reflecting the agent's improved understanding of the environment and its policy.

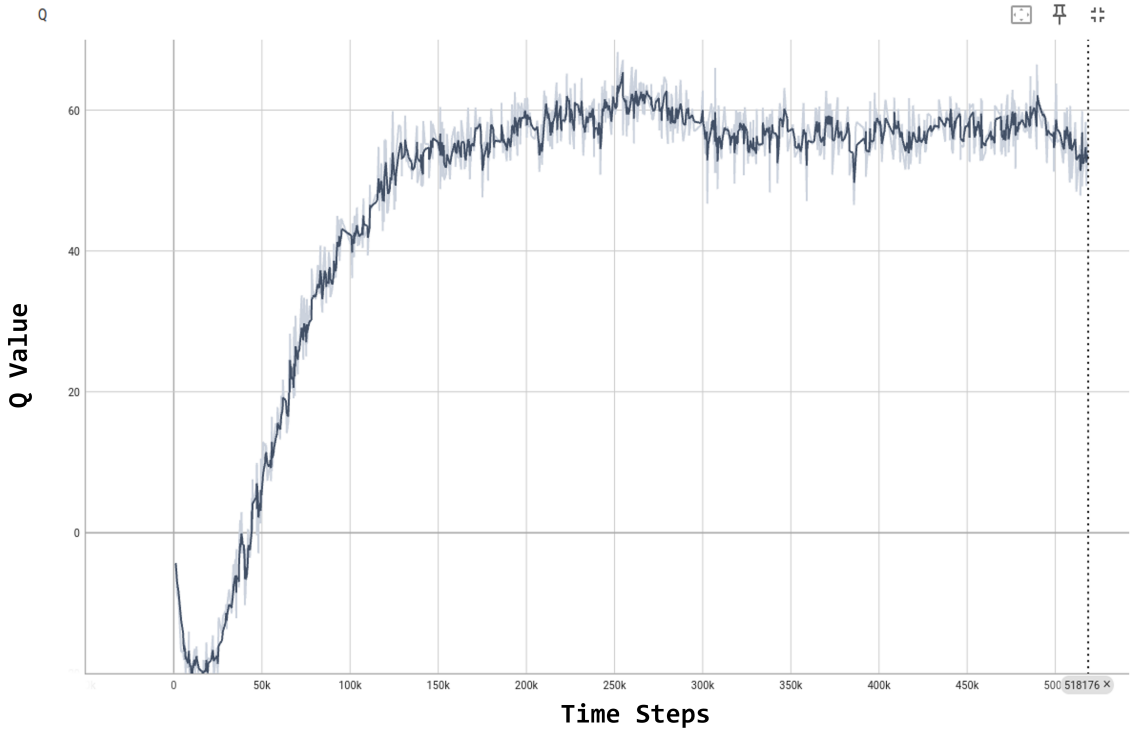


Figure 3.11: Improvement in the average Q value as training proceeds.

In parallel, the evaluation of the agent's performance was conducted periodically throughout the training process. Figure 3.12 presents the average reward obtained during these evaluation phases. The reward graph shows a clear upward trend, especially in the earlier epochs, indicating that the agent was successfully learning to achieve its goals more effectively over time. The spikes and fluctuations in the reward graph can be attributed to the exploration-exploitation trade-offs that are

inherent in reinforcement learning. Over time, the agent's performance became more consistent, as seen in the smoothing of the reward curve in later epochs, suggesting that the agent was converging on an optimal policy.

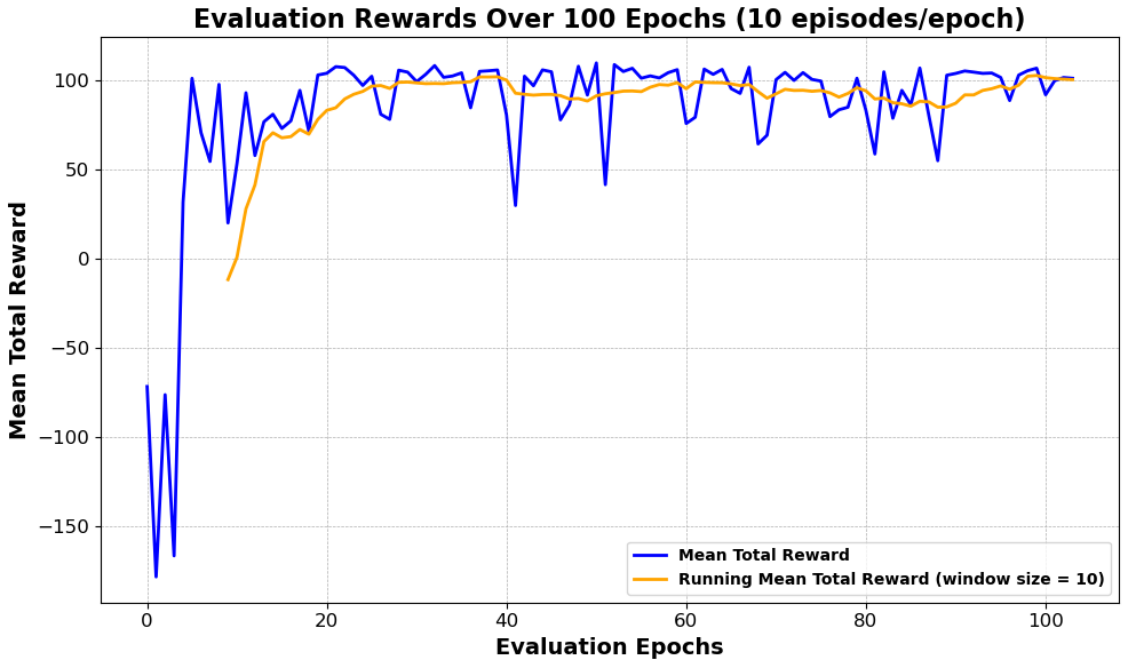


Figure 3.12: Average reward for 10 evaluation episodes at every 5000 time steps.

Algorithm 1 TD7

Initialize:
Policy π_{t+1}, value function Q_{t+1}, encoders (f_{t+1}, g_{t+1}).
Target policy π_t, target value function Q_t, fixed encoders (f_t, g_t), target fixed encoders (f_{t-1}, g_{t-1}).
Checkpoint policy π_c, checkpoint encoder f_c.

Data collection

for episode = 1 to `final_episode` **do**
  Using π_{t+1}, collect transitions and store in the LAP replay buffer.
  Checkpointing
  if `checkpoint_condition` **then**
    if actor π_{t+1} outperforms checkpoint policy π_c **then**
      Update checkpoint networks $\pi_c \leftarrow \pi_{t+1}$, $f_c \leftarrow f_t$.
    end if
  end if
  Training
  for $i = 1$ to `timesteps_since_training` **do**
    Sample transitions from LAP replay buffer.
    Train encoder, value function, and policy.
    if `target_update_frequency` steps have passed **then**
      Update target networks.
    end if
  end for
end for

Chapter 4

Experiment Results

In this chapter, we present the results of the experiments conducted to evaluate the performance of our implemented system. We have tested our system in a range of environments that vary in complexity to assess its robustness and effectiveness. For comparison purposes, we evaluated our system against a baseline: *Goal-Driven Autonomous Exploration Through Deep Reinforcement Learning* (GDAE) [15].

Chapter organization: We begin by examining the results from the simplest environment in Section 4.1. This environment consists of an empty space partitioned by walls, and the primary objective for the agent is to navigate from the start to the goal while avoiding collisions with the walls. Section 4.2 describes a more complex scenario involving static obstacles. In this case, while some obstacles remain in fixed positions, others change their locations upon each reset, adding a layer of challenge for the navigation task. Finally, Section 4.3 presents our most complex environment, which includes both static obstacles and dynamic actors. Here, the agent must navigate through an environment populated with moving obstacles that it has no prior knowledge of, making the task significantly more challenging.

4.1 Test Case 1: Environment with No Obstacle

In this test case, we evaluate the baseline performance of our navigation system within a simple environment devoid of obstacles. The Gazebo simulation environment used for this scenario is depicted in Figure 4.1. This environment consists of an open

space partitioned by walls, creating a straightforward scenario to assess the agent's basic navigation capabilities. The absence of additional obstacles ensures that the agent's task is solely focused on navigating from the start position to a predefined goal, without any external interference.

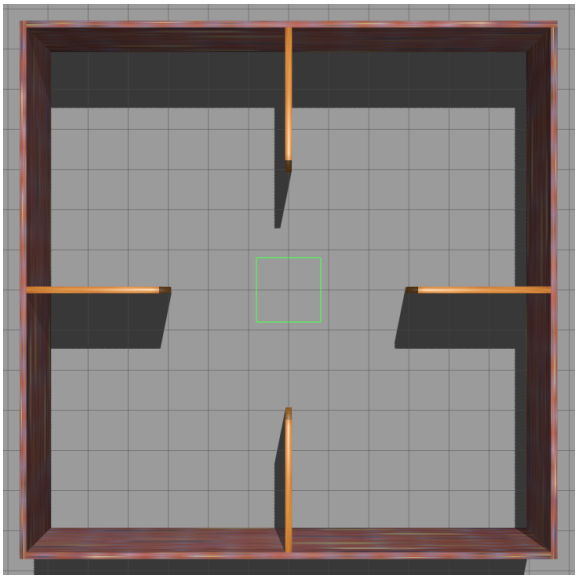


Figure 4.1: Gazebo environment used for test case 1.

4.1.1 Result Discussion

To comprehensively assess the system's performance, we conducted `five` independent test runs. The results of our experiments are summarized in Table 4.1, where we compare the performance of our method, against the baseline. In this relatively simple environment, both methods were able to guide the agent to the target goal with a 100% success rate. The baseline method, however, exhibited slightly better performance in terms of the average time taken to reach the goal and the average distance traveled.

Method	Avg. Time(sec)	Avg. Distance m	Success	Collision
Ours	11.5874	10.2191	1.0	0.0
Baseline	10.9508	9.4772	1.0	0.0

Table 4.1: Environment one, result comparison

Figure 4.2 presents the trajectories generated by the baseline method (on the left) and our method (on the right), both superimposed on the map of the environment. The start positions are marked by blue (■) circles, and the goal positions are indicated by red (■) stars. These trajectories were obtained by recording and saving the agent's position based on its odometry data at each time step. From this figure, we can observe that both methods produced nearly identical trajectories.

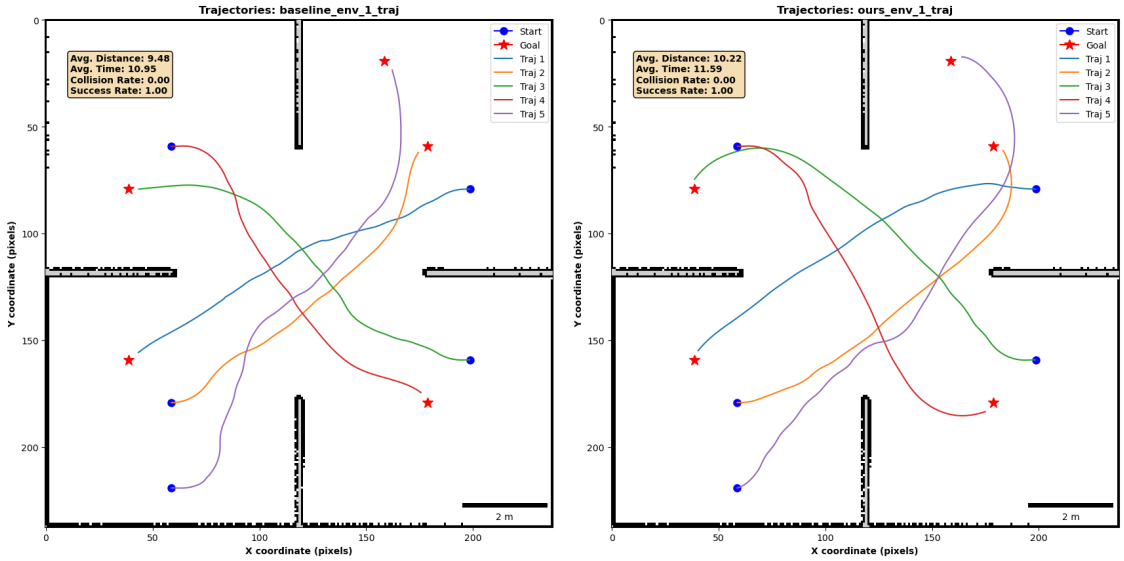


Figure 4.2: Agent trajectories for test case 1 overlaid on the map of the environment.

4.2 Test Case 2: Environment with Static Obstacles

The Gazebo environment for this test case, depicted in Figure 4.3, introduces numerous static obstacles scattered throughout the environment. This setup simulates a more complex and realistic scenario where the agent must navigate around obstacles to reach its goal. The static nature of the obstacles tests the agent's ability to plan and execute a path that avoids collisions.

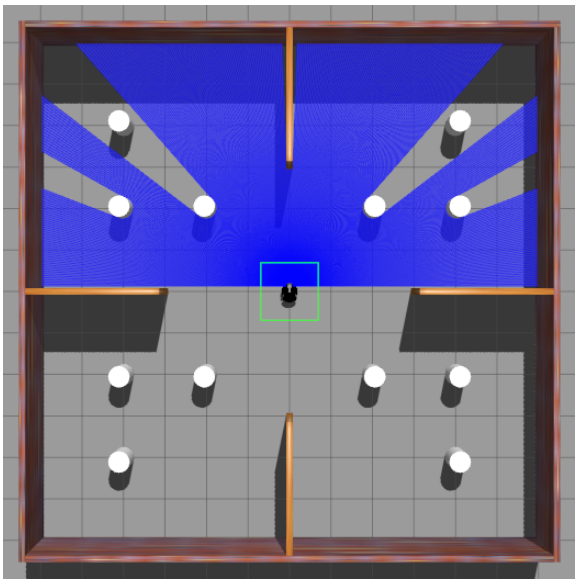


Figure 4.3: Gazebo environment used for test case 2.

4.2.1 Result Discussion

Similar to test case 1, we conducted `five` test runs in this environment. The data presented in Table 4.2 clearly demonstrate that our method outperforms the baseline, particularly in terms of success rate and collision rate. Our method consistently achieved a 100% success rate across all test runs, successfully navigating to the goal without any collisions. To ensure an accurate comparison, trajectories where the target goal was not reached were excluded from the calculation of average time and average distance traveled.

Methods	Avg. Time (sec)	Avg. Distance(m)	Success	Collision
Ours	12.3035	9.9404	**1.0**	**0.0**
Baseline	10.5075	8.9495	0.6	0.4

Table 4.2: Environment two, result comparison

Figure 4.4 provides a qualitative comparison of the trajectories generated by the baseline method (on the left) and our method (on the right). It is evident that the baseline method resulted in collisions with a static obstacle (trajectory 4 and a wall (trajectory 5. As the complexity of the environment increases, the

superior performance of our method becomes more apparent, clearly outperforming the baseline.

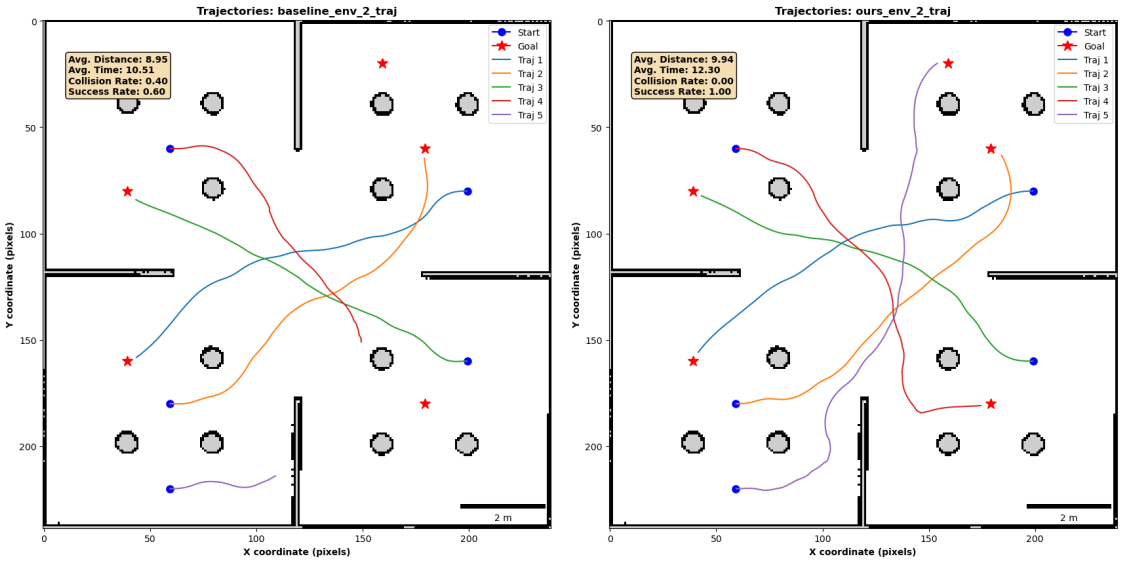


Figure 4.4: Agent trajectories for test case 2 overlaid on the map of the environment.

4.3 Test Case 3: Environment Shared with Other Actors

The Gazebo environment for this test case, illustrated in Figure 4.5, includes both static obstacles (the walls) and dynamic actors. This environment presents the most challenging scenario, requiring the agent to avoid fixed obstacles while also navigating in the presence of moving actors that simulate real-world dynamic elements, such as pedestrians or other agents. This setup tests the agent's adaptability and real-time decision making abilities in a dynamic and unpredictable environment.

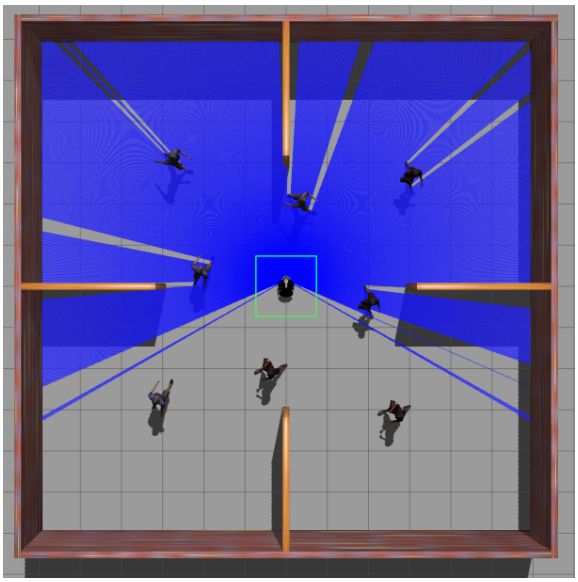


Figure 4.5: Gazebo environment used for test case 3.

4.3.1 Result Discussion

We conducted *five* test runs in this environment, and the quantitative comparison is provided in Table 4.3. As the data in this table indicates, our method outperformed the baseline method across all metrics. However, while our method performed well, it did not achieve a 100% success rate, indicating that there is still room for improvement.

Methods	Avg. Time(sec)	Avg. Distance m	Success	Collision
Ours	**6.0847**	**5.0845**	**0.8**	**0.2**
Baseline	6.6669	5.6258	0.6	0.4

Table 4.3: Environment three, result comparison

Summary of Performance Comparison with Baseline

In this section, we provide a summary of the results comparison with the baseline across all environments. Figure 4.6 presents a summary of the performance comparison with the baseline. In Test Environment 1 (`td7_empty`), both our method and the baseline performed well, achieving an average success rate of 100%. However, in Test Environments 2 and 3, `td7_static` and `td7_dynamic`, our method outperformed the

baseline. The vertical axis represents the three environments, while the horizontal axis displays the values of the respective test metrics: meters for average distance, seconds for average time, and percentages for success and collision rates.

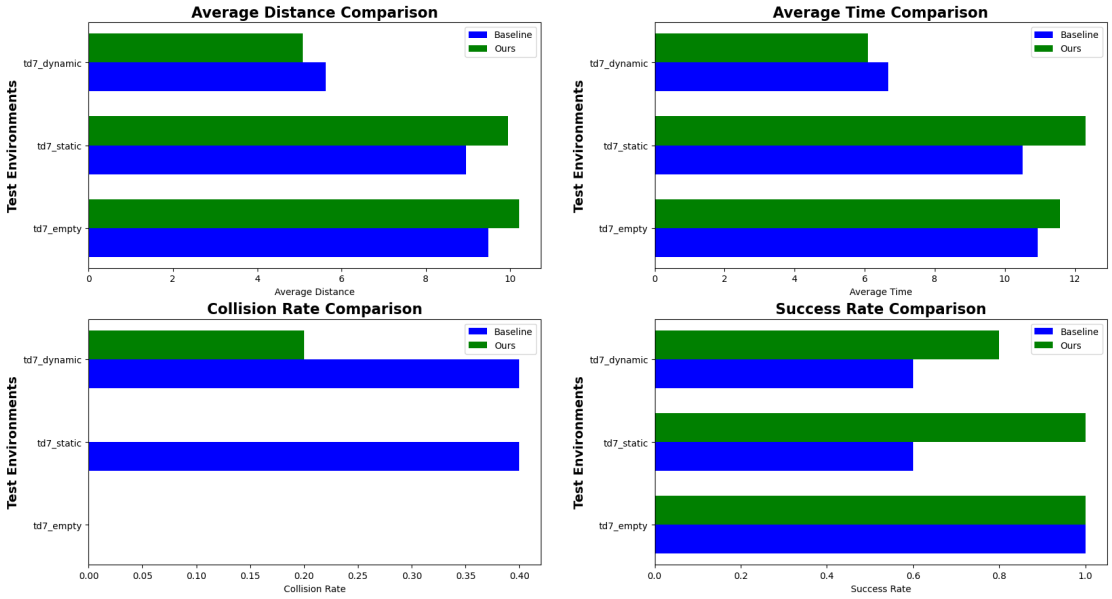
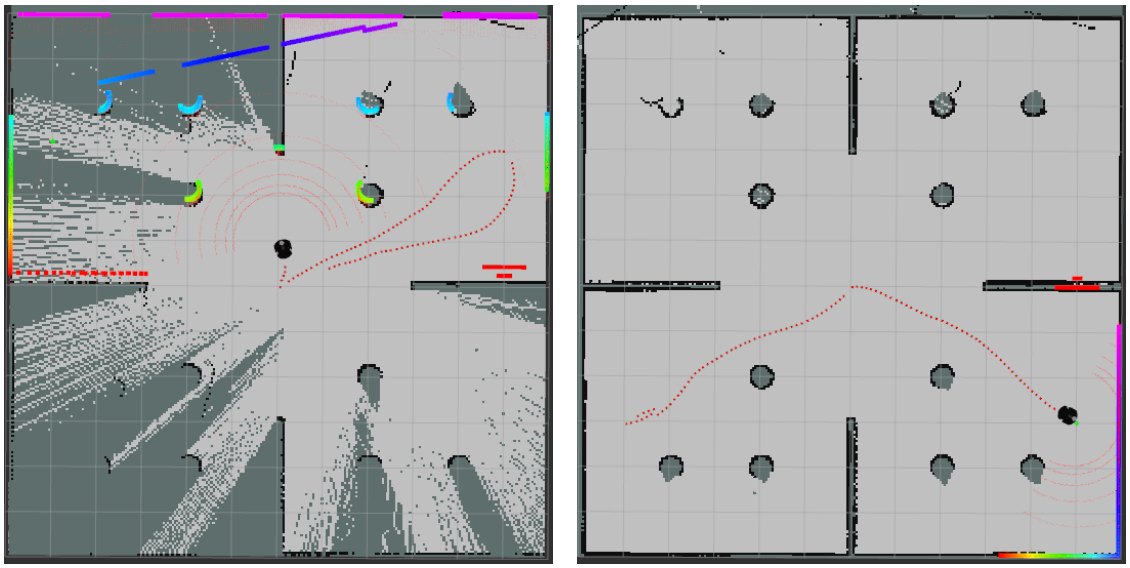


Figure 4.6: Summary of performance comparison with baseline.

4.4 Mapping Unknown Environment

One of the potential application of this framework is exploration. We can use this framework to autonomously navigate in unknown environment and use SLAM frameworks to generate a map of that environment for further application. What is important to note here is that, unlike traditional frameworks, our framework does not require a map for its operation; however, we can easily integrate other functionalities, such as mapping, if that is required by the application at hand.

A key potential application of this framework is autonomous exploration. This system can be utilized to navigate unknown environments and, in combination with SLAM frameworks, generate maps for further use. Notably, unlike traditional systems, our framework does not require a pre-existing map for navigation. However, it can seamlessly integrate additional functionalities, such as mapping, when required by specific applications.



(a) (b)

Figure 4.7: Mapping unknown environment. (a) captured during the active mapping process (b) the resulting map generated upon completion.

Figure 4.7 illustrates the application of this framework in conjunction with the SLAM Toolbox [57] for autonomously mapping an unknown environment.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this thesis, we have presented a framework for mobile robot navigation in dynamic environments using DRL. The proposed approach leverages the TD7 algorithm, an enhancement of TD3 with state-action embeddings, to enable efficient and reliable navigation in environments characterized by both static and dynamic obstacles. The framework was implemented using ROS and validated through extensive simulations in the Gazebo environment.

The experimental results from the three test cases demonstrate that the proposed method outperforms the baseline approach, particularly in complex environments, in terms of success rate and collision avoidance. In simpler environments without obstacles, both our method and the baseline achieved a 100% success rate. However, as the environment's complexity increased—with the introduction of static obstacles and dynamic actors—our method consistently outperformed the baseline.

5.2 Future Work

While the results obtained in this thesis are promising, there are several avenues for future research that could further enhance the capabilities and applicability of the proposed navigation framework:

Real-world Implementation: Although the framework has been thoroughly tested in simulated environments, transitioning to real-world scenarios is a critical next step. Despite the high fidelity of simulation environments, they cannot fully replicate the complexities of the real world, leading to challenges in the sim-to-real transfer. Addressing issues such as sensor noise and the physical limitations of actual robotic hardware will be essential for deploying this framework in real-world settings.

Improved Reward Design through Inverse Reinforcement Learning: One of the critical challenges in reinforcement learning is designing effective reward functions. For tasks such as navigating to a target goal and avoiding obstacles, manually crafting a reward function, as done in this work, can be feasible. However, as task complexity increases, this process becomes time-consuming and prone to errors, potentially leading to sub-optimal policies. Inverse reinforcement learning (IRL) offers a powerful alternative by focusing on learning the reward function from expert demonstrations/feedback rather than explicitly defining it. This approach is particularly beneficial in scenarios where specifying the correct reward function is difficult due to the environment's complexity or unpredictability, such as in human-robot interaction applications like robot companions.

Moreover, IRL can help mitigate the issue of reward sparsity, a common problem in reinforcement learning where the agent receives minimal feedback during training, making it challenging to learn optimal behaviors.

Appendix A

Differential Drive Kinematics

Deriving a model for the whole robot's motion is a bottom-up process. Each individual wheel contributes to the robot's motion and, at the same time, imposes constraints on robot motion. Wheels are tied together based on robot chassis geometry, and therefore their constraints combine to form constraints on the overall motion of the robot chassis.

A.1 Wheeled Kinematics

For a robot using several wheels, we are interested in finding the relationship between wheel speeds, $\dot{\varphi}$, and platform velocity, $\dot{\xi} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}$. We assume that the robot moves on a horizontal plane with point contact of the wheels, which are not deformable. The wheels exhibit pure rolling without any slipping, skidding, or sliding, and there is no friction for rotation around the contact point. Additionally, the wheels are connected to a rigid frame (chassis).

A.1.1 Deriving a General Wheel Equation

The position of the wheel in the inertial frame, Figure A.1, can be expressed as

$$\mathbf{r}_{IW} = \mathbf{r}_{IR} + \mathbf{r}_{RS} + \mathbf{r}_{SW} \tag{A.1}$$

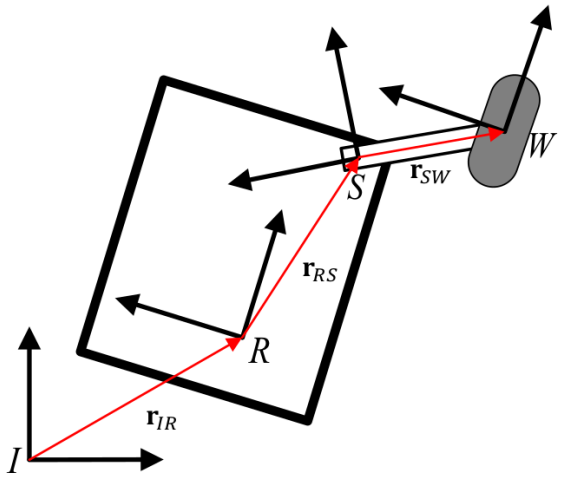


Figure A.1: Reference frames used in deriving the general wheel equation. Here, I represents the inertial frame, R represents the robot frame, S represents the steering frame, and W represents the wheel frame.

By taking the time derivative of Equation (A.1) in the inertial frame, and assuming that $\mathbf{V}_{RS} = \dot{\mathbf{r}}_{RS} = 0$ and $\mathbf{V}_{SW} = \dot{\mathbf{r}}_{SW} = 0$, the wheel equation is derived as follows:

$$\mathbf{V}_{IW} = \mathbf{V}_{IR} + \boldsymbol{\omega}_{IR} \times \mathbf{r}_{RS} + \boldsymbol{\omega}_{IR} \times \mathbf{r}_{SW} + \boldsymbol{\omega}_{RS} \times \mathbf{r}_{SW} \tag{A.2}$$

For a standard wheel, Figure A.2, the wheel equation reduces to:

$$\mathbf{V}_{IW} = \mathbf{V}_{IR} + \boldsymbol{\omega}_{IR} \times \mathbf{r}_{RS} \tag{A.3}$$

Which with respect to the wheels frame can be expressed as:

$$_W\mathbf{V}_{IW} = {}_W\mathbf{V}_{IR} + {}_W\boldsymbol{\omega}_{IR} \times {}_W\mathbf{r}_{RS} \tag{A.4}$$

Where: $_W\mathbf{V}_{IR} = R(\alpha+\beta)R(\theta) \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix}$, $\quad {}_W\boldsymbol{\omega}_{IR} \times {}_W\mathbf{r}_{RS} = \begin{bmatrix} l\sin\beta \\ l\cos\beta \\ 0 \end{bmatrix} \dot{\theta}$

$$constraints = \begin{cases} [-\sin\alpha+\beta,\ \cos\alpha+\beta,\ l\cos\beta]\, R(\theta)\dot{\xi} - \dot{\varphi} = 0,\ \text{rolling} \\ [\cos\alpha+\beta,\ \sin\alpha+\beta,\ l\sin\beta]\, R(\theta)\dot{\xi} = 0,\ \text{no-sliding} \end{cases} \tag{A.5}$$

For the sake of simplicity, we denote $J_1(\beta_s) = [-\sin\alpha+\beta,\ \cos\alpha+\beta,\ l\cos\beta]$

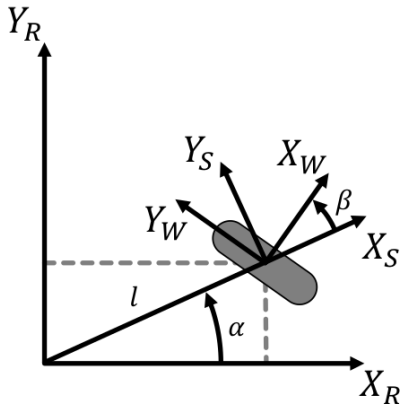


Figure A.2: Reference frames for deriving wheel equation for a standard wheel.

and $C_1(\beta_s) = [\cos\alpha + \beta,\ \sin\alpha + \beta,\ l\sin\beta]$.

A.2 Differential Kinematics

Given a wheeled robot, each wheel imposes n constraints. Notably, only fixed and steerable standard wheels impose no-sliding constraints. For a robot with n wheels of radius r, these individual wheel constraints can be concatenated in matrix form as: Rolling constraints

$$J_1(\beta_s)R(\theta)\dot{\xi}_I - J_2\dot{\varphi} = 0,\ J_1(\beta_s) = \begin{bmatrix} J_{1f} \\ J_{1s}(\beta_s) \end{bmatrix},\ J_2 = \mathrm{diag}(r_1, \cdots, r_n),\ \dot{\varphi} = \begin{bmatrix} \dot{\varphi}_1 \\ \vdots \\ \dot{\varphi}_n \end{bmatrix}$$

No-sliding constraints

$$C_1(\beta_S)R(\theta)\dot{\xi}_I = 0,\ C_1(\beta_S) = \begin{bmatrix} C_{1f} \\ C_{1S}(\beta_S) \end{bmatrix}$$

Combining the rolling and no-sliding constraints, as shown in Equation (A.6), provides an expression for the differential kinematics. Solving this equation for $\dot{\xi}_I$ yields the forward differential kinematics equation, which is essential for computing wheel odometry. Conversely, solving the equation for $\dot{\varphi}$ produces the inverse differential

kinematics equation, which is crucial for control purposes.

$$\begin{bmatrix} J_1(\beta_s) \\ C_1(\beta_s) \end{bmatrix} R(\theta)\dot{\xi}_I = \begin{bmatrix} J_2 \\ 0 \end{bmatrix} \dot{\varphi} \tag{A.6}$$

With these foundational equations established, we can now proceed to derive the forward and inverse kinematics of our differential drive robot.

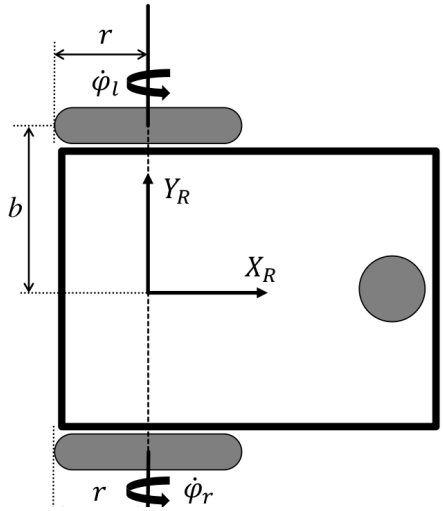


Figure A.3: Schematic of a Differential Drive Robot. The figure illustrates the geometry and kinematics of a differential drive robot with two wheels of radius r, separated by a distance b. The left and right wheel angular velocities are denoted by $\dot{\varphi}_l$ and $\dot{\varphi}_r$, respectively. The coordinate frame X_R and Y_R represents the robot's reference frame.

From Figure A.3, we can determine the parameters for the constraint equation, Equation (A.5), for each wheel. For the right wheel: $\alpha = -\frac{\pi}{2},\ \beta = 0,\ l = b$. For the left wheel: $\alpha = -\frac{\pi}{2},\ \beta = 0,\ l = -b$.

Rolling constraint becomes

$$\begin{bmatrix} 1 & 0 & b \\ 1 & 0 & -b \end{bmatrix} \dot{\xi}_R = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix} \begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix} \tag{A.7}$$

No-sliding constraint

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \dot{\xi}_R = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{A.8}$$

By stacking Equation (A.7) and Equation (A.8), we obtain the overall constraint equation for the robot.

$$\begin{bmatrix} 1 & 0 & b \\ 1 & 0 & -b \\ 0 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \dot{\xi}_R = \begin{bmatrix} r & 0 \\ 0 & r \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix} \tag{A.9}$$

We will consider the simplified stacked equation of motion $A\dot{\xi}_R = B\dot{\varphi}$.

Where: $A = \begin{bmatrix} 1 & 0 & b \\ 1 & 0 & -b \\ 0 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$, $B = \begin{bmatrix} r & 0 \\ 0 & r \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

The forward differential kinematics equation is obtained by solving for $\dot{\xi}_I$. However, since matrix A is non-square, we cannot solve for it directly. Instead, we solve for the pseudo-inverse of matrix A. Consequently, the forward kinematics of our differential drive robot is given by:

$$\dot{\xi}_R = (A^T A)^{-1} A^T B \dot{\varphi} \tag{A.10}$$

By substituting the values of matrices A and B, the solution to Equation (A.10), *i.e.*, the forward kinematics, becomes:

$$\dot{\xi}_R = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2b} & -\frac{r}{2b} \end{bmatrix} \dot{\varphi} \tag{A.11}$$

- Forward velocity: $\dot{x} = r\frac{\dot{\varphi}_r + \dot{\varphi}_l}{2}$
- No-sliding constraint results in: $\dot{y} = 0$
- Angular velocity: $\dot{\theta} = r\frac{\dot{\varphi}_r - \dot{\varphi}_l}{2b}$

Similarly, the inverse kinematics is obtained by solving for $\dot{\varphi}$. Since matrix B is also not square, we need to solve for the pseudo-inverse of matrix B.

$$\dot{\varphi} = (B^T B)^{-1} B^T A \dot{\xi}_R \tag{A.12}$$

$$\dot{\varphi} = \begin{bmatrix} \frac{1}{r} & 0 & \frac{b}{r} \\ \frac{1}{r} & 0 & -\frac{b}{r} \end{bmatrix} \dot{\xi}_R \tag{A.13}$$

Appendix B

TD7

In this section we provide a summary of the TD7 algorithm taken from Fujimoto et al. [24]. TD7 is consists of the following sub-components:

- Two value functions $Q_{t+1,1},\ Q_{t+1,2}$ and two target value functions $Q_{t,1},\ Q_{t,2}$
- A policy network π_{t+1} and a target policy network π_t
- An encoder, with sub-components f_{t+1}, g_{t+1}
- A fixed encoder, with sub-components f_t, g_t
- A target fixed encoder with sub-components f_{t-1}, g_{t-1}
- A checkpoint policy pi_c and checkpoint encoder f_c

Encoder: The encoder is composed of two sub-networks $(f_{t+1}(s), g_{t+1}(z^s, a))$, where each network outputs an embedding:

$$z^s \coloneqq f(s) \quad z^{sa} \coloneqq g(z^s, a) \tag{B.1}$$

At each training step, the encoder is updated with MSE loss:

$$\begin{aligned} \mathcal{L} &\coloneqq \left(g_{t+1}(f_{t+1}(s), a) - |f_{t+1}(s')|_{\mathrm{x}}\right)^2 \\ &= \left(z^{sa}_{t+1} - |z^{s'}_{t+1}|_{\mathrm{x}}\right)^2 \end{aligned} \tag{B.2}$$

Where $|\cdot|_{\mathrm{x}}$ is the stop-gradient operation.

Value Function: TD7 uses a two value functions $(Q_{t+1}, 1, Q_{t+1}, 2)$, each taking

input $\left[z_{t-1}^{s'a'}, z_{t-1}^{s'}, s', a'\right]$. At each training step, both value functions are updated with the following loss:

$$\begin{aligned}
\mathcal{L}(Q_{t+1}) &\coloneqq \texttt{Huber}\left(\texttt{target} - Q_{t+1}(z_t^{sa}, z_t^s, s, a)\right) \\
\texttt{target} &\coloneqq r + \gamma \texttt{clip}\left(\texttt{min}(Q_{t,1}(x), Q_{t,2}(x)), Q_{\min}, Q_{\max}\right) \\
x &\coloneqq \left[z_{t-1}^{s'a'}, z_{t-1}^{s'}, s', a'\right] \\
a' &\coloneqq \pi_t(z_{t-1}^{s'}, s') + \epsilon \\
\epsilon &\coloneqq \texttt{clip}(\mathcal{N}(0, \sigma^2), -c, c)
\end{aligned} \tag{B.3}$$

Taking the minimum of the value functions is from TD3's Clipped Double Q-learning [21]. The use of Huber loss [36] is in accordance to TD3 with the Loss-Adjusted Prioritized (LAP) experience replay [23]. a' is sampled and clipped in the same manner as TD3. The same embeddings (z_t^{sa}, z_t^s) are used for each value function. $Q_{\min}$ and $Q_{\max}$ are updated at each time step with:

$$\begin{aligned}
Q_{\min} &\leftarrow \texttt{min}(Q_{\min}, \texttt{target}) \\
Q_{\max} &\leftarrow \texttt{max}(Q_{\max}, \texttt{target})
\end{aligned} \tag{B.4}$$

Where `target` is defined by Equation B.3.

Policy: TD7 uses a single policy network which takes input $[z^s, s]$. On every second training step (delayed policy updates) the policy π_{t+1} is updated with the following loss:

$$\begin{aligned}
\mathcal{L}(\pi_{t+1}) &\coloneqq -\texttt{mean}\left(Q_{t+1,1}(x), Q_{t+1,2}(x)\right) \\
x &\coloneqq [z_t^{sa_\pi}, z_t^s, s, a_\pi] \\
a_\pi &\coloneqq \pi_{t+1}\left(z_t^s, s\right)
\end{aligned} \tag{B.5}$$

After every `target_update_frequency` training steps, the iteration is updated and each target (and fixed) network copies the network of the higher iteration:

$$\begin{aligned}
(Q_{t,1}, Q_{t,2}) &\leftarrow (Q_{t+1,1}, Q_{t+1,2}) \\
\pi_t &\leftarrow \pi_{t+1} \\
(f_{t-1}, g_{t-1}) &\leftarrow (f_t, g_t) \\
(f_t, g_t) &\leftarrow (f_{t+1}, g_{t+1})
\end{aligned} \tag{B.6}$$

LAP: Gathered experience is stored in a replay buffer [55] and sampled according to LAP [23], a prioritized replay buffer D [72] where a transition tuple $i := (s, a, r, s')$ is sampled with probability:

$$p(i) = \frac{\texttt{max}(|\sigma(i)|^\alpha, 1)}{\Sigma_{j \in D} \texttt{max}(|\sigma(j)|^\alpha, 1)}$$
$$|\sigma(i)| := \texttt{max}\left(|Q_{t+1,1}(z_t^{sa}, z_t^s, s, a) - \texttt{target}|, |Q_{t+1,2}(z_t^{sa}, z_t^s, s, a) - \texttt{target}|\right) \tag{B.7}$$

Where target is defined by Equation B.3.

As suggested by Fujimoto et al. [23], $|\sigma(i)|$ is defined by the maximum absolute error of both value functions. The amount of prioritization used is controlled by a hyperparameter α. New transitions are assigned the maximum priority of any sample in the replay buffer.

Finally, the TD7 train function is given by Algorithm 2

Algorithm 2 TD7 Train Function

```
Sample transition from LAP replay buffer with probability (Equation B.7)
Train encoder (Equation B.2)
Train value function (Equation B.3)
Update (Q_min, Q_max) (Equation B.4)
if i mod policy_update_frequency == 0 then
    Train policy (Equation B.5)
end if
if i mod target_update_frequency == 0 then
    Update target networks (Equation B.6)
end if
```

B.1 Hyperparameters

TD7 hyperparameters and their values used in the implementation is given in Table B.1.

	Hyperparameter	Value
Common	Discount factor γ	0.99999
	Replay buffer capacity	1×10^6
	Batch size	256
	Target update frequency	250
Optimizer	Optimizer	Adam [40]
	Learning rate	3×10^{-4}
TD3 [21]	Target policy noise σ	$\mathcal{N}(0, 0.2^2)$
	Target policy noise clipping c	$(-0.5, 0.5)$
	Policy update frequency	2
LAP [23]	Probability smoothing α	0.4
	Minimum priority	1
Exploration	Initial random exploration time steps	10×10^3
	Initial exploration noise	$\mathcal{N}(0, 1^2)$
	Minimum exploration noise	$\mathcal{N}(0, 0.1^2)$
	Exploration noise decay steps	5×10^5
Policy Checkpoints	Checkpoint criteria	`minimum`
	Early assessment episodes	1
	Late assessment episodes	20
	Early time steps	75×10^4
	Criteria reset weight	0.9

Table B.1: TD7 Hyperparameters [24].

Bibliography

[1] Mary B. Alatise and Gerhard P. Hancke. A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, 8:39830–39846, 2020. doi: 10.1109/ACCESS.2020.2975643. 1.1

[2] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Joshua Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39:20 – 3, 2018. URL https://api.semanticscholar.org/CorpusID:51894399. 2.4

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017. doi: 10.1109/MSP.2017.2743240. 3.1, 3.1.1

[4] Leemon C Baird. Advantage updating. Technical report, Citeseer, 1993. 3.1

[5] Fethi Belkhouche. Reactive path planning in a dynamic environment. *IEEE Transactions on Robotics*, 25(4):902–911, 2009. doi: 10.1109/TRO.2009.2022441. 1

[6] Richard Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957. URL https://api.semanticscholar.org/CorpusID:123329493. 3.1

[7] Berta Bescos, José M. Fácil, Javier Civera, and José Neira. Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018. doi: 10.1109/LRA.2018.2860039. 2.2.2, 2.3, B.1

[8] Berta Bescos, Carlos Campos, Juan D. Tardós, and José Neira. Dynaslam ii: Tightly-coupled multi-object tracking and slam. *IEEE Robotics and Automation Letters*, 6(3):5191–5198, 2021. doi: 10.1109/LRA.2021.3068640. 2.2.2, 2.4, B.1

[9] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991. doi: 10.1109/70.88137. 2.3.2

[10] Kuanqi Cai, Chaoqun Wang, Jiyu Cheng, Clarence W. de Silva, and Max Q.-H. Meng. Mobile robot path planning in dynamic environments: A survey. *ArXiv*, abs/2006.14195, 2020. URL https://api.semanticscholar.org/CorpusID:220055835. 2.3

[11] Sean Campbell, Niall O'Mahony, Anderson Carvalho, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Path planning techniques for mobile robots a review. In *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*, pages 12–16, 2020. doi: 10.1109/ICMRE49073.2020.9065187. 2.3.1, 2.3.2

[12] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6): 1874–1890, 2021. doi: 10.1109/TRO.2021.3075644. 2.2.2

[13] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022, 2019. doi: 10.1109/ICRA.2019.8794134. 2.1, 2.4

[14] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 285–292, 2017. doi: 10.1109/ICRA.2017.7989037. 2.1, 2.4

[15] Reinis Cimurs, Il Hong Suh, and Jin Han Lee. Goal-driven autonomous exploration through deep reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):730–737, 2022. doi: 10.1109/LRA.2021.3133591. 4

[16] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013. 3.1

[17] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. URL https://api.semanticscholar.org/CorpusID:123284777. 2.3.1

[18] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part i. *IEEE Robotics and Automation Magazine*, 13(2):99–110, June 2006. 2.2

[19] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014. doi: 10.1109/ACCESS.2014.2302442. 2.3, 2.3.1

[20] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997. doi: 10.1109/100.580977. 2.3.2

[21] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018. 1, 3.5, B, ??

[22] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2018. URL https://api.semanticscholar.org/CorpusID:54457299. 3.5.2

[23] Scott Fujimoto, David Meger, and Doina Precup. An equivalence between loss functions and non-uniform sampling in experience replay. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546. B, B, B, ??

[24] Scott Fujimoto, Wei-Di Chang, Edward James Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For sale: State-action representation learning for deep reinforcement learning. *ArXiv*, abs/2306.02451, 2023. URL https://api.semanticscholar.org/CorpusID:259075901. 1, 1.3, 3.5, 3.5.1, 3.5.1, 3.5.2, 3.5.2, B, B.1, B.1

[25] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014. doi: 10.1109/IROS.2014.6942976. 2.3.1

[26] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *ArXiv*, abs/1906.02736, 2019. URL https://api.semanticscholar.org/CorpusID:174800787. 3.5.2

[27] Antoni Grau, Marina Indri, Lucia Lo Bello, and Thilo Sauter. Industrial robotics in factory automation: From the early stage to the internet of things. In *IECON 2017-43rd annual conference of the IEEE industrial electronics society*, pages 6159–6164. IEEE, 2017. 1.1

[28] Antoni Grau, Marina Indri, Lucia Lo Bello, and Thilo Sauter. Robots in industry: The past, present, and future of a growing collaboration with humans. *IEEE Industrial Electronics Magazine*, 15(1):50–61, 2021. doi: 10.1109/MIE.2020.3008136. 1.1

[29] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2432–2437, 2005. doi: 10.1109/ROBOT.2005.1570477. 2.2.1

[30] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques

for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007. doi: 10.1109/TRO.2006.889486. 2.2.1

[31] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. 2.3.1

[32] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, May 1995. ISSN 1095-3787. doi: 10.1103/physreve.51.4282. URL http://dx.doi.org/10.1103/PhysRevE.51.4282. 2.1

[33] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016. doi: 10.1109/ICRA.2016.7487258. 2.2.1

[34] Ronald A. Howard. Dynamic programming and markov processes. 1960. URL https://api.semanticscholar.org/CorpusID:62124406. 3.1

[35] Xiaomei Hu, Luying Zhu, Ping Wang, Haili Yang, and Xuan Li. Improved orb-slam2 mobile robot vision algorithm based on multiple feature fusion. *IEEE Access*, 11:100659–100671, 2023. doi: 10.1109/ACCESS.2023.3315326. 2.2

[36] Peter J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35:492–518, 1964. URL https://api.semanticscholar.org/CorpusID:121252793. B

[37] Tete Ji, Chen Wang, and Lihua Xie. Towards real-time semantic rgb-d slam in dynamic environments. In *Proceedings of (ICRA) International Conference on Robotics and Automation*. IEEE, May 2021. 2.2.2

[38] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101 (1-2):99–134, 1998. 3.1

[39] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985. doi: 10.1109/ROBOT.1985.1087247. 2.3.2

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106. ??

[41] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2022. doi: 10.1109/TITS.2021.3054625. 2.4

[42] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727. 1.5

[43] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE, 2004. 3.1.1

[44] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011. 2.2.1

[45] Vijay R. Konda and John N. Tsitsiklis. Onactor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003. doi: 10.1137/S0363012901385691. URL https://doi.org/10.1137/S0363012901385691. 3.1

[46] Kurt Konolige, Giorgio Grisetti, Rainer Kümmerle, Wolfram Burgard, Benson Limketkai, and Regis Vincent. Efficient sparse pose adjustment for 2d mapping. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 22–29, 2010. doi: 10.1109/IROS.2010.5649043. 2.2.1

[47] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000. doi: 10.1109/ROBOT.2000.844730. 2.3.1

[48] Jean-Claude Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research*, 18 (11):1119–1128, 1999. 2.3

[49] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998. URL https://api.semanticscholar.org/CorpusID:14744621. 2.3.1

[50] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17 (39):1–40, 2016. 3.1.1

[51] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37 (4-5):421–436, 2018. 3.1.1

[52] Haoran Li, Qichao Zhang, and Dongbin Zhao. Deep reinforcement learning-based automatic exploration for navigation in unknown environment. *IEEE Transactions on Neural Networks and Learning Systems*, 31(6):2064–2076, 2020. doi: 10.1109/TNNLS.2019.2927869. 2.4

[53] Siding Li, Xin Xu, and Lei Zuo. Dynamic path planning of a mobile robot with improved q-learning algorithm. In *2015 IEEE International Conference on Information and Automation*, pages 409–414, 2015. doi: 10.1109/ICInfA.2015.7279322. 2.3

[54] Xiaoxiao Li, Xiaoguang Hu, Ziqiang Wang, and Zhuoqun Du. Path planning based on combinaion of improved a-star algorithm and dwa algorithm. In *2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, pages 99–103, 2020. doi: 10.1109/AIAM50918.2020.00025. 2.3.2

[55] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3–4):293–321, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699. B

[56] Anbalagan Loganathan and Nur Syazreen Ahmad. A systematic review on recent advances in autonomous mobile robot navigation. *Engineering Science and Technology, an International Journal*, 40:101343, 2023. ISSN 2215-0986. doi: https://doi.org/10.1016/j.jestch.2023.101343. URL https://www.sciencedirect.com/science/article/pii/S2215098623000204. 2.3

[57] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783, 2021. doi: 10.21105/joss.02783. URL https://doi.org/10.21105/joss.02783. 2.2, 2.2, 4.4, B.1

[58] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074. URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074. 1.5

[59] Christoforos Mavrogiannis, Francesca Baldini, Allan Wang, Dapeng Zhao, Pete Trautman, Aaron Steinfeld, and Jean Oh. Core challenges of social robot navigation: A survey. *J. Hum.-Robot Interact.*, 12(3), apr 2023. doi: 10.1145/3583741. URL https://doi.org/10.1145/3583741. 1.1

[60] Kimberly N. McGuire, Guido C.H.E. de Croon, and Karl Tuyls. A comparative study of bug algorithms for robot navigation. *ArXiv*, abs/1808.05050, 2018. URL https://api.semanticscholar.org/CorpusID:52008963. 2.3.2

[61] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013. URL https://api.semanticscholar.org/CorpusID:15238391. 2.4

[62] Raúl Mur-Artal and Juan D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33 (5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103. 2.2.2

[63] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5): 1147–1163, 2015. doi: 10.1109/TRO.2015.2463671. 2.2.2

[64] Saeid Nahavandi, Roohallah Alizadehsani, Darius Nahavandi, Shady M. K. Mohamed, Navid Mohajer, Mohammad Rokonuzzaman, and Ibrahim Hossain. A comprehensive review on autonomous navigation. *ArXiv*, abs/2212.12808, 2022. URL https://api.semanticscholar.org/CorpusID:255125287. 2.2, 2.2.1

[65] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX: The 9th international symposium on experimental robotics*, pages 363–372. Springer, 2006. 3.1.1

[66] Utsav Patel, Nithish K Sanjeev Kumar, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6057–6063, 2021. doi: 10.1109/ICRA48506.2021.9561462. 2.4

[67] Phuwanat Phueakthong, Jittima Varagul, and Nattawat Pinrath. Deep reinforcement learning based mobile robot navigation in unknown environment with continuous action space. In *2022 5th International Conference on Intelligent Autonomous Systems (ICoIAS)*, pages 154–158, 2022. doi: 10.1109/ICoIAS56028.2022.9931272. 2.4

[68] Jie Qi, Hui Yang, and Haixin Sun. Mod-rrt*: A sampling-based algorithm for robot path planning in dynamic environment. *IEEE Transactions on Industrial Electronics*, 68(8):7244–7251, 2021. doi: 10.1109/TIE.2020.2998740. 1

[69] Ankit A. Ravankar, Abhijeet Ravankar, Takanori Emaru, and Yukinori Kobayashi. Hpprm: Hybrid potential based probabilistic roadmap algorithm for improved dynamic path planning of mobile robots. *IEEE Access*, 8: 221743–221766, 2020. doi: 10.1109/ACCESS.2020.3043333. 1

[70] Xiaogang Ruan, Dingqi Ren, Xiaoqing Zhu, and Jing Huang. Mobile robot navigation based on deep reinforcement learning. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 6174–6178, 2019. doi: 10.1109/CCDC.2019.8832393. 2.4

[71] Gavin Adrian Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. 1994. URL https://api.semanticscholar.org/CorpusID:59872172. 3.1

[72] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015. URL https://api.semanticscholar.org/CorpusID:13022595. B

[73] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 3.1

[74] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 2.4

[75] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017. URL https://api.semanticscholar.org/CorpusID:33081038. 2.4

[76] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16:105–133, 2002. 3.1.1

[77] Bruno Steux and Oussama El Hamzaoui. tinyslam: A slam algorithm in less than 200 lines c-language program. In *2010 11th International Conference on Control Automation Robotics & Vision*, pages 1975–1979, 2010. doi: 10.1109/ICARCV.2010.5707402. 2.2.1

[78] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888, 2006. 3.1.1

[79] Yuxiang Sun, Ming Liu, and Max Q.-H. Meng. Improving rgb-d slam in dynamic environments: A motion removal approach. *Robotics and Autonomous Systems*, 89:110–122, 2017. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2016.11.012. URL https://www.sciencedirect.com/science/article/pii/S0921889015302232. 2.2.2

[80] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. ISBN 9780262039246. 3.1, 3.1, B.1

[81] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006. doi: 10.1177/0278364906065387. URL https://doi.org/10.1177/0278364906065387. 2.2

[82] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents. The MIT Press, Cambridge, 2006. ISBN 9780262201629. URL https://worldcat.org/title/1423981945. 2.2

[83] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803, 2010. doi: 10.1109/IROS.2010.5654369. 1.1, 2.1

[84] Xuan-Tung Truong and Trung Dung Ngo. Toward socially aware robot navigation in dynamic and crowded environments: A proactive social motion model. *IEEE Transactions on Automation Science and Engineering*, 14(4):1743–1760, 2017. doi: 10.1109/TASE.2017.2731371. 1

[85] Victor Uc-Cetina, Nicolás Navarro-Guerrero, Anabel Martin-Gonzalez, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *Artificial Intelligence Review*, 56(2):1543–1575, 2023. 2.4

[86] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008. doi: 10.1109/ROBOT.2008.4543489. 2.1, 2.1, B.1

[87] William Yang Wang, Jiwei Li, and Xiaodong He. Deep reinforcement learning for nlp. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 19–21, 2018. 2.4

[88] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. URL https://api.semanticscholar.org/CorpusID:208910339. 3.1

[89] Binbin Xu, Wenbin Li, Dimos Tzoumanikas, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. Mid-fusion: Octree-based object-level multi-instance dynamic slam. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5231–5237, 2019. doi: 10.1109/ICRA.2019.8794371. 2.2.2, 2.2.2

[90] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Comput. Surv.*, 55(1), nov 2021. ISSN 0360-0300. doi: 10.1145/3477600. URL https://doi.org/10.1145/3477600. 2.4

[91] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021. doi: 10.26599/TST.2021.9010012. 2.2, 2.4, 2.6, B.1

[92] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364, 2017. doi: 10.1109/ICRA.2017.7989381.

2.4

[93] Cezary Zieliński. *Robotic System Design Methodology Utilising Embodied Agents*, pages 523–561. Springer International Publishing, Cham, 2021. ISBN 978-3-030-48587-0. doi: 10.1007/978-3-030-48587-0_17. URL https://doi.org/10.1007/978-3-030-48587-0_17. 3.2

[94] C. Zieliński and T. Winiarski. General specification of multi-robot control system structures. *Bulletin of the Polish Academy of Sciences Technical Sciences*, 58 (No 1):15–28, 2010. doi: 10.2478/v10175-010-0002-x. URL http://journals.pan.pl/Content/82981/PDF/02_paper.pdf. 3.2

List of Symbols and Abbreviations

The next list describes several symbols that are used within the body of the document

Abbreviations

APF	Artificial Potential Field
DRL	Deep Reinforcement Learning
DWA	Dynamic Window Approach
EKF	Extended Kalman Filter
FOV	Field of View
LAP	Loss- Adjusted Prioritized Experience Replay
LiDAR	Light Detection and Ranging
MDPs	Markov Decision Processes
MSE	Mean Squared Error
ORB	Oriented FAST and Rotated BRIEF)
POMDPs	Partially Observable Markov Decision Processes
RL	Reinforcement Learning
ROS2	Robot Operating System 2
RRT	Rapid Exploring Random Tree
RRT	Rapidly-exploring Random Tree
SLAM	Simultaneous Localization and Mapping
TD3)	Twin Delayed Deep Deterministic Policy Gradient
TD7	TD3 with four additions
vSLAM	Visual Simultaneous Localization and Mapping

Symbols

γ	Discount Factor
$\mathbb{E}$	Expectation

$\mathcal{A}$	Action Space
$\mathcal{P}$	Environment one-step Dynamics
$\mathcal{R}$	Reward Signal
$\mathcal{S}$	State Space
ω	Angular Velocity
π	Policy
π^*	Optimal Policy
θ	Relative Heading
A	Advantage Function
d	Distance from Target Goal
f	State Encoder
g	State-Action Encoder
Q	Action-Value Function
Q^*	Optimal Action-Value Function
V	State-Value Function
v	Linear Velocity
V^*	Optimal State-Value Function
z^{sa}	State-Action Embedding
z^s	State Embedding

List of Figures

List of Tables