Top International Managers in Engineering

University of Padua
Department of Information Engineering
MSc in Control Systems Engineering

Technical University of Denmark
Department of Applied Mathematics and Computer Science
MSc in Computer Science and Engineering

Huawei's Munich Research Center
Advanced Wireless Technologies Lab
Robotic Taskforce Group

Master Thesis in:

Cooperative Carrying Control for Mobile Robots in Indoor Scenario

Supervisor:
Prof. Maria Elena Valcher

Student:
Alessandro Canevaro

Co-Supervisors:
Prof. Xenofon Fafoutis
Dr. Hanwen Cao
Dr. Massimiliano Maule

Academic Year: 2022 - 2023
Graduation Date: July 13th

Cooperative Carrying Control for Mobile Robots in Indoor Scenario

Master Thesis
June, 2023

By
Alessandro Canevaro

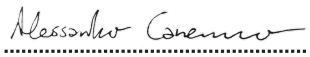
Approval

This master thesis was prepared over a period of five months at the Advanced Wireless Technologies Lab, located at Huawei's Research Center in Munich, Germany.

It is submitted in partial fulfilment of the requirements for the Master of Science degree in Control Systems Engineering at the University of Padua, Italy, and the Master of Science degree in Computer Science and Engineering at the Technical University of Denmark, as part of the Top International Managers in Engineering double degree program.

The thesis encompasses a comprehensive exploration of cutting-edge topics, including Robotics, Reinforcement Learning, Control and Sensing systems, and 5/6G Communications.

Alessandro Canevaro



Signature

14/06/2023

Date

Acknowledgements

I am immensely grateful to my esteemed advisor, Professor Maria Elena Valcher, whose support and advice have been indispensable throughout the entirety of my master's program. Her expertise and endless patience have played a crucial role in the success of my academic journey, and her personal guidance regarding my future pursuits has been priceless.

Likewise, I extend my heartfelt appreciation to Professor Xenofon Fafoutis's mentorship, for his assistance and sage advice during the composition of my master's thesis. His wisdom and feedback have not only augmented but elevated the overall caliber and profundity of my research.

I am sincerely grateful to the Advanced Wireless Technology Lab for granting me the opportunity to conduct my research and for all of the resources and support they generously provided. I would like to express my thanks to Dr. Hanwen Cao for his invaluable feedback and suggestions. Moreover, I extend a special acknowledgement to Dr. Massimiliano Maule, who went above and beyond in aiding me in this work with his exceptional assistance. His insights and guidance were fundamental in shaping the trajectory of my research.

To my beloved parents, I am deeply thankful for their unwavering love and support throughout this transformative journey. Without their constant encouragement and motivation, I would not have had the strength to get through the challenges and complete this academic voyage.

Last but certainly not least, I extend my sincere gratitude to all the exceptional colleagues and friends I encountered during these years. Their willingness to share their experiences and insights has substantially contributed to both the refinement of my professional and personal growth, making this entire process even more enriching and rewarding.

List of Figures

List of Tables

Acronyms

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
AMCL	Adaptive Monte Carlo Localization
APF	Artificial Potential Fields
BFS	Breadth First Search
CL	Curriculum Learning
CNN	Convolutional Neural Network
CoBots	Collaborative Robots
CPS	Cyber Physical Systems
DFS	Depth First Search
DNN	Deep Neural Network
DOF	Degree Of Freedom
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
eMBB	Enhanced Mobile Broadband
GAE	Generalized Advantage Estimator
IMU	Inertial Measurement Unit
InF	Indoor Factory
IoT	Internet of Things
ISAC	Integrated Sensing and Communication
KF	Kalman Filter
LIDAR	Light Detection And Ranging
LOS	Line-Of-Sight
MARL	Multi-Agent Reinforcement Learning
MCL	Monte Carlo Localization
MDP	Markov Decision Process
ML	Machine Learning
mMTC	Massive Machine Type Communications
MPC	Model Predictive Control
MRS	Multi-Robot System
MVProp	Max-Propagation
NLOS	Non-Line-Of-Sight
NN	Neural Network

PF Particle Filter
PID Proportional Integral Derivative
PPO Proximal Policy Optimization
PRM Probabilistic Road Map

RADAR Radio Detection And Ranging
RHC Receding Horizon Control
RL Reinforcement Learning
ROS Robot Operating System
RRT Rapid-exploring Random Tree

SAC Soft Actor-Critic
SB3 Stable Baseline 3
SDF Spatial Data File
SLAM Simultaneous Localization And Mapping
SoA State-of-the-Art
SONAR Sound Navigation And Ranging
STL Standard Tessellation Language

TDOA Time Difference Of Arrivals

UAV Unmanned Aerial Vehicles
URLLC Ultra Reliability and Low Latency Communications

VI Value Iteration
VIN Value Iteration Network
VO Visual Odometry
VPN Value Propagation Network
VProp Value Propagation

Abstract

In recent years, there has been a growing interest in designing multi-robot systems to provide cost-effective, fault-tolerant and reliable solutions to a variety of automated applications. In particular, from an industrial perspective, cooperative carrying techniques based on Reinforcement Learning (RL) gained a strong interest. Compared to a single robot system, this approach improves the system's robustness and manipulation dexterity in the transportation of large objects. However, in the current state of the art, the environments' dynamism and re-training procedure represent a considerable limitation for most of the existing cooperative carrying RL-based solutions.

In this thesis, we employ the Value Propagation Network (VPN) algorithm for cooperative multi-robot transport scenarios. We extend and test the ΔQ cooperation metric to V-value-based agents, and we investigate path generation algorithms and trajectory tracking controllers for differential drive robots. Moreover, we explore localization algorithms in order to take advantage of range sensors and mitigate the drift errors of wheel odometry, and we conduct experiments to derive key performance indicators of range sensors' precision. Lastly, we perform realistic industrial indoor simulations using Robot Operating System (ROS) and Gazebo 3D visualization tool, including physical objects and 6G communication constraints.

Our results showed that the proposed VPN-based algorithm outperforms the current state-of-the-art since the trajectory planning and dynamic obstacle avoidance are performed in real-time, without re-training the model, and under constant 6G network coverage.

Contents

1 Introduction

1.1 Motivation

In recent years, there has been a notable upsurge in the deployment of robots within the manufacturing industry, encompassing sectors such as automotive, electrical, metal, and machinery. Projections indicate that this growth trajectory is likely to persist in the coming years, underlining the profound impact of robotics in this domain. Notably, a prime application for robots is in the transportation and manipulation of large and heavy objects, an arduous task often beyond the capabilities of human workers [1]. To address these complex tasks, the adoption of Multi-Robot Systems (MRSs), specifically cooperative ones, has emerged as a successful concept. Cooperative MRSs have long been a focal point in the field of robotics, as they offer several advantages compared to single robots, and they broaden the spectrum of tasks that can be accomplished by a robotic system.

Concurrently with the ongoing proliferation of robots in industrial settings, two additional catalysts, namely Machine Learning (ML) and 6G technologies, have recently emerged displaying promising potential in addressing the long-standing challenges faced by MRSs. As a result, MRSs have lately garnered considerable attention and generated significant interest within both the scientific and industrial communities.

The primary challenge of MRSs lies in their efficient coordination and control. Traditional methodologies, owing to the intricate dynamics involved, frequently fall short and face considerable difficulties when applied in this context. However, the advent of ML, which has pervaded the field of robotics, offers novel opportunities. These pioneering techniques hold immense potential in bolstering the autonomy of robots and, more precisely, in tackling the planning and coordination challenges encountered within MRSs.

The stringent communication and perception requirements pose additional challenges in the realm of MRSs. Indeed, to attain remarkable outcomes in terms of planning and coordination, these two other aspects mandate a strong focus. In the domain of robotics, telecommunications have already received considerable attention with the advent of 5G technology. Similarly, perception capabilities continue to be refined through the introduction of increasingly sophisticated sensing devices. Nevertheless, MRSs still present numerous challenges in these domains. Consequently, these systems have emerged as a prominent research area in the field of telecommunications, particularly in the context of 6G technologies.

While acknowledging that ML and 6G technologies are still in the developmental phase and not yet mature for real-world deployment, their profound potential drives our motivation to bridge this gap. In this thesis, we present a pioneering solution to the challenge of cooperative carrying tasks within MRSs. Our objective is to specifically address the underlying reasons why existing State-of-the-Art (SoA) techniques have yet to find widespread adoption in industrial settings. By targeting these limitations, this research aims to contribute to the advancement of MRSs and pave the way for their practical implementation in the industry.

1.2 Scope and Key Contributions

Our primary research objective is to investigate the design and implementation of a multi-robot motion-planning algorithm for cooperative carrying. Existing literature lacks a comprehensive solution that effectively navigates in dynamic environments, where obstacles

may be not known in advance, and while also encompassing the advantages offered by Reinforcement Learning (RL) approaches. To address this gap, we propose a novel approach that achieves the desired objective through an adaptation of the Value Propagation Network (VPN) algorithm [2]. We conducted thorough testing and validation of our algorithm across a diverse range of scenarios, including those that closely resemble the ones encountered in the literature, as well as custom-generated environments that emulate indoor scenarios commonly encountered in industrial settings where the robots will operate.

We place significant emphasis on performance metrics, as they are crucial for evaluating the efficacy and capabilities of a solution, particularly when it involves RL algorithms. In line with this, we have extended the recently published novel metric, ΔQ, which quantifies the level of cooperation among robots. This extension enables its applicability to a wider range of RL algorithms.

We leveraged the 6G connectivity to enhance the proposed planning algorithm to address the low signal coverage problem, that manifests when the robot system moves inside the indoor scenario. To overcome this challenge, our solution integrates a radio coverage map into the planning algorithm. As a result, the robot's trajectory proactively avoids areas with insufficient coverage, ensuring seamless connectivity for the robots to accomplish their mission.

The research focus subsequently shifts towards conducting realistic simulations to evaluate the effectiveness of the developed solution. This entails an in-depth investigation of the path generation algorithm and trajectory tracking controllers for MRSs. To facilitate these simulations, we expanded upon the experimental platform originally constructed for testing the RL planner, incorporating additional components and 3D visualization tools. In order to effectively coordinate the simulations and enable seamless communication among the various components, we employed the Robot Operating System (ROS).

To enhance the realism of the simulations, we incorporated simulated sensor measurement noise and a localization algorithm. By introducing these elements, we aimed to emulate real-world conditions more accurately and account for the uncertainties and limitations typically encountered in robotic systems. Furthermore, leveraging this comprehensive framework, we conducted preliminary tests to validate the applicability and effectiveness of 6G sensing devices in robotic contexts.

1.3 Thesis Structure

This thesis is organized as follows. Chapter 2 serves as the foundation, providing a comprehensive technical background for the key concepts addressed in this study. We commence with a thorough overview of cooperative robotics and motion planning. Subsequently, we delve into the fundamental principles of RL. Following this, we explore path generation algorithms, trajectory tracking controllers, and localization strategies. Finally, we present an overview of how the relationship between telecommunications and robotics evolved over time. In Chapter 3, we conduct an in-depth analysis and discussion of the current SoA in MRSs motion planning and formation control, examining the existing methodologies and their advantages and limitations. Chapter 4 is divided into three principal sections. The first section delves into the process of modelling the real-world environment employing established techniques commonly utilized in the realm of RL. The second section introduces the proposed RL-based planning solution. The third section is devoted to designing the control system employed to interface the planning algorithm with the robotic systems, along with the development of a 3D simulation platform. In Chapter

5, we carefully analyze the results obtained from our solution, critically examining their implications. Lastly, in Chapter 6, we summarize the key findings of our work, while also providing valuable insights into potential future extensions and advancements.

2 Background Material

2.1 Cooperative Robotics

In this section, we aim to provide a concise introduction to the field of cooperative robotics. We shall begin by clarifying its fundamental definitions before delving into its advantages and disadvantages. Finally, we will present a comprehensive summary of the cooperative robotic taxonomy.

2.1.1 Definitions

The scientific literature exhibits inconsistencies regarding the definition of cooperative robotics, leading to two distinct interpretations.

The first definition stems from the MRS taxonomy, where cooperation is defined in opposition to competition. Specifically, a cooperative MRS encompasses multiple robots united by a shared objective, necessitating their interaction, communication, and coordination to accomplish it. Prominent tasks within cooperative MRSs involve exploration, search and rescue operations, as well as transportation. Conversely, competitive MRSs entail multiple robots competing against each other to achieve individual objectives, such as in zero-sum games like robot soccer leagues.

Within the context of this thesis, our focus lies specifically in cooperative carrying, which can be delineated as an MRS wherein robots coordinate and synchronize their actions to achieve a collective goal, specifically the transportation of an object from an initial to a target location. This definition, however, does not preclude scenarios where a single robot undertakes the object transportation, while others assume auxiliary roles in planning or sensory operations [3].

A second definition arises from the classification of Collaborative Robots (CoBots), which positions cooperative robots as an intermediary category between industrial robots and CoBots. In this categorization cooperative robots seek to combine the advantages of industrial robots with additional benefits conferred by safety sensors, enabling safe operation in proximity to humans [4].

2.1.2 Benefits and Challenges

The advantages of employing an MRS over a single robot are manifolds:

- Enhanced performance: MRSs demonstrate superior performance in terms of execution time and energy consumption. For instance, a task like exploration can be partitioned into sub-tasks, where each robot covers a smaller area. As a result, the total time required for completing the task is significantly reduced, thanks to the parallel execution of sub-tasks.
- Increased manipulation dexterity: MRSs enable improved manipulation capabilities, as exemplified by some cooperative carrying scenarios. The ability to grasp the object from multiple points simultaneously empowers the robots to execute intricate manoeuvres and more effectively handle large, fragile, or flexible objects.
- Enhanced reliability and fault tolerance: the redundant nature of MRSs potentially offers an increased reliability and fault tolerance to hardware and software failures. In the event of a unit failure, the remaining robots can reassign tasks among themselves to ensure the attainment of the common goal.

- Robustness through information sharing: MRSs can offer enhanced robustness by facilitating the sharing of redundant information among robots and enabling multi-robot sensor data fusion. This fosters improvements in localization and perception capabilities, bolstering the overall system performance.
- Cost-effectiveness: MRSs can be economically advantageous, as the utilization of multiple simple robots often proves less expensive than deploying a single complex robot.
- Overcoming complexity: MRSs prove crucial in accomplishing goals that are inherently complex and beyond the capabilities of a single robot, such as spatially separated tasks.

Nonetheless, the utilization of an MRS introduces notable challenges that necessitate careful consideration when opting to employ such systems. Several key areas require particular attention: firstly, the intricate coordination required by MRSs necessitates a revision and modification of traditional motion and path planning algorithms. The complexity of the system demands innovative approaches to ensure efficient and effective coordination among the robots. Secondly, MRSs may impose more stringent constraints on positioning, especially in scenarios like cooperative carrying. Consequently, improved sensing performance becomes essential to meet these requirements effectively. Communication stands as another critical factor that demands thorough attention within an MRS. Information sharing and synchronization among robots play a central role, requiring robust and efficient communication protocols to facilitate seamless coordination. Moreover, while MRSs offer the potential for increased reliability and fault tolerance if not properly addressed, failure risks can be higher than in single robot systems. In cases where all robots are indispensable for accomplishing the common goal, the failure of a single robot can lead to the failure of the entire system [3], [5].

2.1.3 Taxonomy

In this section, we will provide a concise overview of the prevailing categorizations found in the cooperative robotics literature.

In their study [6], the authors propose a taxonomy for MRS tasks, which is based on two key parameters: the dimension of the goal and the number of task iterations. The dimensionality of the goal defines the characteristics of the objective. For instance, if the goal involves achieving a specific pose configuration, it is considered a zero-dimensional goal. Conversely, if the objective is to follow a path, typically represented as a curve in space, it is referred to as a one-dimensional goal. This applies to tasks such as cooperative carrying or pattern formation. Tasks with higher-dimensional goals, such as exploration, involve targeting an area rather than a specific point or curve. The second parameter, the number of task iterations, pertains to the frequency at which the tasks need to be performed. Certain tasks necessitate multiple executions, as it happens with periodical region sweeping, foraging, or robot soccer.

In [3], the authors propose a taxonomy for differentiating between the many approaches employed by cooperative MRSs in tackling various tasks. The taxonomy primarily focuses on three key factors: coordination, communication, and decision-making. Coordination can be classified as either static or dynamic. Static coordination refers to situations where robots adhere to predefined rules (for example, traffic regulations). Conversely, dynamic coordination involves continuous information exchange among robots during task execution. Communication is divided into two categories: explicit and implicit. Explicit communication entails direct information exchange through unicast or broadcast messages, typ-

ically facilitated by dedicated onboard communication modules. Implicit communication, on the other hand, involves robots gathering information about other robots through their sensors and the environment. Decision-making encompasses the process of selecting actions that enable the robot to achieve its objectives. Cooperative MRS decision-making approaches can be broadly categorized into centralized and decentralized architectures. In centralized decision-making, a central robot or computer assumes the responsibility of making decisions for the entire system. This architecture can offer the advantage of optimal decision-making by leveraging comprehensive information about the environment and the whole MRS. However, it is susceptible to vulnerability in the event of failures. In contrast, decentralized decision-making entails individual robots autonomously taking actions. A hybrid approach known as decentralized hierarchical architecture also exists, combining elements of both centralized and decentralized decision-making approaches.

We will now present a classification of transportation methods for MRSs based on the review conducted in [5]. Cooperative carrying is a task that can be accomplished by a diverse range of robots, including mobile robots, manipulators, drones, and even heterogeneous MRS configurations. However, the following three categories are independent of the specific type of robot employed. The first category is the ”pushing-only” strategy, wherein the robots are not directly attached to the object to be transported. Instead, they rely solely on pushing movements to move the object. The second category is the ”caging” strategy. Similar to the pushing-only strategy, the robots are not physically attached to the object. However, in this approach, the robots are strategically positioned around the object, forming a ”cage”. This configuration ensures that the object follows the same movements as the MRS. Lastly, the ”grasping” strategy involves the robots physically attaching themselves to the object. This direct attachment enables the robots to exert better control over the object’s movements, enhancing precision and stability during transportation.

2.2 Motion Planning

As discussed in the preceding section, motion planning is a critical element of cooperative MRSs. It plays a central role in coordinating the robots, driving the decision-making process, and ultimately ensuring that the MRS achieves its objective. Before delving into an analysis of the current SoA in MRS motion planning, let us first review some fundamental concepts about planning.

2.2.1 What is Motion Planning?

Within the literature, motion planning is frequently used interchangeably with path and trajectory planning, which may result in some confusion [7]. To ensure clarity and prevent any misinterpretation, we will adhere to the following definitions.

Path planning algorithms operate at a topological level and are responsible for determining a geometric path that a robot should follow to reach its destination. These algorithms may take into account constraints imposed by the robot’s geometry. In contrast, trajectory planners focus on defining how the robot should move along the path in relation to time. Consequently, trajectory planners typically output reference velocities or accelerations that the robot should track. These outputs may be subject to constraints imposed by the robot’s kinematic properties. Both path planning and trajectory planning assume an empty workspace, meaning they do not consider any environmental information. On the other hand, motion planners are specifically designed to navigate robots through complex environments that may contain obstacles. They aim to find paths that enable the robot to safely manoeuvre in such environments. The output of motion planners is typically a geometric path or, more commonly, a sequence of waypoints that guide the robot towards

its destination. Due to the computational complexity involved, motion planning algorithms often operate in a discrete representation of the workspace [8].

To address the computational challenges and uncertainties of real-world scenarios, motion planners are commonly divided into two components: a global planner and a local planner. The global planner primarily relies on prior knowledge of the environment. It performs offline planning before the robot initiates its movement. This allows the planner to generate a high-level plan based on the known environment, taking into account factors such as the robot's destination and potential obstacles. However, relying solely on a precomputed plan can be insufficient in real-world scenarios, where uncertainties arise from factors like localization errors, environmental changes, or unknown obstacles. To compensate for these uncertainties, local planners are employed. Local planners are computationally lighter and designed for real-time (online) planning. They are specifically tailored to handle the uncertainties and dynamic changes encountered by the robot during its motion. Rather than considering the entire environment, local planners focus on a small neighbouring area around the robot. This localized approach allows for faster computation and adaptation to real-time conditions [9].

2.2.2 Single-robot Motion Planning

When evaluating motion planning algorithms, three key characteristics are commonly used to assess their effectiveness:

- Optimality: it pertains to the ability of an algorithm to generate paths that optimize a specific cost/reward function. Common cost (reward) functions in motion planning are related to the length of the path (to the distance from obstacles).
- Completeness: it refers to the assurance that the motion planning algorithm will always find a valid path, if one exists. A complete algorithm guarantees that, given enough time, it will either find a feasible path or determine that no path exists.
- Efficiency: it evaluates the computational performance of the algorithm in terms of time and space complexity. Motion planning algorithms need to be efficient to be applicable in real-time scenarios.

Numerous solutions have been proposed in the scientific literature to address these metrics. These solutions can be broadly categorized into two macrocategories: graph search-based algorithms and sampling-based algorithms. The first category primarily operates on a graph representation of the workspace, where discrete locations are represented as nodes and spatially adjacent locations are connected by links. The fundamental algorithms within this category include Depth First Search (DFS) and Breadth First Search (BFS). However, due to their high computational complexity, these algorithms are seldom utilized in practical scenarios. More efficient variants have been introduced, such as the Dijkstra algorithm and A*, which leverage heuristic functions to guide informed graph searches. Although these algorithms are complete and capable of finding the shortest path, their main limitation lies in their computational cost, particularly when dealing with planning in high-dimensional spaces [10]. The second class of algorithms aims to overcome these limitations by reducing the size of the graph used for the search. In this approach, the graph is constructed by randomly sampling the workspace and attempting to connect nodes until a viable path from the starting to the ending point is obtained. Among the most successful sampling-based planners are the Probabilistic Road Map (PRM) and Rapid-exploring Random Tree (RRT) algorithms. However, these planners have the drawback of producing non-optimal or asymptotically optimal paths, which may necessitate additional refinement in a subsequent step [9]. In addition to these two main

families of planners, there exist several other noteworthy approaches. One such example is the Voronoi diagram-based planner, which prioritizes maximizing the distance from surrounding obstacles. Another notable mention is the Artificial Potential Fields (APF) based algorithm, which is particularly suitable for online planning due to its adaptability to workspace updates.

Thus far, the presented methods have primarily focused on planning within the workspace, which corresponds to the physical environment where the robot operates, typically represented as an $\mathbb{R}^2$ or $\mathbb{R}^3$ Euclidean space. Nevertheless, these techniques are equally applicable to planning in the configuration space. In this space, a dimension is assigned to each Degree Of Freedom (DOF) of the robot (typically to each joint). Therefore, once a path is determined, it is straightforward to derive the position that each joint should assume. However, these approaches often necessitate planning in a high-dimensional space due to the high number of joints a robot typically possesses. Additionally, constructing such a space from prior knowledge or sensing data requires extra computation [8].

2.2.3 Metrics

Another aspect to consider in the design of robotics systems is performance metrics, which serve as direct indicators of the system's effectiveness. Additionally, these metrics enable direct comparisons between different systems or algorithms. The commonly employed performance metrics include the time required to complete a task, the number of successfully accomplished tasks, the length of the traversed path, the number of obstacle collisions, the smoothness of the followed path, and the clearance distance from obstacles. Although these metrics have been extensively tested for single robot systems, their applicability to MRSs may not always be evident. For instance, in cooperative carrying scenarios, it remains unclear how to compare the path taken by the robots with the optimal one, as the complex dynamics of MRSs result in an ill-defined notion of optimality [11], [12]. Recent literature has exhibited a growing interest in MRS metrics, such as the work by the authors in [13], who propose a novel metric known as ΔQ, designed to quantify the degree of cooperation among robots. However, note that this metric is not universally applicable to all planning algorithms, but is specifically tailored to Q-Learning-based approaches.

2.2.4 Path Planning & Control Strategies

Motion planners conventionally output a series of positions (waypoints) intended to guide the robot towards its objective. However, the direct connection of these waypoints with straight-line segments may not always suffice to generate a cohesive path that the robot can seamlessly pursue. In fact, owing to the robot's kinematics, it may not be feasible to adhere to such fragmented paths. Consequently, path generation algorithms frequently incorporate diverse geometric curves, such as polynomials, Dubins curves, or splines, to foster the generation of a smooth and feasible path.

In conjunction with the geometric constraints, it is important to account for the limitations imposed by the velocities of the robot. This consideration is incorporated during the trajectory generation phase, wherein a timing law is applied to the purely geometric path. The timing law empowers us to specify the duration of the manoeuvre while imposing constraints on the permissible maximum velocity or acceleration.

Wheeled mobile robots are commonly equipped with a low-level control system that receives a reference velocity signal as input and generates a control signal to actuate the robot's motors. This control system is typically built upon a dynamic model that accounts for the specific mechanical and electrical characteristics of the robot's hardware. From

the perspective of the end user, this controller is often regarded as a black box, with its internal workings abstracted away. While it is possible to directly feed the reference velocity signals from the trajectory planner into the low-level controller, such an open-loop scheme would result in the robot deviating rapidly from the desired trajectory. To address this, a second high-level feedback controller is commonly employed to robustly track the reference inputs. This feedback controller operates based on a kinematic model of the robot enabling it to abstract away the intricate dynamics. By incorporating this high-level controller, the robot can achieve more accurate and reliable trajectory tracking while effectively compensating for any discrepancies between the reference inputs and the robot's actual motion.

Up until now, our focus has centred on trajectory tracking, whereby a reference pose and velocity are provided for each time step to guide the robot's motion. However, there exists another class of controllers known as regulators, which operate with only the goal pose as input and navigate the robot towards that target without the need for explicitly specifying a trajectory. It is worth noting, however, that regulators do not afford the capability to precisely control the robot's trajectory at each individual time step [8].

2.3 Reinforcement Learning

RL solutions for motion planning have gained substantial interest in recent years, as evidenced by the multitude of works presented in the next chapter. Envisioning that these techniques may be extended to address the environmental dynamism present in our scenario, we dedicate this section to introducing some fundamental concepts of RL.

At the core of RL there are three fundamental components: the environment, the agent, and the training algorithm. In essence, the agent utilizes its sensors to process the information it gathers from the environment, which is commonly referred to as observations, and subsequently takes actions based on this processed data. The training algorithm then accumulates all the interactions between the agent and the environment and leverages these interactions to iteratively adjust the agent's parameters. The primary objective of this iterative adjustment is to maximize a predefined RL objective function. In the subsequent sections, we will undertake a more comprehensive exploration of these fundamental ingredients [14].

2.3.1 Modelling the Environment

One of the most prevalent and effective approaches for modelling the environment and the interaction with the agents is through a Markov Decision Process (MDP). Such a model is typically represented by a four-element tuple: $\langle S, A, T_a, R_a \rangle$, where:

- S denotes the state, which belongs to the set of states $\mathscr{S}$. In the context of robotics applications, a state could represent the robot's pose at a specific time step.
- A denotes the action and $\mathscr{A}(S)$ refers to the set of actions available to the agent at a given state. These actions describe the possible movements or operations that the agent can undertake, such as "turn left" or "go forward."
- $T(a, s, s')$ represents the transition probability function, also known as the environment dynamics. It quantifies the likelihood that the agent will transition to state s' given that it is in state s and it takes action a.
- $R(a, s, s')$ represents the immediate reward acquired when transitioning from state s to state s' due to action a. It is a numerical value that informs the agent of the quality of the chosen action based on predefined metrics.

In RL, these four components are commonly indexed by time steps ranging from $t = 0$ to $t = T$. T represents the total amount of steps performed in an episode. Although in general T could be assumed to be infinite, in our scenario, we will only consider finite episodes. Therefore S_t refers to the state of the agent at time step t (see Figure 2.1). S_T is called the terminal state. For actions the reasoning is analogous, but the last action will be A_{T-1}. Reward R_t is received upon performing action A_{t-1} from state S_{t-1}, hence the first obtained reward will be R_1 (not R_0). Figure 2.1 gives an example of the notation used in MDP in RL contexts. Moreover, the transition probability function that can be rewritten as $T(a, s, s') = P(S_{t+1} = s'|S_t = s, A_t = a)$ is often assumed to be Markovian, i.e. $P(S_{t+1} = s'|S_t = s, A_t = a) = P(S_{t+1} = s'|S_0, S_1, ..., S_t = s, A_0, A_1, ..., A_t = a)$. This assumption implies that the current state provides a complete representation of the environment, rendering any prior history irrelevant. Note that we indicate with small letters (e.g. s_t, a_t, r_t) a specific realization of the underlying stochastic variables which are indicated with capital letters (e.g. S_t, A_t, R_t).

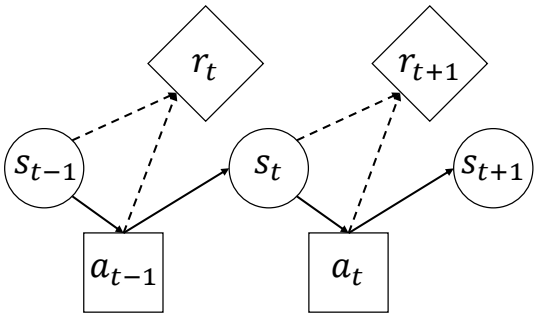


Figure 2.1: Example of RL notation used for MDP. The states are indicated with s, and to move from state s_t to state s_{t+1} the action a_t is required. Upon successful transition the reward r_{t+1} is yield.

2.3.2 Agent Decision Making

The objective of each agent is to find the best action based on certain criteria and given the current state. As the agent progresses through different states, it receives corresponding rewards. The action selected by the agent should aim to maximize a specific quantity known as the gamma-discounted return, defined as follows:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} \tag{2.1}$$

The parameter γ plays the role of weighting the rewards with respect to time. Typically, γ is assigned a value within the interval $[0, 1]$. When γ approaches 0, the agent primarily prioritizes immediate rewards, leading to a more ”myopic” behaviour. Conversely, as γ approaches 1, the agent aims to maximize both present and future rewards, exhibiting a more ”farsighted” approach.

While we have discussed the desired behaviour of an agent, we have yet to provide a formal definition of what an agent is. Mathematically, an agent is described by a function known as a policy, denoted by $\pi(a|s)$, which maps each state, $s \in S$, and action, $a \in A$, to the probability of choosing action a when in state s. In essence, the policy defines the decision-making process of the agent. Consequently, the problem of RL revolves around discovering an optimal policy, denoted by π^*, such that an agent following this policy can maximize its overall return.

2.3.3 RL Algorithms

This section introduces the fundamental that serves as a way to evaluate the agent's performance, specifically the value functions. Following this, we present an iterative algorithm known as Value Iteration (VI) which aims to find the optimal policy. However, this algorithm necessitates the knowledge of certain variables that frequently remain unavailable within real-world scenarios. Consequently, we delve into contemporary methodologies proposed to surmount this issue.

Value Functions

In order to guide the learning process, all RL algorithms rely on metric functions that assess the desirability of a state and/or action for the agent. As mentioned in the previous paragraph, we have utilized the gamma-discounted return (Equation 2.1) as a measure of an agent's performance. Now, let us introduce the concept of state-value function:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} | S_t = s \right] \tag{2.2}$$

At first, it may appear puzzling that the right-hand side of the equation involves the time variable, t, while it is absent on the left-hand side. However, it is important to understand that when computing the state-value function, the specific time step at which the agent reaches state s is irrelevant. This is because the state-value function focuses only on future rewards, disregarding any information from the past, including the number of steps it took to reach state s. It is worth noting that the state-value function, when combined with a policy, serves as an evaluation tool for assessing the goodness or utility of a given state and enables us to determine where the agent is expected to receive the highest return. Moreover, by leveraging this function, we can compare two distinct policies and assert which one is better. This observation provides us with a blueprint to identify the optimal policy. If we were capable of computing the optimal state-value function, denoted by $V^*(s) = \max_\pi V_\pi(s)$, then we would also be able to determine the optimal policy: $\pi^*(\cdot|s) = \text{argmax}_\pi V_\pi(s)$. This implies that the optimal policy corresponds to the policy that maximizes the state-value function for each state.

In a similar way let us also recall the definition of the action-value function:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} | S_t = s, A_t = a \right] \tag{2.3}$$

The key distinction between the state-value function and the action-value function is that the former accounts for the average expected return across all possible actions available to the agent when in state s. In contrast, the action-value function divides the total expected return among the various actions that the agent can choose from in a given state.

A more explicit and detailed explanation of the practical applications and uses of the state-value and action-value functions will be provided in subsequent sections.

Value Iteration

To solve an MDP and determine the optimal policy, various algorithms have been developed. One of the pioneering algorithms for solving such problems is the VI algorithm, introduced by R. Bellman in 1957. Let us first note that the gamma-discounted return can be rephrased recursively by isolating the first element of the summation:

$$G_t = \gamma^0 R_{t+0+1} + \sum_{k=1} \gamma^k R_{t+1+k} = R_{t+1} + \gamma G_{t+1} \tag{2.4}$$

By applying this definition to the state-value function, we obtain the following expression:

$$V_\pi(s) = \mathbb{E}[r + \gamma V_\pi(s')|S = s] = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a) \left[r + \gamma V_\pi(s')\right] \tag{2.5}$$

Let us introduce the following definition:

$$g(a, s) = \sum_{s',r} P(s', r|s, a) \left[r + \gamma V_\pi(s')\right] \tag{2.6}$$

so that:

$$V_\pi(s) = \sum_a \pi(a|s) g(a, s) \tag{2.7}$$

Let us recall that the primary objective of the optimal policy is to maximize the value function. Hence, the optimal policy π^* is expected to select the action that maximizes the value of $g(a, s)$. In other words, π^* should be defined as 1 for a^*, where $a^* = \text{argmax}_a\, g(a, s)$. By employing this reasoning, we arrive at the Bellman optimality equation:

$$V_{\pi^*}(s) = \max_a g(a, s) = \max_a \sum_{s',r} P(s', r|s, a) \left[r + \gamma V_\pi(s')\right] \tag{2.8}$$

As it stands, the Bellman optimality equation is not solvable. Thus, we aim to develop an iterative algorithm that converges to $V_{\pi^*}(s)$, which is the VI algorithm. The pseudocode is provided in Listing 2.1.

Table 2.1 Value iteration

for all $s \in S$ **do**
    $V(s) \leftarrow 0$
for $k = 0, 1, ..., K$ **do**
    for all $s \in S$ **do**
        $V(s) \leftarrow \max_a \sum_{s',r} P(s', r|s, a) \left[r + \gamma V(s')\right]$
for all $s \in S$ **do**
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} P(s', r|s, a) \left[r + \gamma V(s')\right]$
return π

Through the application of contraction mapping it can be demonstrated that, for a sufficiently large value of K, the mentioned algorithm converges to the optimal value function $V_{\pi^*}(s)$ [15]. However, for the sake of brevity, we will not present the proof in this context.

Learning Algorithms

Unfortunately, in most cases, obtaining access to the transition probabilities $P(s', r|s, a)$ is not feasible, rendering the direct application of the VI algorithm impractical in real-world scenarios. To address these challenges, researchers have been exploring methods for learning an approximate policy or value function by enabling the agent to interact with the environment. Typically, the policy is parameterized: π_θ, where θ represents the learnable parameters. Furthermore, the emergence of Deep Neural Network (DNN) has had a profound impact on RL algorithms, as it offers a novel approach for approximating policies

and value functions. The integration of these two fields is known as Deep Reinforcement Learning (DRL). The concept of approximating distributions shares similarities with ML, where a model learns the patterns connecting inputs to outputs through data sampled from the underlying distribution. However, the key distinction lies in the fact that in RL the data is not provided by an external entity, but rather generated by the model itself during its interactions with the environment.

This situation gives rise to several challenges, one of which is the exploration-exploitation dilemma. This dilemma arises from the fact that the agent must strike a balance between exploiting its existing knowledge to maximize its objective and exploring new strategies to potentially discover better ones. If the agent solely relies on exploiting its current strategy, it may miss out on learning more effective policies. Moreover, in contrast to ML, RL algorithms often face high variance when estimating training losses, which can impact the learning convergence speed. Another significant challenge pertains to reward specification. Generally, rewards are defined based on the agent's objectives. However, there are typically multiple ways to define the same desired behaviour. RL algorithms are highly sensitive to these different reward specifications, making it a complex task to determine the most appropriate one [16].

More formally, the class of RL algorithms that learn through trial and error by interacting with the environment is referred to as Model-Free algorithms. These algorithms can be categorized into three main types:

- Value-based algorithms: in this category, the agent learns an approximate value function, from which the policy is derived. Prominent algorithms within this class include Q-Learning [17], Deep Q-Network (DQN) [18], and the family of Monte Carlo methods [19].
- Policy search approaches: these algorithms aim to directly learn the policy itself. A popular algorithm in this category is policy gradient [20].
- Hybrid techniques: this category encompasses the so-called actor-critic methods that simultaneously estimate both the value function and the policy. It involves using the policy (actor) to predict actions while utilizing the value function (critic) to evaluate the policy choices. Typical algorithms within this class include Soft Actor-Critic (SAC) [21], Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic (A3C) [22], Proximal Policy Optimization (PPO) [23] and Deterministic Policy Gradient (DPG) [24].

2.4 Mobile Robots

As discussed in previous sections, path generation algorithms and trajectory tracking controllers are essential in translating the output of motion planners into low-level commands, used for governing the movements of the robot. A key aspect to consider during the design of these components lies in understanding the kinematic structure of the robot itself. Indeed, the freedom of movement may be constrained by this very structure. In the following, our attention will be directed towards wheeled mobile robots, and more precisely towards differential drive robots.

2.4.1 Overview of Mobile Robot Architectures

Let us provide a concise overview of the various categories of wheeled mobile robots, characterized primarily by three key attributes: wheel configuration, wheel type, and steering properties. Among the simplest architectures is the two-wheel differential drive system, comprising of two independently driven wheels and typically accompanied by a third

castor wheel to ensure stability. This arrangement enables the robot to execute spot rotations by driving the wheels in opposing directions. However, it is incapable of lateral movement, necessitating a prior turn in the desired direction before commencing forward motion. A more restricted system is the renowned Ackermann steer, akin to the steering mechanism employed in automobiles. Unlike the differential drive, this system precludes rotation in place, obliging the robot to concurrently move forward or backward to facilitate rotation. Conversely, the omnidirectional drive is an unconstrained system, which allows the robot to perform unrestricted manoeuvres. This system leverages a distinct type of wheels known as Swedish wheels, which, owing to their distinctive design, enables holonomic movement. Nonetheless, these wheels typically exhibit suboptimal performance on uneven terrains.

2.4.2 Unicycle Kinematic model

In our proposed scenario, we will employ differential drive robots as our primary platform. While the subsequently showcased path generation and trajectory tracking algorithm can potentially be extended to various robot architectures, it is crucial to comprehend the implications of utilizing a specific robot type. Thus, for the purpose of our study, our focus lies on comprehending the kinematic properties of differential drive robots. To this end, we shall introduce a widely adopted kinematic model suitable for such robots: the unicycle model.

The unicycle system comprises a single wheel that can be oriented. Its pose can be fully characterized by three parameters: $q = [x, y, \theta]$. Herein, the coordinates (x, y) define the position of the unicycle on a two-dimensional plane, while θ represents its orientation or heading angle. The control inputs governing the unicycle's motion encompass two distinct quantities: the linear velocity (v) and the angular velocity (ω). The linear velocity, often referred to as the driving velocity, is obtained by multiplying the wheel's angular speed by its radius. While the angular velocity corresponds to the rotational speed around the vertical axis. Figure 2.2 provides a schematic depiction of the unicycle model for visual reference.

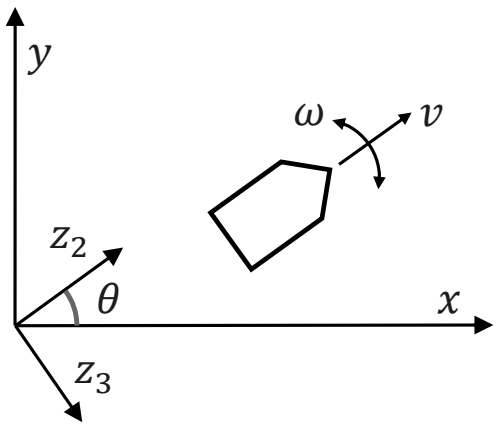


Figure 2.2: A schematic representation of the unicycle model. The vehicle is characterised based on its position (x, y) on the two-dimensional plane and its orientation θ. It is useful to introduce the concept of generalized coordinates z_2 and z_3 shown here. The former represents the direction along which the linear velocity vector v acts, while the latter represents the orthogonal axis that imposes non-holonomic constraints, rendering the robot incapable of movement in that direction.

Owing to its straightforward physical interpretation, deriving a kinematic model that establishes a relationship between the control inputs and the configuration vector q is a relatively simple task. The derived kinematic model is as follows:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{2.9}$$

In this formulation, we establish a relationship between the control inputs and the time derivative of the configuration vector, denoted by $\dot{q}$. However, the configuration vector q can be obtained by integrating $\dot{q}$ over time. This kinematic model describes the movement of the robot, specifically highlighting the non-holonomic constraints, namely that the robot cannot move in the direction orthogonal to the sagittal axis (the z_3 direction). While this model pertains to single-wheeled robots, it is important to note that it is kinematically equivalent to the more popular two-wheel differential drive architecture. Typically, these two-wheel robots are equipped with two independent motors, each responsible for driving a respective wheel. Thus, establishing a relationship between the control velocities (v, ω) and the individual wheel speeds (ω_R, ω_L) becomes fundamental. In this regard, a one-to-one correspondence exists:

$$v = \frac{r(\omega_R + \omega_L)}{2}, \quad \omega = \frac{r(\omega_R - \omega_L)}{d} \tag{2.10}$$

Here, r represents the radius of the wheels, while d denotes the distance between the two wheels.

2.5 Localization and Mapping

In tracking controllers and regulators, the feedback mechanism relies on the comparison between reference inputs and the actual positions or velocities of the robots. However, in real-world scenarios, these quantities are not directly accessible and can only be estimated using sensor measurements. This limitation motivates the need for investigating localization algorithms. In this section, we provide a comprehensive overview of the most widely used sensors for localization. We discuss the intrinsic relationship between the localization problem and mapping, and how multiple sensors can be effectively combined using sensor fusion techniques to enhance performance. Furthermore, we present commonly employed models that effectively capture the errors associated with odometry and range finder measurements.

2.5.1 Sensors for Localization

Localization can be approached in two ways: if the robot has prior knowledge of its initial pose, the localization algorithm focuses solely on tracking the position. However, if the robot's starting position is unknown, the localization problem becomes more complex, as it does not only involve errors that accumulate with motion, but it also requires a precise understanding of the surrounding environment. This is known as the global localization problem. However, in our scenario, we assume that the initial pose is known.

One of the most prevalent solutions to address this problem is through the utilization of wheel odometry. This technique involves employing rotary encoders to measure velocities on each wheel of the robot. By employing a kinematic model and integrating

these measurements over time, an estimate of the robot's position can be obtained. However, this approach is highly susceptible to errors due to the inherent sensitivity associated with integral computations. Additionally, when encountering off-road terrains, where wheel slippage can occur, the reliability of the measurements decreases significantly. Consequently, wheel odometry proves to be insufficiently robust for accurately tracking the robot's position over long distances. To overcome these limitations, more sophisticated approaches leverage alternative sensors to estimate the robot's position. One such method is Visual Odometry (VO), which, as the name suggests, relies on camera information to infer the robot's position. This technique entails extracting and matching features from each frame, and by analyzing the relative shift of the key points, the robot's position can be derived. One of the first successful uses of such an algorithm in real-world scenarios occurred on NASA's Mars Exploration Rovers, which could not rely solely on wheel odometry due to the sandy terrain [25]. One limitation of using cameras for localization is their dependency on well-illuminated environments. To overcome this drawback, researchers have explored alternative sensor types, particularly range finder sensors. Among these, Sound Navigation And Ranging (SONAR), Radio Detection And Ranging (RADAR), and Light Detection And Ranging (LIDAR) have gained significant popularity. Despite their distinct technologies, these sensors operate based on a common underlying principle: the measurement of distances. They achieve this by emitting a beam, whether it be sound waves or electromagnetic waves, and subsequently detecting the returning echo. Such sensors allow the user to capture a view of the surrounding environment. Typically the generated data is a 360-degree point cloud[1]. An advantageous characteristic of these range sensors is their resilience to variations in illumination and weather conditions, making them particularly useful in outdoor scenarios. However, this family of sensors also shares similar limitations. For instance, when the beam strikes an obstacle surface at a highly oblique angle, it may be deflected away, making the obstacle invisible to the sensor. These sensors are also susceptible to interference from other sensors operating at the same frequency or from external sources. Additionally, range and visual sensors performance are affected by the absence of features in the scene, as it may happen in a long, uniform hallway [26]. In such cases, a potential solution is to enhance the environment with artificial landmarks, such as QR codes. This strategy is explored in [27], where the authors analyze the use of roof-mounted QR codes for localization purposes.

2.5.2 Mapping

To achieve accurate localization, the measurements from range sensors must be compared with a map of the environment. This map can be generated using mapping algorithms, with one of the most prevalent families being occupancy grid mapping. An occupancy map represents the environment as a grid with evenly spaced cells, where each cell denotes a location and can have one of two values based on whether it is occupied by an obstacle or not. In order to create this map, the robot utilizes data from its sensors, but it is assumed that the robot's pose is known. It becomes evident that the mapping and localization problems are tightly interconnected. Localization necessitates a map of the environment, while mapping requires knowledge of the robot's pose. This interdependency is resolved by another family of algorithms known as Simultaneous Localization And Mapping (SLAM) algorithms. This approach enables the concurrent execution of both tasks, making it one of the most valuable and popular algorithms in the field of autonomous robotics. Continuous mapping, carried out while the robot is performing its tasks, is particularly important to cope with dynamic environments where relying only on a pre-existing map would be challenging [28].

[1] A point cloud is a discrete set of data points in space, typically representing 3D objects.

2.5.3 Sensor Fusion

In order to exploit the advantages of multiple sensors simultaneously and thus mitigate their individual drawbacks, researchers have been investigating methods for fusing measurements to enhance localization accuracy. One of the earliest breakthroughs in this direction was the introduction of the Kalman Filter (KF) by R.E. Kalman in 1960 [29]. The KF is proven to be optimal for linear systems with Gaussian process and measurement noise. However, for more complex scenarios, alternative solutions have demonstrated superior performance. Among these solutions, we find the Extended and Unscented KF, as well as the widely popular Particle Filter (PF). These filtering techniques enable the fusion of data at various levels. For instance, one approach involves fusing raw sensor data to obtain more accurate measurements. Alternatively, data fusion can occur at a higher level. Taking the position tracking problem as an example, one could obtain separate estimates of the robot's position using two different sensors and then fuse these position estimates [30].

Localization and mapping algorithms can be directly applied to MRSs. However, in such systems, there are opportunities to further exploit these algorithms by leveraging the collaborative nature of multiple robots. Combining the information gathered by each robot can greatly enhance both localization and mapping tasks. In the context of localization, one robot can estimate the positions of other robots, thereby influencing their knowledge with its own measurements. For example, a collaborative localization strategy is proposed by the authors in [31]. They utilize cameras and Time Difference Of Arrivals (TDOA) from wireless sensors to estimate the positions of other robots. The authors then perform sensor fusion, combining odometry and the other estimates using a KF. Additionally, in [32], researchers explore a cooperative observation scenario where a MRS composed by Unmanned Aerial Vehicles (UAV) aims to locate a target object. The authors present a strategy to position the aerial vehicles in a manner that minimizes localization uncertainties when fusing different measurements.

Particle Filters

Filtering techniques, in general, enable us to estimate the internal states of dynamical systems, such as the pose of a robot, even in scenarios where only partial observations, such as range finder measurements, are available, and random perturbations are present in both the sensors and the dynamical system. PF have demonstrated their effectiveness in handling nonlinear systems, which is particularly relevant to our robot's characteristics. These filters effectively represent the posterior distribution by utilizing a set of samples (particles) drawn from that distribution. In our case, this distribution describes the probability of the robot being in pose q_t given the measurements z_t and the control input u_t.

Initially, the posterior distribution is approximated based on the prior knowledge of the system state. Subsequently, the next set of particles is generated by updating the current particles using the system's dynamics and the control input u_t. Each particle represents a hypothetical state of the system, and thus the new particle is obtained by sampling from the conditional distribution $p(x_t|u_t, x_{t-1})$. Next, the measurements are incorporated using the importance factor w_t. For each particle, this factor is computed as $w_t^{[m]} = p(z_t|x_t^{[m]})$, where m denotes the particle index ranging from 1 to M. To combine these weights with the particle set, the PF employs importance sampling. This sampling procedure enables the drawing of new particles from the posterior distribution, which is in turn represented by the set of particles. Each sample is drawn with a probability determined by its corresponding weight, $w_t^{[m]}$.

Through the iterative process of updating particles and weights as described, it has been demonstrated that the set of particles will converge to the underlying posterior distribution

[30].

2.5.4 Odometry Motion Model

Based on the measured rotational displacement of the wheels obtained from the encoders, we can estimate the relative motion displacement, namely the transition from pose q_{t-1} to pose q_t. In the context of the robot's odometry system, this transition is represented as a movement from $\bar{q}_{t-1}$ to $\bar{q}_t$, where the notation with a bar denotes the coordinate system specific to the robot's odometry system, distinct from the global reference frame Oxy. The relationship between these two reference systems remains unknown, constituting the essence of the localization problem. Nevertheless, the key concept lies in the fact that the difference between $\bar{q}_{t-1}$ and $\bar{q}_t$, provides an estimate of the difference between the true poses q_{t-1} and q_t. In the position tracking problem it is assumed that the initial pose is known, i.e., $q_0 = \bar{q}_0$, however, due to the drift errors, the uncertainty in the relationship between these two frames increases over time. To better characterize this uncertainty we focus on studying the probability $p(q_t|q_{t-1}, \bar{q}_t, \bar{q}_{t-1})$.

To analyse such probability we employ the probabilistic motion model proposed by the authors of [28]. This particular model decomposes the relative displacement between two poses into three fundamental steps: a rotational adjustment towards the subsequent position, a linear movement along a straight path to reach the next position, and a final rotational alignment to match the orientation of the target pose. Figure 2.3 illustrates these sequential movements.

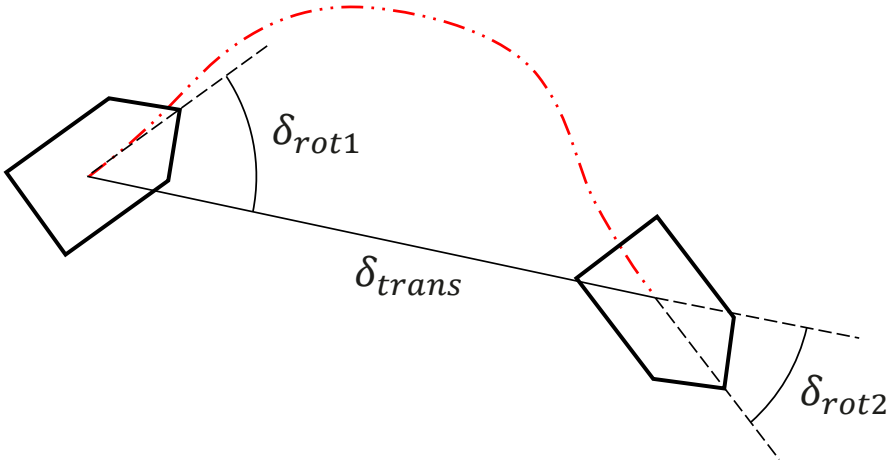


Figure 2.3: The robot transitions from its initial pose (on the left) to the target pose (on the right) by following the path indicated by the red dashed line. This path is decomposed into three distinct steps by the motion model. Firstly, the rotation δ_{rot1} is performed to align the robot with the target position. Secondly, the translation δ_{trans} is executed to move the robot to its final position. Lastly, the rotation δ_{rot2} is carried out to align the robot with the target orientation.

Mathematically, given two consecutive poses: $q_{t-1} = [x_{t-1}, y_{t-1}, \theta_{t-1}]$ and $q_t = [x_t, y_t, \theta_t]$, the displacement can be decomposed as:

$$\begin{aligned} \delta_{rot1} &= \text{Atan2}\,(y_t - y_{t-1}, x_t - x_{t-1}) - \theta_{t-1} \\ \delta_{trans} &= \sqrt{(x_{t-1} - x_t)^2 + (y_{t-1} - y_t)^2} \\ \delta_{rot2} &= \theta_t - \theta_{t-1} - \delta_{rot1} \end{aligned} \tag{2.11}$$

The probabilistic motion model then assumes that these three quantities are independently corrupted by zero-mean Gaussian noise ϵ_{b^2} with variance b^2. To compute the probability $p(q_t|q_{t-1}, \bar{q}_t, \bar{q}_{t-1})$, first we compute $(\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ from (q_t, q_{t-1}) and $(\hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2})$ from $(\bar{q}_t, \bar{q}_{t-1})$. Then the probabilities p_1, p_2, p_3 are derived as follows:

$$\begin{aligned} p_1 &= \epsilon_{\alpha_1\delta^2_{rot1}+\alpha_2\delta^2_{trans}}(\delta_{rot1} - \hat{\delta}_{rot1}) \\ p_2 &= \epsilon_{\alpha_3\delta^2_{trans}+\alpha_4\delta^2_{rot1}+\alpha_4\delta^2_{rot2}}(\delta_{trans} - \hat{\delta}_{trans}) \\ p_3 &= \epsilon_{\alpha_1\delta^2_{rot2}+\alpha_2\delta^2_{trans}}(\delta_{rot2} - \hat{\delta}_{rot2}) \end{aligned} \tag{2.12}$$

The variance of the noise depends on the magnitude of the translation and rotations steps and on the parameters α which depend on the specific robot used. Finally, the sought probability is: $p(q_t|q_{t-1}, \bar{q}_t, \bar{q}_{t-1}) = p_1 \cdot p_2 \cdot p_3$.

2.5.5 Range Finder Measurements Model

The shared characteristics of the range finders family enabled scientists to develop a unified model to describe their behaviour. We introduce the model initially proposed by the authors in [28], which characterizes the measurement process as the probability density function $p(z^k_t|q_t, m)$. Here, q_t represents the current pose of the robot, m denotes the prior map of the environment, providing information about the expected locations of obstacles, and z^k_t represents the measured distance. Given that the sensor typically performs multiple measurements in different directions at time t, the index k specifies which specific measurement is under consideration.

This probability density function is formulated as a mixture of four components:

- A normal distribution $p_{hit} \sim \mathcal{N}(z^{k*}_t, \sigma^2_{hit})$, which captures the local measurement noise when the correct range is detected. Such noise may arise from limitations in sensor resolution and atmospheric conditions.
- An exponential distribution $p_{short} \sim Exp(\lambda_{short})$, which models unexpected objects. The prior map m permits estimating the expected measured distance in a given direction. However, in the presence of an unknown object obstructing the beam path, the measured distance will be considerably shorter than initially anticipated.
- A point-mass distribution $p_{max} \sim I(z_{max})$, that represents missed object failures. These failures occur when the beam echo fails to return, possibly due to reflections or light-absorbing objects.
- A uniform distribution $p_{rand} \sim U([0, z_{max}])$, which accounts for completely unexplainable measurements.

These four contributions are subsequently linearly combined with weights w to obtain the sought conditional probability:

$$p(z^k_t|x_t, m) = \begin{bmatrix} w_{hit} \\ w_{short} \\ w_{max} \\ w_{rand} \end{bmatrix}^T \cdot \begin{bmatrix} p_{hit} \\ p_{short} \\ p_{max} \\ p_{rand} \end{bmatrix} \tag{2.13}$$

The weights coefficients and the parameters of the distributions enable tailoring the model to the characteristics of a specific sensor.

2.6 Telecommunications in Robotics

Telecommunications play a crucial role in the realm of modern robotics, facilitating seamless communication among robots and humans. The evolution of these technologies has been driven by the growing requirements of robotic systems. However, the urgent necessity to enhance the interconnectivity of robots and Cyber Physical Systems (CPS) surged notably with the onset of the fourth industrial revolution, driven by intelligent automation.

2.6.1 5G Technologies

To meet the demands posed by the proliferation of interconnected devices, including robots, and their stringent communication requirements in terms of latency and bandwidth due to their heightened autonomy, novel telecommunications technologies have been embraced. In response, the design of 5G standards defines three macro types of services with specific distinct requirements [33]:

- Massive Machine Type Communications (mMTC) focuses on supporting a vast number of low-power devices, such as sensors, meters, trackers, and other Internet of Things (IoT) devices, which sporadically transmit small amounts of data. The aim of mMTC is to achieve a remarkably high connection density of over 1 million devices per square kilometer.
- Ultra Reliability and Low Latency Communications (URLLC) pertains to the ability to deliver exceptionally reliable and low-latency communication for mission-critical applications. These include industrial automation, remote surgery, autonomous vehicles, and smart grids. The goal of URLLC is to achieve latency of less than 1 millisecond and a reliability rate of 99.999%.
- Enhanced Mobile Broadband (eMBB) focuses on empowering 5G networks to provide enhanced mobile broadband services characterized by higher data rates, increased capacity, and an improved user experience. This facet supports diverse applications such as high-definition video streaming, virtual reality, augmented reality, and cloud gaming. The objective of eMBB is to attain a peak data rate surpassing 10 Gbps and an average user-experienced data rate exceeding 100 Mbps.

2.6.2 6G Technologies

Nevertheless, the advancement of telecommunications in the realm of robotics does not culminate with the advent of 5G. On the contrary, researchers are actively engaged in the development of 6G technology, driven by the pursuit of even greater performance and more advanced capabilities compared to its predecessor. The potential advantages of 6G in the context of robotics are manifold, encompassing the utilization of terahertz frequencies for communication, millimeter sensing capabilities, and seamless integration of Artificial Intelligence (AI). Figure 2.4 highlights the evolution of 5G services in 6G. These envisioned attributes hold the promise of enabling robots to engage in real-time communication and collaboration, fostering seamless interaction with humans. The incorporation of AI into the fabric of 6G would empower robots with autonomous decision-making abilities and the capacity to learn from their experience.

Moreover, robots equipped with 6G technology would possess enhanced perceptual capabilities, allowing them to precisely perceive and interact with their surrounding environment. In fact, one of the key pillars on which 6G technologies are based is Integrated Sensing and Communication (ISAC). The concept behind this is that the same sensing device can perform perception and communication operations simultaneously. This is of great interest for MRSs, CoBots, autonomous vehicles, smart devices, and smart cities. The operational principle of such sensors resembles that of conventional range finders,

albeit operating at different frequencies and employing different waveforms. Generally, the operating frequency of range finders directly impacts the sensor's resolution and, consequently, the measurement noise. Thus, 6G sensing devices strive to employ high frequencies such as mm-Waves and terahertz to achieve accuracy comparable to LIDARs. Moreover, these frequency bands offer high channel capacity and transmission range, making them excellent communication devices [34]. Despite being a nascent technology, several prototypes have already entered the experimental phase, as exemplified in [35], wherein researchers employ terahertz and mm-Wave-based ISAC devices for SLAM operations.

The impact of 6G in robotics extends beyond these fundamental improvements, potentially unlocking novel applications in sectors such as healthcare, manufacturing, agriculture, and entertainment. The convergence of 6G telecommunications and robotics is poised to revolutionize these domains, paving the way for ground-breaking advancements and unimaginable possibilities [36], [37], [38].

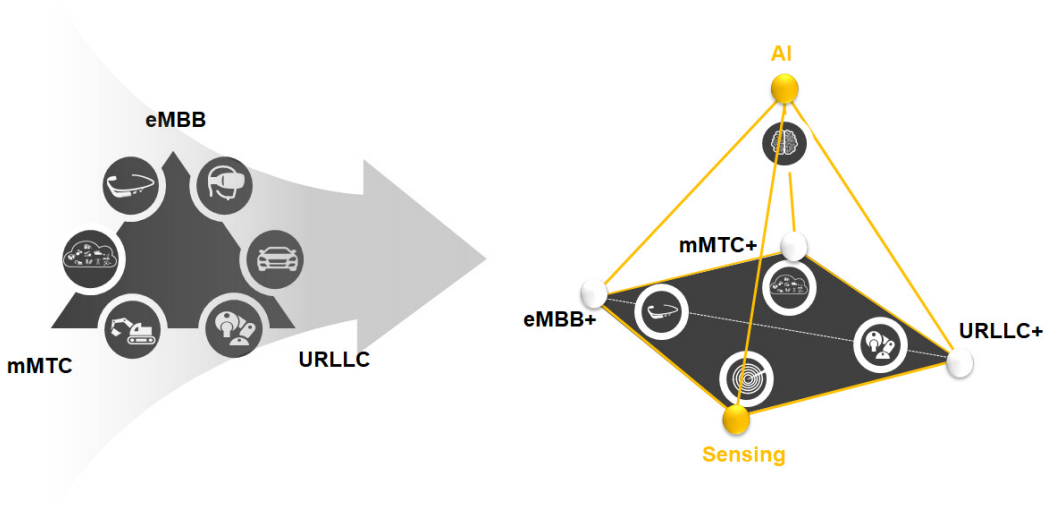


Figure 2.4: New pillars of 6G technologies.

3 Literature Review

This chapter provides a comprehensive literature review of the most recent SoA solutions in motion planning for MRSs. Subsequently, we will provide a brief overview of the current SoA in formation control, as it constitutes an indispensable component of the solution we aim to develop. Lastly, we will delve into two works that closely align with our specific scenario and we will discuss their advantages and limitations.

3.1 MRSs Motion Planning and Formation Control

When considering MRSs, the task of motion planning assumes a significantly higher level of complexity. Take, for instance, exploration missions where robots must plan and coordinate their movements to survey different regions of the environment, ideally without overlap. Similarly, certain tasks necessitate the maintenance of a specific formation among the robots, as observed in cooperative carrying scenarios. Accomplishing this presents a considerable challenge, as the planning algorithm must account for the spatial relationships among all robots, particularly during obstacle avoidance operations. Furthermore, it is worth noting that the optimal planning problem for MRSs typically falls under the category of NP-hard problems [3]. Consequently, in order to address such intricate challenges, the conventional approaches discussed earlier must undergo thorough revision, often needing to resort to simplifying assumptions.

3.1.1 Traditional Techniques

An initial and highly effective solution employed in the domain is the leader-follower architecture. Within this framework, a single robot is designated as the leader of the MRS, a decision determined through consensus algorithms or preexisting knowledge. The leader is tasked with devising the plan for the entire system, while the followers receive their motion plan either by direct communication with the leader or by inferring its trajectory. However, the centralized approach employed by this architecture may encounter problems with robustness if the leader fails [39].

An example of such an architecture is proposed in [40]. The authors extended the RRT* planning algorithm to a MRS composed of hexacopters. Notably, in this approach, the leader is not fixed but dynamically determined as the robot closest to an obstacle, thereby simplifying the obstacle avoidance process. Similarly, in [41], the authors introduce a cooperative carrying scenario utilizing UAV and employ a centralized motion planning algorithm based on RRT, coupled with a Model Predictive Control (MPC) technique for trajectory tracking. In [42], the motion planning and obstacle avoidance tasks are carried out by the leader of the formation, employing an enhanced version of the APF method. However, the trajectory generation and tracking aspects are executed in a decentralized manner, allowing each multirotor to adhere to velocity and curvature constraints. Naturally, this planning architecture finds application not only in aerial MRSs but also in various other types of robots. In [43], the authors introduce a novel two-stage planning algorithm based on RRT. Their approach to obstacle avoidance involves constructing a local configuration space with a fixed number of DOF, independent of the number of robots in the MRS. Likewise, in [44], a configuration space graph search-based motion planning technique is employed for cooperative carrying using mobile manipulators.

Furthermore, these conventional motion planners are not exclusively employed in cooperative carrying scenarios but also find applications in other prevalent tasks within cooperative MRSs. For instance, in [45], a motion planning technique based on RRT is proposed

for exploration tasks in unknown environments. The efficacy of this solution is validated through experimentation with a team of four mobile robots. Similarly, [46] utilizes Voronoi regions to partition the map into subsections and determine the specific area that each robot should explore.

Further studies delve into the exploration of novel types of motion planners to address the limitations posed by traditional approaches. In [47], an occlusion-based strategy is introduced, wherein the robots push the object solely when the line of sight between the robot and the goal is obstructed by the object itself. While this method demonstrates proficiency in the cooperative aspect of transportation, the authors do not delve into autonomous planning for obstacle avoidance. In fact, in the conducted experiments, the robots pursued a teleoperated goal.

3.1.2 AI-based Techniques

A prominent area of focus for researchers in the context of MRSs is AI. These AI-based solutions hold the potential to enhance performance when compared to traditional techniques, as they are well-suited to address the complexities of MRS dynamics. An early achievement in this direction involves a behaviour-based fuzzy logic controller, where the authors successfully coordinated a team of robots to move in formation and carry out cooperative tasks [48]. Within the realm of AI, ML has emerged as a particularly successful subfield. ML techniques possess the capacity to ”learn” from data, alleviating the need for explicit programming to accomplish specific tasks. In [49], a neural fuzzy controller is presented, showcasing the efficacy of ML in cooperative MRS scenarios. This controller enables a team of two carrying robots to execute goal-reaching and wall-following manoeuvres.

In recent years, the robotics community has witnessed a significant surge of interest in RL, a branch of ML. It involves learning through interactions with the environment, has emerged as a fundamental component in the development of intelligent robotic systems. These solutions appear to bring radical transformation in the structural organization of robot software's framework. With the advent of RL, the SoA in motion planning has shifted from a two-stage planning architecture to an end-to-end approach, where algorithms directly take sensory information from robots as input and output control signals for robot actuators [16]. This paradigm shift is exemplified in [50], where the authors propose a Q-Learning-based approach for a cooperative carrying scenario in a heterogeneous MRS, leveraging external sensors like a roof-mounted camera. Another instance is presented in [13], where the authors introduce an end-to-end DQN controller capable of learning and guiding a cooperative carrying MRS in complex environments. Furthermore, in [51], researchers adopt Hierarchical-Hops Graph Neural Networks to enable MRS ensemble exploration in unknown environments. They employ Multi-Agent Reinforcement Learning (MARL) based training algorithm to acquire collaborative strategies.

3.1.3 Formation Control

In cooperative carrying scenarios, the need for imposing constraints on the trajectory becomes even more critical to prevent object falls. The geometric paths followed by each robot must ensure the maintenance of formation throughout the manoeuvre, which also entails adjusting the speeds of the robots accordingly. To address these constraints, various solutions have been proposed in the literature.

One approach, presented in [52], involves a modified pose regulator designed for two differential drive mobile robots engaged in cooperative transportation tasks. In their scenario, the leader robot is teleoperated, while the follower robot must maintain a fixed distance to preserve the object's balance. The authors contribute by incorporating kine-

matic constraints and proposing an algorithm based on Receding Horizon Control (RHC) to guide the motion of the follower robot. Similarly, [53] tackles the same scenario by employing a Proportional Integral Derivative (PID) controller to regulate the speed of the follower robot. The controller takes as input the current mass distribution of the object between the two robots. In [54], the authors present a method for controlling cooperative carrying scenarios involving two or more non-holonomic robots. They outline a procedure for deriving smooth paths for each robot, based on an initial plan for the leader, which is represented as a virtual robot located at the formation's center. Additionally, they employ a nonlinear trajectory tracking controller to guide the robots along their designated paths. Taking a behaviour-based approach, [55] proposes an infrared sensor-based solution. The infrared sensor detects when the object is on the verge of falling, prompting the robots to execute manoeuvres to restore the object to its desired position. Lastly, [56] introduces a dynamic-based optimal controller for a cooperative carrying scenario featuring two mobile robots with manipulators holding the object. The presence of manipulators enables the object to remain stable even if the mobile platform moves away from the desired location.

3.2 Analysis of Key Related Works

The previous review of existing literature reveals a multitude of approaches in the field of motion planning for MRSs. Notably, an evident trend is emerging wherein researchers are increasingly focusing on AI-based solutions as opposed to traditional algorithms. Two studies, closely aligned with our specific scenario and task objectives, warrant particular attention: the two-stage planning algorithm presented in [43] and the DQN controller-based technique delineated in [13]. Despite their merits, both aforementioned solutions present certain drawbacks that require careful consideration.

The two-stage planner algorithm, built upon RRT, generates non-smooth paths as a consequence of its two-stage architecture, often deviating significantly from the optimal path. Additionally, the authors operate under the assumption that unknown obstacles are lower in height than the transported object, thereby only impacting the carrying robots. Moreover, this approach adopts a centralized framework, which may imply potentially stringent communication constraints.

In contrast, the DQN controller-based solutions exhibit a considerable improvement in the quality of generated paths, offering the distinct advantage of being a fully decentralized methodology. Regrettably, its most notable shortcoming lies in the fact that, from the perspective of the RL algorithm, the robots possess limited perception, detecting obstacles solely upon collision. Consequently, these robots are confined to operating in static environments, rendering the application of such techniques to real-world scenarios exceedingly challenging. Whenever environmental changes occur or unknown obstacles are introduced, the agents necessitate retraining from scratch within the new environment.

Notwithstanding the current limitations of RL-based solutions, we firmly believe in their tremendous potential. Hence motivating us to focus on such techniques to solve the challenges presented by our scenario.

4 System Model & Key Enablers

In section 3.1, it has been observed that RL holds promise as an effective approach for addressing motion planning problems in MRSs. However, the current SoA strategies fail to meet the requirements of our specific scenarios. In light of this, we propose an adapted version of the VPN algorithm [2] tailored to our cooperative carrying scenario. The experimental results demonstrate that our developed approach surpasses the current SoA techniques. Notably, our approach exhibits performance similar to the cutting-edge methods presented in [13], while additionally offering the advantage of handling dynamic environments without requiring model retraining each time the environment changes.

This chapter commences by elucidating the process of modelling the environment in which the robot will operate through Grid Worlds (section 4.1). We explain how this model acts as an intermediary between the real environment and the underlying MDP. Moving forward, section 4.2 introduces our RL planning algorithm. We outline the architecture of the DRL agent and delve into the training procedure. Within this section, we also outline advanced evaluation and training techniques and introduce an extension of our planning algorithm that incorporates 6G radio connectivity. Furthermore, we provide an extensive account of the technical implementation strategies that pertain to the RL algorithm. Transitioning to section 4.3, we shift from the discrete 2D RL domain to a more realistic 3D environment. Here, we illustrate the critical components required to translate the motion planning output into control directives for the robotic systems. This includes our multi-robot path generation algorithm and a trajectory tracking controller. Additionally, to enhance the realism of the simulated sense-plan-act loop, we integrate measurement noise into the simulated sensors. Lastly, we provide a comprehensive description of the developed 3D simulation platform.

4.1 Environment Model

A fundamental component of the RL framework is the environment. It serves as a platform that enables interactions with the underlying MDP. Specifically, it allows the agent to transition between different states based on its chosen actions and receive the corresponding rewards. It is essential for the environment to closely mimic the real-world setting and accurately represent the dynamics of the agents' interactions. At the same time, the RL environment must filter out any extra information to maximize computational efficiency. Given that the learning process entails a large number of agent-environment interactions (steps), our objective is to identify the simplest environment model capable of encapsulating the dynamics of the robots.

4.1.1 Grid Worlds

We focus on developing a model that represents the essential environmental structures, encompassing obstacles, robots, and the target location. The model should account for critical factors, such as preventing robot-obstacle collisions and, in the case of cooperative carrying, maintaining a consistent inter-robot distance. Additionally, our model should reflect the requirement of reaching the designated target area, potentially by following the shortest path available.

An excellent candidate that fulfils all the aforementioned requirements is the grid world environment. Similar to occupancy maps, these environments represent the real-world setting using a discretized grid-like map. Although they are commonly used for modelling

video game scenarios, they are also well-suited for robot planning applications. Each grid cell can contain an obstacle, a robot, or be empty. The robots are capable of moving to adjacent cells using discrete actions such as move left, right, forward, and backward. In some cases, diagonal movements may also be included, allowing for eight possible actions. However, these movements can be restricted to prevent collisions with obstacles or, in the case of cooperative carrying, to avoid object falls. The objective of the agent is to reach the cell designated as the target. For mobile robots, where most of the dynamics occur on a two-dimensional plane, grid worlds offer an excellent choice for modelling such environments. Moreover, they demonstrate computational efficiency as they can be encoded as binary matrices, further enhancing their suitability for use in robotic applications.

4.1.2 Markov Decision Processes

In order to apply RL algorithms, it is convenient to model the environment as an MDP. Initially, we will concentrate on the single-robot scenario and later extend it to the cooperative carrying case.

In an MDP, the tuple $\langle S, A, T_a, R_a \rangle$ defines its components. In the case of grid worlds, a state s simply consists of the current map of the environment, including the locations of the agent and the target. This information is sufficient to fully describe the environment at the current time step without the need for previous information[1]. We refer to the state where the agent's location coincides with the target position as the "goal state". This state also serves as the terminal state, meaning that once the agent reaches it, it remains there regardless of subsequent actions. The set A includes the allowable movements for the agent, such as $\{left, right, forwards, backwards\}$. The transition probability function T_a describes the likelihood of the agent transitioning to state s' given that it is currently in state s and takes action a. It is important to note that in our scenarios, the agent's actions have a 100% probability of leading to a specific state s'. Therefore, each action is deterministic. However, it is worth mentioning that T_a also encodes the function that indicates which is the resulting state s' (action to state map). If an action would result in the agent colliding with an obstacle, the transition is changed so that the agent remains in the current state. Finally, the reward function R_a allows us to model the desired behaviour that the agent should exhibit. Specifically, we assign a positive reward when the agent reaches the goal location and a small negative reward for each step it takes[2], regardless of whether the steps are towards the goal or not. Formally, the reward function is:

$$R(a, s, s') = \begin{cases} +1 & \text{if } s' \text{ is the goal state} \\ -0.01 & \text{otherwise} \end{cases} \tag{4.1}$$

By maximizing the cumulative rewards, the agent is motivated to strive for the positive reward associated with reaching the goal state. Simultaneously, the small negative rewards encourage the agent to minimize unnecessary transitions, leading it to select the shortest path and avoid collisions with obstacles. The specific values assigned to the rewards are design parameters that require careful tuning during the implementation phase. Typically, these values depend on factors such as the expected number of transitions necessary to reach the goal, which, in turn, is influenced by the size and complexity of the environment. Balancing the reward values effectively is crucial to ensure the agent's optimal behaviour in navigating the environment.

[1] As we are considering 2D grid world environment, without loss of generality, we will indicate with $s_{i,j}$ the state where the agent is in location (i, j).

[2] For diagonal movements, the negative step reward is multiplied by a $\sqrt{2}$ factor to account for the greater distance that the agent moves.

In the case of cooperative carrying, slight modifications are required in the formulation mentioned earlier. Specifically, the state s will now encompass the positions of all robots within the MRS. The target location will no longer be a single cell but a larger area capable of accommodating the entire MRS. The goal state is achieved when the MRS is within the designated area. As the MRS is viewed as a single entity in the MDP framework, the actions need to be adapted accordingly. Each agent can still take individual actions, but these actions must be combined to determine the collective movement performed by the MRS. One way to obtain these combined actions is by taking the Cartesian product of all individual action sets. On the other hand, the definition of the transition probability function remains unchanged. Given the current state and the combined action, there is only one possible next state s'. However, the action-to-state mapping function becomes more complex, as it needs to consider the intricate dynamics of the MRS. Similarly, the reward system remains the same, but with the new definition of the goal state.

4.1.3 Types of Environments

During the execution phase, the layout of the grid world environment is constructed based on a prior knowledge map, which typically includes static obstacles such as walls. As the robots move within the environment, the map is dynamically updated to incorporate information about unknown obstacles that the robots sense. However, in order to train RL algorithms effectively, it is necessary to generate maps that closely resemble real-world scenarios in an artificial way. One commonly used approach in RL settings is to utilize environments with uniformly randomly placed obstacles. In this case, the complexity of the environment is regulated by adjusting the percentage of occupied blocks. This scenario serves as a standard benchmark to validate the algorithm implementation and assess its performance. Additionally, we aim to analyze the specific scenario proposed in [13]. This scenario features a static environment where the MRS needs to navigate through a narrow passage to reach the goal location. Although the environment itself is static, this scenario is of great interest as it allows us to benchmark our approach against the SoA techniques. Furthermore, it enables us to evaluate the cooperative behaviour of the robots, as they are forced to cooperate and coordinate their actions to manoeuvre through the bottleneck. Nevertheless, the previously mentioned environments may not accurately represent realistic indoor scenarios. To address this limitation, ArenaBench [57] and Bench-MR [58] introduce benchmarking scenario generators specifically designed for mobile robot navigation. These research works present open-source grid world map generators capable of creating indoor maps that include rooms, hallways, and other typical indoor structures with randomized layouts. Moreover, the tools provide the flexibility to specify parameters controlling the size and quantity of generated structures, allowing for the customization of map appearance. These map generators serve as a valuable resource, not only for conducting realistic simulations but also as a benchmarking tool for robot navigation. Figure 4.1 shows some examples of the analyzed environments.

4.2 RL-based Motion Planning

With the Grid World model serving as our foundation for the environment, our focus in this section is to design an agent that can effectively leverage the benefits offered by this model. Tightly linked to the agent is the training algorithm, which must be customized to suit the agent's specific task. While these represent the primary themes of this section, we will also delve into several related topics and strategies that enhance the performance of the training algorithm and planning agent.

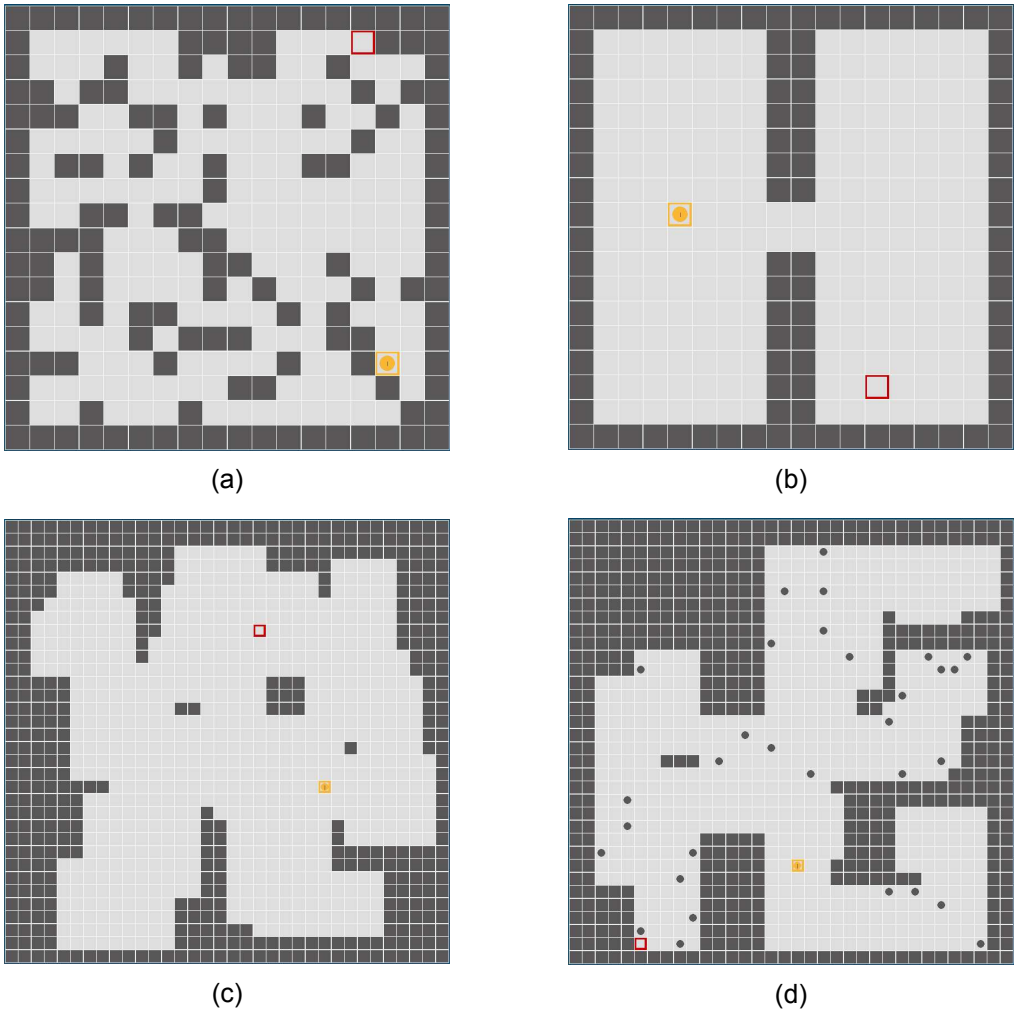


Figure 4.1: Types of analyzed grid world environments. In each illustration, dark grey squares depict obstacles, light grey cells represent empty spaces, the red square signifies the goal, and the yellow circle represents the agent. (a) presents a grid world with obstacles uniformly placed at random, where the overall percentage of obstacle cells is $p = 30\%$. (b) presents a recreation of the environment proposed in [13]. (c) showcases an indoor-like environment generated using the ArenaBench tool. (d) depicts another ArenaBench generated environment, featuring also moving obstacles represented by dark grey circles.

4.2.1 Agent's Neural Network Architecture

The objective is to determine a policy capable of effectively guiding the agents towards the desired goal. While the VI algorithm has been previously demonstrated to serve this purpose, its direct application is unfeasible in our specific scenario due to the lack of knowledge regarding the transition probability function (T_a). Consequently, researchers have been investigating methodologies for estimating this function through the utilization of ML techniques. Among the various viable solutions, one particularly promising approach that emerges in addressing our specific case is the Value Iteration Network (VIN) method [59].

The hallmark of VIN lies in its innovative approach of approximating the VI algorithm through the use of Convolutional Neural Network (CNN). The creators of VIN have demonstrated how this reformulation can be achieved in a fully differentiable manner, which means that it can be integrated into a larger Neural Network (NN) framework and trained by utilizing conventional ML techniques like backpropagation. This larger NN structure performs the dual functions of approximating the transition and reward functions[3] while simultaneously serving as a parameterization for the policy. The central idea proposed by the creators of VIN is that each iteration of the VI algorithm can be viewed as passing the previous value and rewards function through a convolution layer and a max-pooling layer. Specifically, every channel of the convolutional layer represents a distinct action, while the kernel's weights correspond to the discounted transition probabilities. Therefore, by repeatedly applying these convolutions K times, the VI algorithm can be obtained.

A drawback of VIN is its inherent difficulty in training, particularly regarding the layers responsible for approximating the transition and reward functions. The presence of interdependencies within the convolutional layers can lead to instability during the training process. In light of these challenges, a recent advancement has emerged as a potential solution: the introduction of the VPN. It is founded upon the same underlying principle of achieving a differentiable VI algorithm that can be seamlessly integrated within a NN framework. Remarkably, the findings presented by the authors indicate that VPNs exhibit significantly enhanced training efficiency and superior overall performance.

Value Propagation

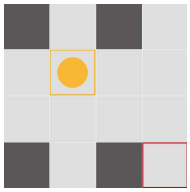
The architecture of VPNs bears resemblance to that of VINs. Specifically, the initial segment of the NN is dedicated to modelling the reward and transition functions. This is accomplished through the utilization of a NN referred to as Φ, which accepts the state s as input and outputs three quantities:

$$\overline{r}^{in}_{i,j}, \overline{r}^{out}_{i,j}, p_{i,j} = \Phi(s)_{i,j} \tag{4.2}$$

The quantities involved are indexed by the parameters i and j, which serve as coordinates in a 2D grid-like structure. Specifically, the agent is restricted to moving from its current position (i, j) to an adjacent cell. The state, denoted by s, is typically represented using binary matrices. These matrices commonly consist of an environment matrix (indicating the presence of obstacles), an agent location matrix, and a goal location matrix (see Figure 4.2).

These matrices are then treated as a 3-channel image and processed by a DNN Φ, commonly implemented using CNNs as the transition probabilities are assumed to be translation invariant. The output of this network consists of three matrices ($\overline{r}^{in}_{i,j}, \overline{r}^{out}_{i,j}, p_{i,j}$) of the same size as the input. Notably, the first two matrices model the reward function, where

[3] Although in our scenario the reward function is known, VIN still treats it as an unknown entity as it plans on an unknown MDP related to the original one only by the states and actions sets.



(a)

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(c)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(d)

Figure 4.2: Example state matrices that represent a grid world environment. (a) a bird's-eye view of the environment. (b) the environment matrix, which indicates the presence of obstacles in each corresponding cell. (c) the agent location matrix, highlighting the position of the agent within the grid world. (d) the goal location matrix, indicating the position of the goal within the grid world.

$\overline{r}^{in}_{i,j}$ represents the rewards obtained upon entering position i, j, and $\overline{r}^{out}_{i,j}$ represents the reward obtained when the agent moves out of cell i, j. The matrix $p_{i,j}$, referred to as the propagation matrix, is responsible for impeding the propagation of the state-value function. We will provide a more intuitive explanation of the meaning of these outputs shortly. Let us first introduce the Value Propagation (VProp) module utilized in the VPN, which is based on the VI algorithm:

$$v^k_{i,j} = \max\left(v^{(k-1)}_{i,j}, \max_{(i',j')\in\mathcal{N}(i,j)}\left(p_{i,j}v^{(k-1)}_{i',j'} + \overline{r}^{in}_{i',j'} - \overline{r}^{out}_{i,j}\right)\right) \tag{4.3}$$

Here $\mathcal{N}(i,j)$ is the set of states neighboring $s_{i,j}$. For instance in case there are four actions (representing the four cardinal directions) the neighboring states will be:

$$\mathcal{N}(i,j) = \{s_{(i+1,j)}, s_{(i-1,j)}, s_{(i,j+1)}, s_{(i,j-1)}\} \tag{4.4}$$

The value function at position (i, j) is indicated as $v_{i,j}$ and it is initialized to zero. Subsequently, employing the propagation equation, an estimation of the value function is obtained after K iterations.

Figure 4.3 depicts an illustrative example showcasing the rewards model, propagation factor, and state-value function resulting from training an agent on a 16×16 map with randomly positioned obstacles. $\overline{r}^{in}_{i,j}$ exhibits values close to zero in correspondence with obstacle locations, accurately representing the agent's need to avoid these cells. Conversely, $\overline{r}^{out}_{i,j}$ showcases reduced values near the goal position, effectively capturing the agent's requirement to remain at that location once reached. The matrix p manifests low values precisely where obstacles are located, effectively preventing the propagation of the state-value function through these obstructed areas.

Furthermore, the far right matrix in Figure 4.3 showcases the outcomes of the VProp module. Evidently, higher values are concentrated near the goal, gradually decreasing as the distance from the goal grows. This outcome aligns intuitively with the concept that the state-value function correlates with the expected cumulative return. Consequently, proximity to the goal results in higher state-value function estimates, as the agent can reach the goal with relatively fewer steps. It is worth noting that the example presents three regions (top right and bottom corners) where the state-value function is zero, which may initially appear as an invalid number. However, the top right corner of the map is entirely

obstructed by obstacles, rendering it impossible for the agent to navigate towards the goal if spawned in that region. Whereas, the zeros in the bottom areas can be attributed to the limited number of iterations ($K = 16$) executed by the VProp algorithm, which, in this case, is insufficient to cover the entire map comprehensively.

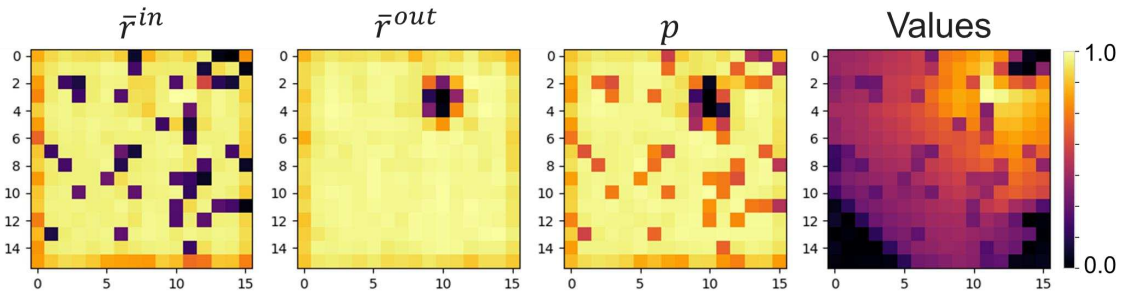


Figure 4.3: Output matrices of Φ layer ($\overline{r}^{in}$, $\overline{r}^{out}$, and p) and output matrix of the VProp module (Values).

The authors additionally introduce an alternative formulation for Equation 4.2 and Equation 4.3, referred to as Max-Propagation (MVProp), which should exhibit improved performance in larger environments:

$$\overline{r}_{i,j}, p_{i,j} = \Phi(s)_{i,j} \tag{4.5}$$

$$v^0_{i,j} = \overline{r}_{i,j}; \quad v^k_{i,j} = \max\left(v^{(k-1)}_{i,j}, \max_{(i',j')\in\mathcal{N}(i,j)}\left(\overline{r}_{i,j} + p_{i,j}(v^{(k-1)}_{i',j'} - \overline{r}_{i,j})\right)\right) \tag{4.6}$$

This modified formulation constrains the model to propagate only positive rewards. While it may yield superior outcomes for a single agent, as demonstrated by the author's experiments, our investigation within the MRS scenario did not reveal any advantages in employing MVProp. Consequently, our subsequent analysis will concentrate exclusively on the VProp architecture.

Policy

Figure 4.3 further provides valuable insight into the expected behaviour of the policy. In order to maximize the cumulative reward, the policy should prioritize selecting the actions that move the agent to the adjacent cell with the highest state-value function estimates. In the case of the scenario involving a single agent, such policy can be formulated as follows:

$$\pi(s, (i_0, j_0)) = \operatorname*{argmax}_{(i',j')\in\mathcal{N}(i_0,j_0)} v^K_{i',j'} \tag{4.7}$$

However, it is important to note that this policy formulation holds true only when the actions can be straightforwardly associated with the positions of adjacent cells. In the context of cooperative MRSs, this mapping is often unknown due to the involvement of intricate dynamics. Consequently, to facilitate the learning of such mapping function, we employ a policy that is parameterized by a NN. This NN takes as input the state's neighbourhood surrounding the agent's position, along with the corresponding estimated state-value function:

$$\pi(s, (i_0, j_0)) = F\left([s_{i',j'}]_{(i',j')\in\mathcal{N}(i_0,j_0)}, [v^K_{i',j'}]_{(i',j')\in\mathcal{N}(i_0,j_0)}\right) \tag{4.8}$$

In this case, the definition of $\mathcal{N}(i,j)$ has been revised to encompass an expanded neighbourhood, thus enabling the policy to capture a more comprehensive view of the surrounding conditions involving also other agents in the case of MRSs. The schematic representation of the proposed network architecture is depicted in Figure 4.4. The system includes a first module responsible for computing an approximation of the state-value function based on input observations, and a second module that with the assistance of the first one computes the distribution of action probabilities. Although the second module apparently only operates on local information, specifically the neighbour of the agent, it possesses the ability to make globally optimal decisions. This is made possible by leveraging the computed state-value function approximation, which encapsulates global information.

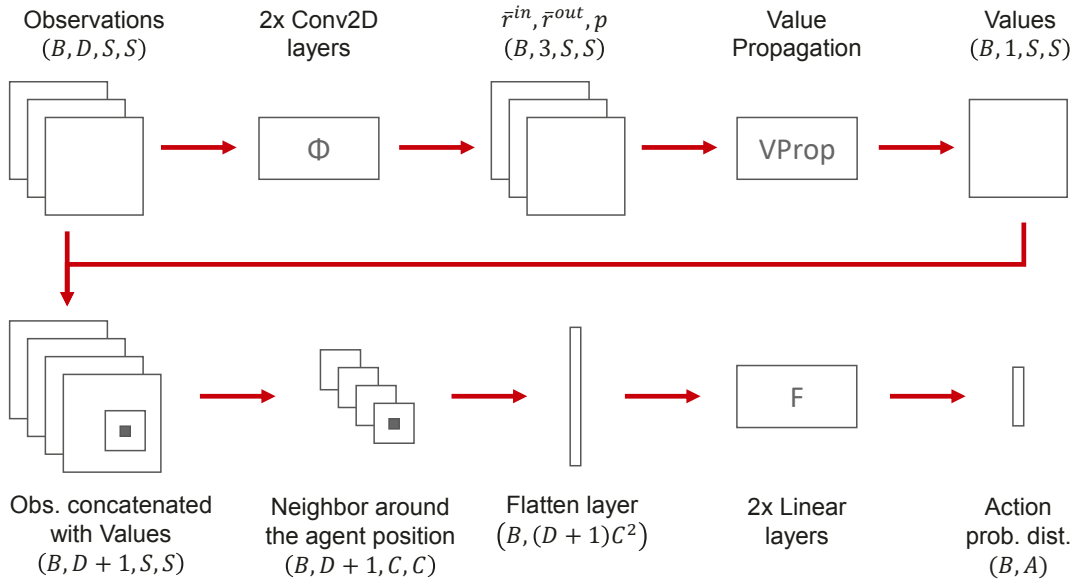


Figure 4.4: Scheme of the proposed architecture. B denotes the batch size, and D represents the number of input channels, which corresponds to the total number of agents along with the environment and goal matrices. The size of the environment is denoted by S, while C represents the cut-out area surrounding the agent's position. Lastly, A symbolizes the number of available actions.

4.2.2 Actor-Critic Training Algorithm

The primary objective of RL training algorithms is to optimize the parameters of the policy to maximize the agent's return. Analogous to loss functions utilized in ML algorithms, in RL, we also define an objective function:

$$J(\theta) = \int P(S_0) V_{\pi_\theta}(S_0) dS_0 = \mathbb{E}_{\pi_\theta}[G_0] \tag{4.9}$$

It should be noted that if we are able to compute the derivative of the state-value function and maximize the aforementioned cost function, we will, by definition, obtain the optimal policy. Furthermore, in comparison to traditional ML where the objective is typically defined by comparing predictions to ground truth data, RL operates differently. In RL, there is no ground truth data available. Instead, the data is generated through the policy's interaction with the environment. However, this introduces additional challenges as the generated samples often exhibit high variance.

By computing the gradient of the aforementioned cost function with respect to the policy parameters (detailed calculations are available in section A.1), we derive the following unbiased estimator also known as REINFORCE gradient estimator:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{\tau=0}^{T-1} \gamma^\tau G_\tau \nabla_\theta \log \pi_\theta(A_\tau|S_\tau)\right] \tag{4.10}$$

Baseline
The presence of large gradients in the estimator of Equation 4.9, caused by the derivative of the logarithm, contributes to the high variance problem. These large gradients can lead to unstable learning and cause the policy distribution to skew in non-optimal directions. While we cannot directly modify the logarithm, we can address the issue by focusing on the return G_τ. The proposed approach involves subtracting a factor from the return: $(G_\tau - b(S_\tau))$. One might question whether this alteration would significantly impact the estimator. However, it can be demonstrated that the estimator remains unaffected. By leveraging the linearity of the expectation, it can be proven that (for more details see section A.2):

$$\mathbb{E}\left[b(S_\tau)\nabla_\theta \log \pi_\theta(A_\tau|S_\tau)\right] = 0 \tag{4.11}$$

The next question to address is: what function should we utilize as $b(S_\tau)$? To achieve the objective of having the quantity $(G_\tau - b(S_\tau))$ close to zero and reducing the magnitude of the gradients, it is reasonable to select a function $b(S_\tau)$ that, on average, equals G_τ. A function that precisely achieves this requirement is the state-value function $V_\pi(S_\tau)$. However, in practice, we do not have access to the true underlying state-value function. Therefore, we approximate it using a parameterized version, denoted by $V_\phi(S_\tau)$, where ϕ represents the tunable parameters of a NN.

Advantage Actor-Critic
In the literature, the term $(G_\tau - b(S_\tau))$ is commonly referred to as the advantage function. Various versions of the advantage function exist, and here we have presented a basic formulation to provide an understanding of its working principle. To summarize, we have explored how to address the high-variance problem by employing an estimation of the state-value function, which is itself approximated by another NN that requires training. This architecture is known as the advantage actor-critic method. Specifically, the critic's role is to estimate the state-value function and provide feedback to the actor. The actor, in turn, updates the policy based on the received feedback. This actor-critic method aligns well with our network structure, as our network already utilizes an approximation of the state-value function to support the policy.

In summary, the policy training procedure operates as follows: initially, the agents interact with the environment, executing B episodes. During each episode, the trajectories in the MDP are collected: $s_0^{(i)}, a_0^{(i)}, r_1^{(i)}, ..., s_T^{(i)} \forall i \in \{1, ..., B\}$. Subsequently, the gamma-discounted return $g_\tau^{(i)}$ is computed for each episode. Finally, the policy parameters are updated using the following formula:

$$\theta \leftarrow \theta + \epsilon \frac{1}{B} \sum_{i=1}^{B} \left[\sum_{\tau=0}^{T-1} (g_\tau^{(i)} - V_\phi(s_\tau^{(i)}) \nabla_\theta \log \pi_\theta(a_\tau|s_\tau)\right] \tag{4.12}$$

To update the critic parameters, it is important to recall that we aim to approximate the return with our parametrized state-value function, i.e., $V_\phi(s_\tau) \approx g_\tau$. In this context, a common approach is to employ the square loss as a training objective to update the weights ϕ:

$$E^{critic}(\phi) = \frac{1}{B}\sum_{i=1}^{B}(g_0^{(i)} - V_\phi(s_\tau^{(i)}))^2 \tag{4.13}$$

Finally, the weight update equation is:

$$\phi \leftarrow \phi - \xi\nabla_\phi E^{critic}(\phi) \tag{4.14}$$

Note that in Equation 4.12 and Equation 4.14, the symbols ϵ and ξ represent the learning rates.

Further Developments
Compared to the theoretical framework we have presented, the current SoA methods (A3C and the A2C [22]) extend the concept of parallelism by simultaneously running multiple agents. The experiences collected from these parallel runs are then utilized to collectively update the policy and state-value function. Furthermore, these methods often employ a more advanced advantage function known as the Generalized Advantage Estimator (GAE). By introducing a control parameter (λ), it enables the regulation of the bias-variance trade-off that rises from the approximation of the state-action function (V_ϕ) [60]. Additionally, these methods incorporate various techniques to address high-variance issues, such as gradient clipping, which help stabilize the learning process.

Multi Agent Training
In the literature, numerous approaches have been explored to address the challenges of training MARL systems. One prominent paradigm is the "Centralized training for Decentralized Execution," which involves various combinations of centralization and decentralization in the training of actors and critics [61], [62]. In our specific scenario, we will employ a simple yet effective strategy. Our objective is to have a single trained model that can be deployed on all robots within the multi-agent environment. To achieve this, we will train a single agent using the actor-critic structure described earlier. However, during the episode trajectory collection, we will use the same policy to move the other agents in the MRS. The actions of all the agents are combined to govern the movements of the MRS. Likewise, the rewards and observations provided by the environment to the MRS will be used to train the single agent. Furthermore, in each episode, we will randomly select a different agent to be trained, enhancing the generalization capabilities of the NNs. A simplified overview of the proposed algorithm is presented in Table 4.1.

4.2.3 Evaluation Techniques

To assess the performance of the agents during the training phase and the execution phase, it is crucial to employ metrics that effectively highlight their behaviour when interacting with the environment. In light of this, we present the metrics utilized to evaluate the agents in our specific scenario. These metrics encompass not only the widely employed measures for assessing RL algorithms but also incorporate specialized techniques designed for evaluating robot navigation.

Training Metrics
To assess the success of the training process and enable comparisons between different training algorithms, two widely adopted metrics are the mean episode rewards and the

Table 4.1 Multi-Agent Actor-Critic (single training step)

1: **for** episode $= 1$ **to** M **do**
2: Initialize pose of the MRS randomly
3: **for** step $= 1$ **to** T **do**
4: **for** agent $= 1$ **to** N **do**
5: Using the current policy, compute action ${}^{(i)}a_t$ from observation ${}^{(i)}obs_t$
6: Execute all ${}^{(i)}a_t$
7: Receive rewards: r_{t+1}, and observation obs_{t+1}
8: Store all transitions $({}^{(i)}s_t, {}^{(i)}a_t, {}^{(i)}s_{t+1}, {}^{(i)}r_{t+1})$
9: Update Actor parameters (θ) using Equation 4.12
10: Update Critic parameters (ϕ) using Equation 4.14

mean episode length. The former represents the average cumulative rewards obtained by the agent over a specified number of episodes. Initially, when the policy has not yet been learned, the agent may move randomly and struggle to reach the goal, resulting in a low value for this metric. Conversely, in the case of successful training, this metric should exhibit an increasing trend as the agents progressively learn to execute the appropriate actions. Similarly, the mean episode length measures the average number of steps required by the agent to reach the goal. Throughout the training process, we anticipate a decrease in this metric, meaning that the agent becomes more efficient in reaching the desired objective. It is important to note that, given our defined rewards (Equation 4.1), these two metrics are deterministically interconnected. In other words, providing one of them allows to reconstruct the corresponding value of the other metric.

Additional metrics aim to evaluate the performance by directly focusing on the training algorithm. Specifically, the policy loss and value loss are two metrics derived from the update equations 4.12 and 4.14, respectively. It is expected that these metrics converge to zero after the training phase, indicating that the model's weights have reached optimal values.

Furthermore, two other commonly used metrics are the policy entropy loss and the explained variance. The policy entropy loss measures the entropy of the action distributions (x) and is typically defined as follows:

$$H(x) = -\sum_{i=1}^{n} P(x_i) \log_e P(x_i) \tag{4.15}$$

This metric quantifies the agent's confidence in making decisions by assessing the level of uncertainty in the chosen action.

The explained variance is a widely employed indicator used across various fields. It measures the proportion to which the model's predictions ($\hat{y}$) account for the variance in a given dataset (y) and is calculated as follows:

$$ExpVar(y, \hat{y}) = 1 - \frac{Var(y - \hat{y})}{Var(y)} \tag{4.16}$$

In actor-critic algorithms, this metric is often used to evaluate the accuracy of the estimated state-value function. Scores close to 1 are desirable, indicating that the predictions closely

align with the actual values, while scores close to zero suggest that the predictions are akin to random guessing.

Performance Metrics

To assess the performance of the agent once the training is completed and verify its capabilities, researchers commonly employ domain-specific metrics. Specifically, for robotic navigation, we have previously presented the most prevalent approaches in subsection 2.2.3. We are primarily interested in two metrics: the number of successful missions and the length of the path. The number of successful missions is determined by setting a threshold on the maximum number of steps allowed, beyond which an episode is considered failed. The length of the path is closely intertwined with the number of steps taken by the agent to reach the goal.

Comparing the path of the agent with the optimal path, or equivalently, comparing the number of steps taken by the agent to reach the goal with the minimum possible, is of particular interest. This allows us to assess the deviation of the agent's policy from the optimal one. In the case of a single agent, determining the minimum number of steps can be easily accomplished using simple graph search algorithms such as BFS or DFS. However, for cooperative MRSs, this task becomes significantly more challenging. In our study, we opted to estimate the optimal path for MRSs by treating them as a single agent positioned at the center of the formation. However, this approach only provides a lower bound and does not account for additional steps the MRS may need to rotate and manoeuvre through the environment. Nevertheless, this approach is still relevant as it establishes a baseline for comparison. To quantify the deviation, we found it particularly useful to compute the relative difference, rather than the absolute difference, between the number of steps taken by the robot (n_{steps}) and the optimal number of steps ($n_{steps}*$). This is expressed using the following formula:

$$RelDiff = \frac{n_{steps} - n_{steps}*}{n_{steps}*} \tag{4.17}$$

The utilization of this formula and its implications will be clarified further in subsection 4.2.4.

Cooperation Metrics

This last section is dedicated to the novel metric proposed in [22] to quantitatively assess cooperation between individuals in MRSs. The authors suggest utilizing the mean absolute error between the Q-values of the two agents:

$$^{(ij)}\Delta Q = \frac{1}{T}\sum_{t=0}^{T-1} |^{(i)}Q_t - ^{(j)}Q_t| \tag{4.18}$$

with:

$$^{(i)}Q_t = Q^{\pi}(^{(i)}s_t, ^{(i)}a_t) \tag{4.19}$$

Intuitively, this metric reflects how differently the two robots evaluate their respective situations at the same moment. A small value of ΔQ indicates a high level of cooperation between the two robots. One limitation of this metric is that it can only be computed for Q-learning-based agents, where the Q-values are available. Therefore, we present a way to extend this metric to other types of RL algorithms. Specifically, we are interested in agents that employ the Actor-Critic algorithm, where the V-values are available as opposed to Q-values.

To address this challenge, it is important to consider that the Q-values can be mathematically linked to the V-values through the utilization of the Bellman equation. This relationship can be expressed as follows:

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s'|s,a) \left(R(s',a,s) + \gamma V^{\pi}(s') \right) \tag{4.20}$$

Here, $P(s'|s,a)$ represents the transition probability associated with the environment dynamics, indicating the likelihood of transitioning to state s' when the agent is currently in state s and performs action a. In our grid world environment, the dynamics are deterministic, meaning that for any given state and action, there exists only a single potential next state. Leveraging this observation, we can simplify the previous equation by eliminating the summation, resulting in the following reformulation:

$$Q^{\pi}(s,a) = R(s',a,s) + \gamma V^{\pi}(s') \tag{4.21}$$

and after the agent performed action a, the realization of the previous stochastic equation becomes:

$${}^{(i)}Q_t = {}^{(i)}\ r_{t+1} + \gamma {}^{(i)}V_{\phi t+1} \tag{4.22}$$

thus replacing Equation 4.19. By substituting into the Equation 4.18, we obtain:

$$\begin{aligned} {}^{(ij)}\Delta Q &= \frac{1}{T} \sum_{t=0}^{T-1} \left| {}^{(i)}Q_t - {}^{(j)}\ Q_t \right| \\ &= \frac{1}{T} \sum_{t=0}^{T-1} \left| \left({}^{(i)}r_{t+1} + \gamma^{(i)} V_{t+1} \right) - \left({}^{(j)}r_{t+1} + \gamma^{(j)} V_{t+1} \right) \right| \\ &= \frac{1}{T} \sum_{t=0}^{T-1} \left| \left({}^{(i)}r_{t+1} - {}^{(j)}\ r_{t+1} \right) + \gamma \left({}^{(i)}V_{t+1} - {}^{(j)}\ V_{t+1} \right) \right| \end{aligned} \tag{4.23}$$

By adopting this revised formulation, we gain the ability to utilize the ΔQ metric for our agent. This metric enables us to assess whether the robots are successfully acquiring cooperative behaviours.

4.2.4 Curriculum Learning

In our scenario, a challenge that significantly impacts the rate of convergence of the training algorithm is the restricted amount of environmental feedback received by the agent. Specifically, the agents solely receive a positive reward upon successfully reaching the goal. This issue, commonly referred to as the ”sparse reward problem”, has been extensively discussed in the literature. In order to address this challenge, one approach is to provide additional positive feedback through rewards for accomplishing intermediate subgoals. However, this approach often leads to suboptimal policies. Alternatively, another viable solution lies in the application of Curriculum Learning (CL). This concept draws inspiration from human learning processes, for example, students progressively tackle increasingly complex concepts after mastering simpler ones. Likewise, in the domain of RL, the CL framework partitions the overall task into a sequence of subtasks, which exhibit

an incremental increase in difficulty [63]. In order to implement CL effectively within our specific scenario, it is crucial to delve into several fundamental questions. For instance, what is the precise definition of ”difficulty” for our agent? How can we generate a well-structured sequence of tasks that progressively increase in complexity? Furthermore, how can we determine the readiness of the agent to advance to the next task?

In order to address the first question, it is important to recognize that the primary obstacle impeding the robot's ability to reach the goal lies in the distance between them. In fact, when the policy is not yet trained, the probability of reaching the goal solely through random movements is inversely proportional to the size of the environment, specifically the distance between the starting point and the goal. This observation also provides insight into the second question. To construct a sequence of tasks that progressively increase in difficulty, a straightforward approach involves systematically augmenting the distance between the robot's starting position and the goal. In the literature, this approach is commonly referred to as a ”region-growing curriculum.” Another curriculum that aligns well with our particular scenario is known as the ”single-to-multi-robot curriculum.” In this curriculum, the robot initially learns to interact with the environment and successfully reach the goal in an individual manner, before subsequently tackling the multi-agent coordination scenario [64].

To address the final question, we propose employing a manual approach for the single-to-multi-robot curriculum and automatic CL techniques for the region-growing curriculum [65]. In the former approach, we assume the responsibility of determining when the single agent has sufficiently learned the task and is ready to tackle the multi-agent environment. Conversely, in the latter approach, this decision is made automatically, utilizing the $RelDiff$ metric (as presented in Equation 4.17) as feedback. In the ”region-growing curriculum”, the agent initially starts in close proximity to the goal, thereby increasing the chances of reaching it. As the $RelDiff$ metric (averaged over the last n episodes) drops below a predefined threshold, it indicates that the agent has effectively learned the subtask, signifying that the distance between the agent's starting position and the goal can be increased. The utilization of the relative difference instead of the absolute difference allows us to define a constant threshold that remains unaffected by the current path length. For the single-to-multi-robot curriculum, we employed a manual approach as it only requires one update when the agent is ready for the multi-robot scenario. However if possible automatic CL techniques should be preferred as they offer numerous advantages. In fact, a manual curriculum typically requires predetermining the number of episodes the agent must complete before transitioning to the next subgoal. As a result, if the designated number of episodes is excessively large, the training process becomes unnecessarily long. Conversely, if the number is too small, the agent may progress to a more challenging environment without having fully mastered the preceding one, thereby compromising its performance.

4.2.5 Enhanced Planning with 6G Connectivity

Robot operations are conventionally designed to take place in areas where network coverage is optimal. In practical terms, robots depend on an external infrastructure for communication with the external world. However, this infrastructure often faces challenges in ensuring uniform radio coverage across the entire operational environment, particularly in large-scale factories. Consequently, issues arise when robots traverse areas with inadequate coverage, resulting in a loss of connectivity. This loss can lead to the immobilization of the robot, as it no longer receives instructions from the external world, or make it ineffective, in case its tasks consist of transmitting its collected sensor data.

Our proposed solution entails treating these low network coverage areas as virtual obstacles and integrating them into the motion planning algorithm. Hence enabling the robots to actively avoid them. Given the static nature of the telecommunication infrastructure, characterized by fixed access points, our approach involves combining these virtual obstacles with the prior map of the environment. We expect that dynamic updates to these virtual obstacles during the execution phase will be unnecessary, as the presence of physical dynamic obstacles, such as other robots or humans, should not significantly impact the signal strength due to their limited size. To estimate the locations of these low-coverage areas, we employ pathloss values, which quantify the attenuation of the electromagnetic wave based on the distance between the transmitting and receiving antennas. We compute these values for all robot positions on the map and subsequently apply a threshold to identify areas where the radio signal falls below the required level. The calculation of pathloss can be performed through experimental measurements or, more conveniently, by simulating the channel model. This model provides a map that relates the transmitted signal to the received one, taking into account factors such as distance, carrier frequency, physical obstacles, and various other parameters. Lastly, to combine the obtained virtual obstacle map with the environment map, we utilize a simple element-wise OR operator on the two binary matrices.

For our experimental setup, we opted to select the 72 GHz E-band frequency, as it provides an excellent balance between range and capacity[4]. To accurately simulate the channel characteristics, we employed the sparse clutter, low base station variant of the Indoor Factory (InF) propagation model outlined in the 3GPP Release 17 TR 38 901 [67]. This scenario closely aligns with our target environment, ensuring relevant and realistic results. To conduct the simulations and generate the pathloss values, we utilized the mmWave module of the NS-3 simulator [68]. This module incorporates the channel model by employing the following formulas:

$$\begin{aligned} PL_{LOS} &= 31.84 + 21.50\log_{10}(d_{3D}) + 19.00\log_{10}(f_c) \\ PL_{NLOS} &= \max(PL, PL_{LOS}) \quad \text{with} \quad PL = 33 + 25.5\log_{10}(d_{3D}) + 20\log_{10}(f_c) \end{aligned} \tag{4.24}$$

The distance between the transmitting and receiving antenna (i.e., the access point and the robot) is denoted by d_3D, while f_c represents the carrier frequency. The model incorporates two distinct formulas, depending on whether the antenna have a Line-Of-Sight (LOS) or Non-Line-Of-Sight (NLOS) propagation. Within the NS-3 simulator,the scenario is configured according to our testing environment. By moving the robot, the pathloss is estimated on the whole environment and combined with the inputs of our motion planning model, as it will be illustrated in subsection 5.1.5.

4.2.6 RL Experimental Configuration

As part of this thesis research, we have successfully implemented the cooperative carrying RL motion planning algorithm previously described to assess and validate its capabilities and performance. In this section, we present the design principles adopted and provide insights into the algorithmic aspects of our implementation. In recent years, the OpenAI Gym Python library [69] has emerged as a prominent software framework within the RL community. It offers a standardized approach to building RL software by facilitating the seamless integration of RL environments with agents and training algorithms. Given the intertwined relationship between RL algorithms and ML, it is common for agents and

[4]Additionally, this frequency range is currently being investigated for indoor scenarios as part of the ongoing 6G development [66].

training algorithms to rely on ML backend frameworks. For our implementation, we have opted to utilize the PyTorch library [70], renowned for its emphasis on code readability and user-friendliness.

Environment
The OpenAI Gym Library mandates that environments developed within its framework adhere to specific guidelines and implement key functions with standardized input and output arguments. The principal functions include:

- reset: this function is called whenever the environment requires reinitialization, such as at the conclusion of an episode. In our specific scenario, this function is responsible for generating the map, determining the location of the goal, and specifying the agent's initial spawning position.
- step: this function orchestrates the agent's interaction with the environment. It accepts the agent's action as input, executes the state transition accordingly, and returns the new state observation and the associated reward.
- render: this function is dedicated to the graphical interface, enabling users to visualize the interactions between the agent and the environment. It provides a means to observe the dynamic behaviour and outcomes of the agent-environment interactions.

Existing literature has already introduced grid world environments for various purposes. For instance, in the VPN article, the authors utilized the MazeBase environment [71]. Another widely used solution is the MiniGrid library [72]. However, it is important to note that these environments primarily focus on video game scenarios rather than specifically addressing robotic navigation tasks. Given our unique requirements and the need to seamlessly integrate other components, such as the indoor map generation algorithm, we opted to develop a custom environment.

Considerable emphasis has been placed on the reset function, particularly in regard to the generation of indoor scenario maps. To accomplish this, we leveraged the ArenaBench map generation tool, which yields a binary matrix representing the randomly generated map. The map generation process can be further fine-tuned, allowing for the creation of diverse map configurations, ranging from open spaces to narrow corridors. These tunable parameters enable us to control the complexity of the generated maps. In addition to the static map generated using the aforementioned tool, we introduce dynamic obstacles that change their position at each step of the agent. To ensure that the MRS and the target area spawn in suitable locations without overlapping with map obstacles, we employ the convolution operator. Specifically, we treat the MRS and the goal area as filters, with a shape matching that of the occupied cells, and set the weights to 1. By convolving this filter with the binary matrix representing the map, we obtain an output matrix of the same size, which exclusively indicates the valid locations for spawning the MRS and the goal area, free from any interference with the obstacles present in the map. Figure 4.5 gives an example of how the convolution operator is used in this context.

Agent and Training
Multiple libraries already provide implementations of popular RL training algorithms, and two recent and widely used examples are RAY rllib [73] and Stable Baseline 3 (SB3) [74]. While RAY rllib excels in performance and natively supports distributed training, we have opted to utilize SB3 due to its ease of use and customizability. In particular, we are interested in the A2C algorithm as it closely aligns with the actor-critic method described in subsection 4.2.2. However, the A2C algorithm is originally designed for single-agent scenarios, requiring it to be adapted for the multi-agent setting, as discussed previously. Fur-

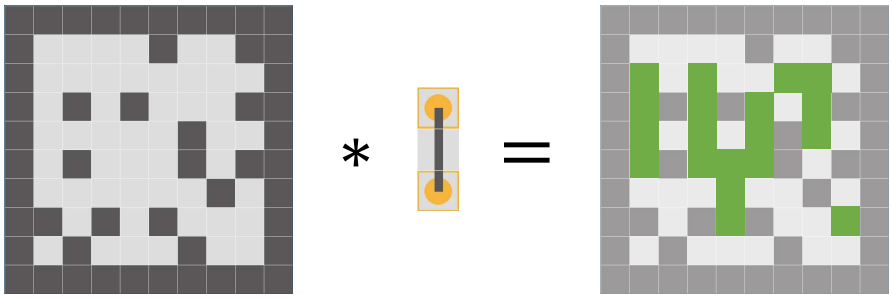


Figure 4.5: Convolution operator employed to identify suitable spawning locations for the MRS and the target area. The left side shows the grid-world environment (obstacles matrix). In the middle, there is the convolutional filter. In this example the MRS occupies three vertical cells (in its initial configuration), resulting in a 3x1 matrix with all values set to 1. On the right, the output of the convolution is depicted. Namely a matrix containing the positions where it is possible to spawn the MRS (green cells). Note that those are all the available locations where both the top and bottom adjacent cells are also empty.

thermore, the A2C algorithm in SB3 is designed to work in conjunction with SB3 ActorCritic Policies. These policies typically comprise a shared Feature Extractor NN, responsible for preprocessing input observations, followed by shared or separate actor-critic networks. In our implementation, we have customized this structure by removing the Feature Extractor and substituting the default actor-critic network with our adapted VPN. However, we have maintained the same interface as the original policy class to ensure smooth interaction with the other components of the system.

To implement the NN Φ, we utilized two Conv2D layers with the parameters outlined in Table 4.2. The VProp module follows the formulation presented in Equation 4.3, but instead of performing element-wise computations, we parallelized the computation using tensor manipulation functions provided by PyTorch. Lastly, the policy network (F) is constructed using two fully connected linear layers. In Appendix B we report our implementation of the VProp module and the of the actor network.

Conv2D layers	**1st layer**	**2nd layer**
input channels	4	3
output channels	3	3
kernel size	3	3
padding	1	1
stride	1	1
dilation	1	1
bias	true	true
activation function	ReLU	Sigmoid
Linear layers	**1st layer**	**2nd layer**
input channels	$(4+1)7^2$	64
output channels	64	8
activation function	ReLU	SoftMax

Table 4.2: NN parameters for the case of a two-robot system. In this case, the policy cutout region spans a 7×7 area. The action set has size 8.

We opted to employ the RMSProp optimizer for the purpose of maximizing the objective function. Specifically, we utilized the same parameters as the SB3 implementation of the A2C algorithm, albeit with a gradient clip threshold set to 10. Throughout all our experiments, we implemented a linear learning rate scheduler, which gradually reduces the learning rate during the training process. Intuitively the benefits of this approach are that during the initial stages, when we are distant from the optimal solution, larger steps are allowed to expedite the convergence to the optimum. Conversely, as we approach the end, smaller steps are favoured to prevent over-shootings. In our experiments, the initial learning rate was set to 0.01, and it was gradually reduced until reaching a value of 0.

To further mitigate the issue of high variance in the generated samples, it is common practice to employ a relatively large batch size compared to scenarios involving supervised machine learning. Consistent with previous experiments documented in the literature, we utilized batches of size 128. In our implementation, these interactions were obtained by executing 16 agents in parallel, each operating in its own environment for a duration of 8 steps. Through careful experimentation, we determined this specific configuration to be optimal for the hardware utilized in our study.

In addition, in our implementation, the VProp algorithm dynamically adjusts the K parameter, which is synchronized with the automatic CL updates. Specifically, during the early stages of the curriculum, when the agent is in close proximity to the goal, a lower value of K suffices for the propagation of the state-value function and for covering all relevant states. As the agent's paths become longer, the K coefficient is incrementally increased. This technique speeds up the training process during the initial phases. Indeed, the K coefficient directly influences the computational cost associated with making predictions and backpropagating weight updates, as it represents the number of convolutional layers employed to approximate the VI algorithm. It is worth mentioning that modifying the number of layers during training does not impact other components of the NN, as these layers do not contain learnable parameters.

Callbacks and Monitoring
To facilitate the automatic CL feedback loop and data logging, we leveraged the customizable callback functions[5] provided by SB3. In our implementation, we designed a callback function to retrieve the training metrics of the agent and, based on these metrics, we dynamically adjust the complexity of the environment, as described in subsection 4.2.4. Additionally, SB3 offers a set of built-in callbacks that enable the logging of training data on TensorBoard, a widely used data-logging tool in the field of ML. By leveraging this tool, we were able to effectively monitor the training progress in real-time, gaining valuable insights into the performance and behaviour of our RL motion planning algorithm.

4.3 Multi-Robot 2D-3D Control System

As outlined in section 2.2, motion planners that adopt a graph-based representation of the environment generally yield a sequence of waypoints that guide the robot towards the goal. However, robot locomotion requires trajectories that are continuous in time and space. In the following sections, we shall initially explore the methodology of adjusting the output format of our RL planning algorithm such that it matches the output of conventional motion planners. Subsequently, we shall explore the generation of paths under the constraints that the MRS requires the maintenance of a constant inter-robot distance. Following that, we will present the procedure by which paths are transformed into trajectories, alongside an examination of how trajectory-tracking controllers can be employed to

[5] Callbacks are functions that are executed periodically during the training process, providing a low-level interface with various components.

move the robots. Thereafter, we introduce a localization algorithm employed to estimate the position of the robot, a crucial requirement for the control system. We then present two metrics to assess the localization performance within the context of MRS. Finally, we propose a method to effectively simulate the planning and control techniques developed in this study using 3D models of the environment and the robots.

For the purposes of this project, we shall consider, without loss of generality, an MRS configuration consisting of two differential-drive robots tasked with the transportation of an elongated, bar-shaped object. Each robot is positioned beneath one end of the bar. It is assumed that the object is attached to each robot by means of a revolute joint, enabling 360-degree rotational motion, as well as a prismatic joint that allows for slight adjustments in the robot's movements before the object becomes dislodged. A schematic view of the joints connecting the robots to the transported object is shown in Figure 4.6.

Figure 4.6: Joints connecting the robots with the transported object. On the bottom, two differential drive mobile robots serve as the base platform. Each robot is surmounted by a revolute joint and subsequently by a prismatic joint which is connected to the to-be-transported object.

4.3.1 Transforming Actions into Waypoints

The first question we seek to tackle pertains to the conversion of the discrete actions generated by the RL policy into corresponding positional updates for the MRS. A limitation of utilizing our VPN-based planner is its lack of an end-to-end architecture, making it incapable of directly controlling the robots. Nevertheless, this architecture exhibits the advantage of being independent of the mechanical characteristics of the robots, and thus applicable across various robot types.

To facilitate the translation of policy actions into the corresponding new position of the MRS, our approach involves interpreting these actions as forces exerted on a rigid body (the transported object). By applying such forces, a roto-translational motion is induced, leading the MRS to its new position.

In our approach, we establish a 2D reference frame, denoted by Oxy, which remains fixed with respect to the environment. Additionally, we introduce a secondary frame, $O_{mrs}xy$, positioned at the center of the transported object. This auxiliary frame enables us to monitor the pose of the MRS in relation to the base coordinates. Each action-generated force possesses a unit magnitude and a direction corresponding to the action itself. These

forces are applied at the robot's position responsible for generating the respective action. Figure 4.7 offers an illustrative representation to help understanding the spatial relationships between the previously defined frames.

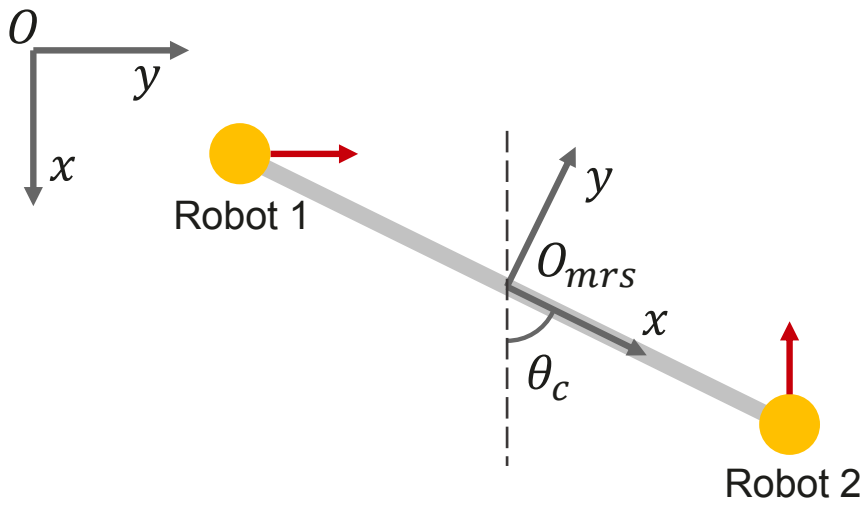


Figure 4.7: The yellow circles and the light gray bar represent the MRS seen from a top-down view. The fixed reference frame Oxy is located in the upper left corner, while the auxiliary frame $O_{mrs}xy$ is positioned at the center of the transported object. The angle formed between these two frames is denoted by θ^c. Finally, the red arrow symbolizes the forces that are generated as a result of the actions undertaken during the current time step.

The exerted forces contribute to both the translation and the rotation of the MRS. The translational motion is determined by summing all the individual forces and applying the resultant force at the center of the object (at the origin of the $O_{mrs}xy$ frame). The rotational motion is computed by calculating the mechanical moment induced by each force (using the MRS center as the fulcrum). Subsequently, the torques generated by all the forces are summed and applied at the center of the MRS formation.

We proceed to calculate the linear and angular accelerations by dividing the resulting force and torque by the mass (m) and inertia (I) of the MRS, respectively. As a subsequent step, assuming that these forces and torques act upon the MRS for a predefined time interval (Δt), we determine the linear and angular velocities, followed by the computation of the displacement in the pose. The pseudocode for the described algorithm is presented in Table 4.3.

Since our primary focus is not on achieving a physically accurate rigid-body simulation, we treat the mass (m) and inertia (I) as design parameters that provide control over the strength of translation and rotation, respectively. For the same reason, each time the algorithm is executed, the accelerations and velocities are reinitialized to zero, ensuring that no momentum is carried over from previous actions.

Note that, in our implementation, this algorithm is directly embedded into the step function of the developed RL grid world environment. It implicitly contains the unknown mapping between actions and states of the underlying MDP.

Table 4.3 Roto-Translation

procedure RotoTranslation($p^{(c)}, \theta^{(c)}, \{(f^{(1)}, r^{(1)}), ..., (f^{(N)}, r^{(N)})\}, \Delta t, m, I$)

$f^{(c)} \leftarrow f^{(1)} + \cdots + f^{(n)}$ ▷ Sum all forces

$a \leftarrow {}^{f^{(c)}}/_{m}$ ▷ Compute linear acceleration

$v \leftarrow a \cdot \Delta t$ ▷ Compute linear velocity

$p^{(c)}_{new} \leftarrow p^{(c)} + v \cdot \Delta t$ ▷ Compute new position

$\tau^{(c)} \leftarrow f^{(1)} \times r^{(1)} + \cdots + f^{(N)} \times r^{(N)}$ ▷ Compute total torque

$\alpha \leftarrow {}^{\tau^{(c)}}/_{I}$ ▷ Compute angular acceleration

$\omega \leftarrow \alpha \cdot \Delta t$ ▷ Compute angular velocity

$\theta^{(c)}_{new} \leftarrow \theta^{(c)} + \omega \cdot \Delta t$ ▷ Compute new orientation

return $p^{(c)}_{new}, \theta^{(c)}_{new}$

4.3.2 Multi-Robot Paths and Trajectories

The motion planning algorithm employed in our study operates within a discrete environment (grid world), where time is also discretized into steps. However, it is important to acknowledge that real-world robots do not operate in such discrete settings. To bridge this gap, the following path generation algorithm aims to identify a continuous path in the physical space, linking the waypoints provided by the motion planning algorithm. Specifically, we seek to determine a path that navigates the robots from an initial configuration, denoted by $q_i = [x_i, y_i, \theta_i]$, to a desired final configuration, represented by $q_f = [x_f, y_f, \theta_f]$. Moreover, this path must accommodate the non-holonomic constraints imposed by the underlying kinematic model and it should ensure a constant inter-robot distance in the formation.

Our proposed approach for addressing this challenge involves a two-step process. Firstly, we focus on finding a path for the center of the formation. Subsequently, leveraging this central path, we derive individual paths for each robot within the formation.

Path for the Formation's Center

Considering the previously described MRS with its pose characterized by the $O_{mrs}xy$ frame or, equivalently, by the configuration vector $q^{(c)} = [x^{(c)}, y^{(c)}, \theta^{(c)}]$, our objective is to determine a path $q^{(c)}(s) = [x^{(c)}(s), y^{(c)}(s), \theta^{(c)}(s)]$ that guides the MRS from the initial pose q_i to the final pose q_f. To accomplish this, we employ the Cartesian polynomials method, which generates smooth paths by interpolating a cubic polynomial between the initial and final poses as shown by the following equations:

$$\begin{aligned} x^{(c)}(s) &= s^3 x^{(c)}_f - (s-1)^3 x^{(c)}_i + \alpha_x s^2(s-1) + \beta_x s(s-1)^2 \\ y^{(c)}(s) &= s^3 y^{(c)}_f - (s-1)^3 y^{(c)}_i + \alpha_y s^2(s-1) + \beta_y s(s-1)^2 \end{aligned} \tag{4.25}$$

The parameter s serves as a parameterization for the polynomial curve and ranges from $s_i = 0$ to $s_f = 1$. Additionally, the parameters α and β are utilized to enforce boundary conditions that ensure the initial and final orientations align with the specified values $\theta^{(c)}_i$ and $\theta^{(c)}_f$, respectively. To impose these conditions, we employ the following systems of equations:

$$\begin{bmatrix}\alpha_x \\ \alpha_y\end{bmatrix} = \begin{bmatrix}k\cos\theta_f^{(c)} - 3x_f^{(c)} \\ k\sin\theta_f^{(c)} - 3y_f^{(c)}\end{bmatrix}, \quad \begin{bmatrix}\beta_x \\ \beta_y\end{bmatrix} = \begin{bmatrix}k\cos\theta_i^{(c)} - 3x_i^{(c)} \\ k\sin\theta_i^{(c)} - 3y_i^{(c)}\end{bmatrix} \tag{4.26}$$

In the systems, k represents an additional design parameter that impacts the curvature of the resulting path. Consequently, its selection must be made with careful consideration, taking into account the desired path characteristics and the size of the robots.

To obtain the heading angle function $\theta^{(c)}(s)$, a linear interpolation approach is employed[6]. Specifically, we interpolate between the initial orientation $\theta_i^{(c)}$ and the final orientation $\theta_f^{(c)}$ using the equation:

$$\theta^{(c)}(s) = s \cdot \theta_f^{(c)} + (1-s) \cdot \theta_i^{(c)} \tag{4.27}$$

An example of the Cartesian polynomials method applied to MRSs is depicted in Figure 4.8.

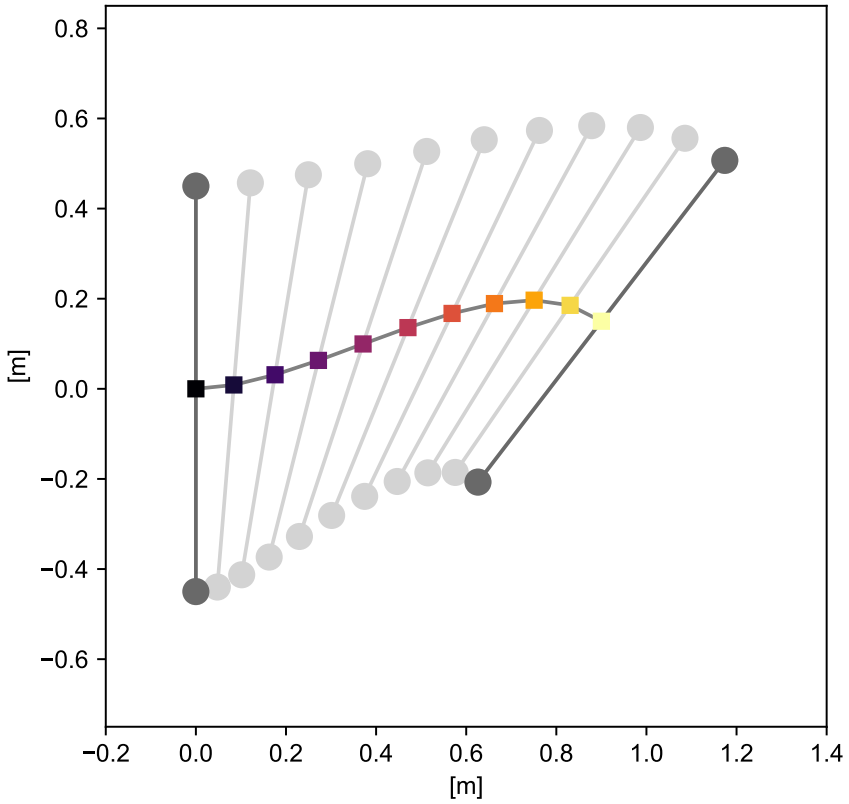


Figure 4.8: Cartesian polynomials path generation algorithm for MRSs. The left side presents a two-robot system, depicted in dark grey, in its initial configuration. On the right side, the same system is depicted in its target pose. The coloured squares represent samples of the path $q^{(c)}(s)$ and, additionally, the MRS pose at each step is illustrated in light grey.

[6]Note that with this definition, the function $\theta^{(c)}(s)$ is monotonic.

Individual Robot's Paths
Deriving the path for each individual robot within the MRS becomes straightforward once we have obtained the path for the center of the formation. To facilitate this process, we introduce a notation that expresses the position of each robot in the MRS coordinates ($O_{mrs}xy$). For implementation purposes, we employ polar coordinates, where the position of robot i is defined by a radius and an angle $(r^{(i)}, \psi^{(i)})$. With this notation in place, we can establish the path of each robot, $q^{(i)}(s) = [x^{(i)}(s), y^{(i)}(s), \theta^{(i)}(s)]$, based on the given center path $q^{(c)}$, by utilizing the following expressions:

$$\begin{aligned} x^{(i)}(s) &= x^{(c)}(s) + \cos\left(\theta^{(c)}(s) + \psi^{(i)}\right) r^{(i)} \\ y^{(i)}(s) &= y^{(c)}(s) + \sin\left(\theta^{(c)}(s) + \psi^{(i)}\right) r^{(i)} \end{aligned} \tag{4.28}$$

To define $\theta^{(i)}(s)$, a commonly adopted approach involves assigning the robot the same heading angle as the tangent to the path. This can be expressed as follows:

$$\theta^{(i)}(s) = \text{Atan2}\left(\dot{y}^{(i)}(s), \dot{x}^{(i)}(s)\right) \tag{4.29}$$

We would like to emphasize that the paths generated using the aforementioned equations possess the property of ensuring a constant inter-robot distance. Given the square of the distance between the position of the two robots:

$$d_{ij}^2(s) = \left(x^{(i)}(s) - x^{(j)}(s)\right)^2 + \left(y^{(i)}(s) - y^{(j)}(s)\right)^2 \tag{4.30}$$

The constant inter-robot distance property can be expressed as:

$$d_{ij}^2(s) = d_{ij}^2(0) \quad \forall s \quad \forall i, j \tag{4.31}$$

Where (i, j) is a pair of robots in the MRS. While this property may seem geometrically intuitive, we will now present a mathematical proof specifically for the case of our two-robot system. However, the following proof can be readily extended to other configurations, even those of a more complex nature.

Given the two-robot system defined as $[(r^{(1)}, \psi^{(1)}), (r^{(2)}, \psi^{(2)})] = [(r, 0), (r, \pi)]$, the distance between the paths of the two robots, $q^{(1)}(s)$ and $q^{(2)}(s)$, remains constant and equal to $2r$ for all values of s. In other words, the following relationship holds:

$$\sqrt{d_{ij}^2(s)} = 2r \quad \forall s \tag{4.32}$$

Proof
First, given the definition of $d_{ij}^2(s)$, let us focus on:

$$
\begin{aligned}
x^{(1)}(s) - x^{(2)}(s) &= x^{(c)}(s) + \cos\left(\theta^{(c)}(s) + \psi^{(1)}\right) r^{(1)} - x^{(c)}(s) - \cos\left(\theta^{(c)}(s) + \psi^{(2)}\right) r^{(2)} \\
&= \cos\left(\theta^{(c)}(s) + \psi^{(1)}\right) r^{(1)} - \cos\left(\theta^{(c)}(s) + \psi^{(2)}\right) r^{(2)} \\
&\text{by using the definitions of } r^{(i)} \text{ and } \psi^{(i)} \\
&= r\left(\cos\left(\theta^{(c)}(s)\right) - \cos\left(\theta^{(c)}(s) + \pi\right)\right) \\
&\text{by the rule } \cos(\alpha + \pi) = -\cos(\alpha)\text{:} \\
&= 2r\cos(\theta^{(c)}(s))
\end{aligned}
\tag{4.33}
$$

Similarly $y^{(1)}(s) - y^{(2)}(s) = 2r\sin(\theta^{(c)}(s))$. Hence:

$$
\begin{aligned}
2r &= \sqrt{\left(x^{(1)}(s) - x^{(2)}(s)\right)^2 + \left(y^{(1)}(s) - y^{(2)}(s)\right)^2} \\
&= \sqrt{\left(2r\cos(\theta^{(c)}(s))\right)^2 + \left(2r\sin(\theta^{(c)}(s))\right)^2} \\
&= \sqrt{4r^2\cos^2(\theta^{(c)}(s)) + 4r^2\sin^2(\theta^{(c)}(s))} \\
&= \sqrt{4r^2} \\
&= 2r \quad \square
\end{aligned}
\tag{4.34}
$$

Further Considerations

There exist two special cases that enable us to simplify the computational complexity associated with generating paths. The first case occurs when the MRS moves on a straight line. In such instances, complex cubic polynomials are unnecessary for interpolating the initial and final positions. The second case arises when the MRS executes an in-place rotation, causing each robot in the formation to follow a circular arc path.

As a negative consequence of these two special cases, it is possible for discontinuities to arise in the orientation of the robot between consecutive path segments. The final orientation of the preceding segment may not correspond to the initial orientation of the subsequent one. This mismatch in orientations can result in significant errors between the reference path and the actual robot position, potentially leading to the displacement of the transported object. Consequently, prior to commencing a new path segment, the robot will execute an in-place rotation to ensure proper alignment with the correct direction. This step ensures smoother transitions between path segments and minimizes tracking errors.

Trajectory Generation

To transform the purely geometric path $q(s)$ into a trajectory expressed as a function of time, a timing law $s(t)$ needs to be applied. This timing law not only converts the variable s into t, which ranges from t_i to t_f, but also serves as a means to enforce additional constraints. Real robots have limitations in terms of their top speeds, both linear and angular, and to ensure accurate tracking performance, the reference trajectories should not exceed these limits. Consequently, one of the most commonly employed approaches is uniform scaling, which involves slowing down the timing law by dividing it by the duration

of the trajectory[7] T:

$$s(t) = \frac{t}{T} \tag{4.35}$$

Hence, by increasing the value of T, we can effectively reduce the maximum speed required for the robot to follow the trajectory. However, the trajectory $q(t)$ solely comprises the robot's pose information, whereas controlling the robot also necessitates knowledge of the linear and angular velocities $v(t)$ and $\omega(t)$. These reference velocities can be computed from $q(t)$ using the following formulas:

$$v(t) = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}, \quad \omega(t) = \frac{\ddot{y}(t)\dot{x}(t) - \ddot{x}(t)\dot{y}(t)}{\dot{x}^2(t) + \dot{y}^2(t)} \tag{4.36}$$

Trajectory Tracking
The velocities reference values outlined in Equation 4.36 of course represent an input signal for the robot to follow the desired trajectory $q_r(t)$. Nevertheless, relying solely on this feedforward approach is impractical for real-world robotic systems due to its susceptibility to process noise. In order to address this challenge and bolster the tracking system's robustness, we opt for employing a feedback error-space architecture based on pose error. This error is not merely the difference between the reference pose ($q_r(t)$) and the current pose ($q(t)$), but rather, it is defined in the following manner:

$$e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{4.37}$$

The primary function of the rotation matrix is to align the Cartesian component of the error with the robot's reference frame. More specifically, e_1 characterizes the error along the sagittal axis of the robot. Our objective is to develop a controller that drives the error dynamics to zero while simultaneously tracking the reference velocities. In essence, the controller should generate a signal $u = [u_1, u_2]$ that modulates the velocities based on the current pose error. For the sake of conciseness, we will omit the detailed derivation of the error dynamics and the controller design, as they are available in [8]. Instead, we will provide the final outcome. The input transformation that relates the control signal u, the reference velocities and the pose error to the actual velocity input signal is:

$$\begin{aligned} v(t) &= v_r(t) \cdot \cos(e_3(t)) - u_1(t) \\ \omega(t) &= \omega_r(t) - u_2(t) \end{aligned} \tag{4.38}$$

Regarding the controller, it has been demonstrated that, due to the nonholonomy of the unicycle system, there does not exist a universal controller capable of tracking arbitrary trajectories [75]. Consequently, a common assumption is often made, wherein the trajectories to be tracked are deemed persistent. This assumption implies that the reference velocity $v_r(t)$ can converge to zero as long as $\omega_r(t)$ does not, and vice versa. Fortunately, our trajectories, designed using the Cartesian polynomials method, fulfil this criterion. Therefore, we can employ a family of controllers that asymptotically stabilize persistent trajectories. Specifically, we utilize the following nonlinear controller:

[7] Note that we have employed the same symbol to denote the number of steps within an RL episode. However, the context in which the symbol is used should provide sufficient information to prevent any confusion or ambiguity.

$$
\begin{aligned}
u_1(t) &= -k_1 \cdot e_1(t) \\
u_2(t) &= -k_2 \cdot v_r(t) \frac{\sin(e_3(t))}{e_3(t)} \cdot e_2(t) - k_3 \cdot e_3(t)
\end{aligned}
\tag{4.39}
$$

where k_1, k_2, k_3 are the controller's gains. We have adopted the same values employed by the authors of the controller in their simulations: $k_1 = k_3 = 1.4$ and $k_2 = 1$. The stability of this controller is proven by the authors using conventional techniques that involve a Lyapunov candidate function.

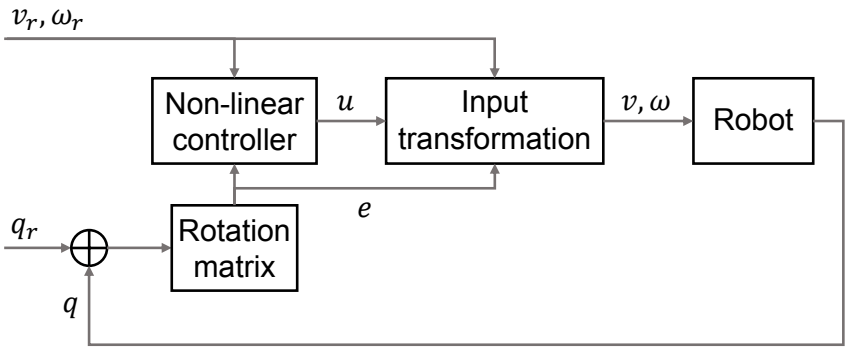


Figure 4.9: Non-linear trajectory tracking control scheme. In this scheme, the pose error is determined by applying a rotation matrix to the difference between the reference pose and the actual pose. The nonlinear controller utilizes this pose error along with the reference velocities to generate the control signal u. Subsequently, the signal u is converted into control velocities v and ω through the input transformation block.

During the design of the control system in the preceding sections, an underlying assumption was made, namely, that the robot's current pose q is known. However, this assumption does not hold true in real-world scenarios, where the robot's pose can only be estimated. In the following, our focus shifts towards the investigation of pose estimation methods. Finally, we propose a method for generating measures that closely mimic real-world data, thereby enhancing the realism of our simulations. We also provide detailed insights into the integration process of localization algorithms within the developed simulation framework.

4.3.3 Monte Carlo Localization

As we have seen in subsection 2.5.1 odometry-based localization is prone to drift errors, therefore we aim to fuse odometry and range finders measurements to obtain a better localization algorithm. To achieve this, we employ the Adaptive Monte Carlo Localization (AMCL) approach.

Monte Carlo Localization (MCL) is a family of localization algorithms that leverages the PF as its core algorithm. In the context of localization problems, the initial distribution of particles is typically a uniform distribution that encompasses all possible states, especially when dealing with global localization. Conversely, for position-tracking problems where the initial state is known, the initial distribution tends to be concentrated around the starting state. The system's dynamics are determined by the motion model, wherein the control inputs (denoted by u_t) correspond to odometry measurements $\bar{q}_t, \bar{q}_{t-1}$. These measurements provide information about the robot's motion and aid in updating the particle set to

represent the potential states of the system. The importance factors, which weigh the contribution of each particle, are computed based on the range finder measurements model $p(z_t^k|q_t, m)$.

The adaptive version of the MCL algorithm optimizes certain steps to reduce computational costs and make the algorithm more suitable for online applications. One key factor contributing to the algorithm's efficiency is the number of particles that need to be simulated. Typically, to achieve good results, the number of particles ranges from 100 to 10,000. However, a high number of particles is usually only necessary when the algorithm has little knowledge about the robot's initial position. As the algorithm converges and gains more information, the number of particles can be reduced. This is precisely what the adaptive version of the algorithm accomplishes. It dynamically adjusts the number of particles at each step based on the current requirements.

Despite the improved accuracy of the AMCL algorithm in estimating the robot's pose compared to simple odometry-based localization, it is not suitable for direct usage in the control feedback loop due to its computational slowness. In practical implementations, odometry localization is employed within the feedback loop as it can be executed at high frequencies, allowing for responsive control. However, to mitigate drift errors, the odometry estimates are periodically adjusted with the more accurate AMCL estimates.

To effectively merge the two pose estimates, we leveraged the AMCL algorithm implementation found in the Nav2 package [76], which also is directly compatible with ROS. Our approach involves extracting the transformation between the fixed global reference frame and the odometry frame. This transformation, implicitly computed by the AMCL algorithm, compensates for the drift in odometry, as depicted in Figure 4.10. This strategy ensures that the localization maintains its responsiveness while benefiting from the drift error mitigation capabilities given by the AMCL algorithm.

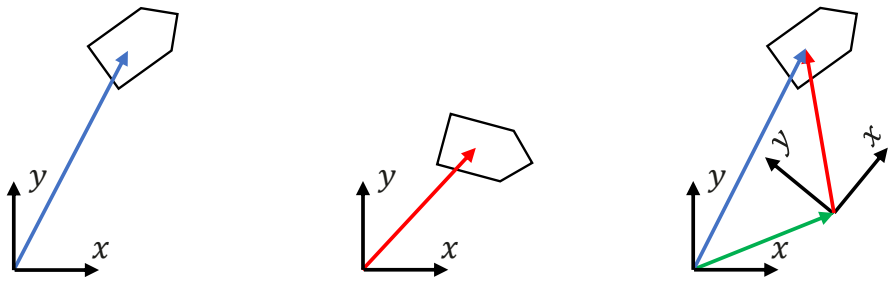


Figure 4.10: Compensation of odometry drift error by the AMCL algorithm. The diagram illustrates the true pose of the robot on the left. In the middle, the robot's pose in the odometry model frame of reference deviates from the true pose due to drift errors. On the right, the odometry reference frame is adjusted using the frame transformation provided by the AMCL algorithm (represented by the green arrow), resulting in an accurate pose estimation that aligns with the true pose.

4.3.4 Metrics for Localization

In order to assess the performance of the localization algorithm and evaluate different sensing devices, we introduce two metrics: the global position error and the local position error. The global position error measures the difference between the reference position

and the actual position of the MRS center at each time step, namely:

$$\text{Global pos. err.} = \sqrt{\left(x_r^{(c)}(t) - x^{(c)}(t)\right)^2 + \left(y_r^{(c)}(t) - y^{(c)}(t)\right)^2} \tag{4.40}$$

On the other hand, the local position error focuses on the error associated with maintaining a constant inter-robot distance. This metric specifically evaluates the precision of the MRS in preserving a desired formation. For our two-robot system, it can be expressed by the following equation:

$$\text{Local pos. err.} = \frac{1}{2} \cdot \left(\sqrt{\left(x^{(1)}(t) - x^{(2)}(t)\right)^2 + \left(y^{(1)}(t) - y^{(2)}(t)\right)^2} - d_{nom} \right) \tag{4.41}$$

with d_{nom} the nominal (reference) distance between the two robots:

$$d_{nom} = \sqrt{\left(x_r^{(1)}(t) - x_r^{(2)}(t)\right)^2 + \left(y_r^{(1)}(t) - y_r^{(2)}(t)\right)^2} \tag{4.42}$$

In the context of transportation scenarios, requirements are typically imposed on this local position error as it is essential to prevent any unintended displacement or destabilization of the transported object.

4.3.5 Generating Noisy Measures

The odometry motion model presented in subsection 2.5.4 has two key applications that make it highly relevant in our scenario. Firstly, it serves as a means to comprehend the accumulation of motion errors over successive time steps, while also being an integral component of the AMCL algorithm. Secondly, it finds application in simulating odometry errors. In our simulation framework (as described in subsection 4.3.6), the robots exhibit ideal motion without any process noise, and the sensor returns noiseless measurements. In order to facilitate a more realistic simulation, we artificially introduce noise-corrupted measures by reversing the equations of the motion model. The step-by-step procedure for generating the noisy measures q_t^n is provided in Table 4.4.

The Turtlebot3 robot is equipped with a LIDAR scanner that enables it to measure distances up to 3.5 meters in all directions around the robot, covering a full 360-degree range. In the simulated robot, the LIDAR operates similarly to its real-world counterpart, providing measurements with noise generated according to the model described in subsection 2.5.5. However, due to constraints in simulation, the p_{max} and p_{rand} components of the model are excluded. Additionally, we assume that the environment is fully mapped in advance, eliminating the need to consider the p_{short} component. As a result, the measurement model in the simulated environment reduces to a simple Gaussian noise model. Despite this simplification, the model remains representative of real-world measurements, as the p_{hit} component typically is the predominant one.

4.3.6 Experimental Platform: Simulations and 3D Visualization

This section presents an implementation-oriented description of the integration process between the motion planning algorithm, the path generation module, and the trajectory tracking controller outlined previously. Additionally, we describe how we conducted realistic simulations utilizing the ROS framework and the Gazebo 3D visualization tool.

Table 4.4 Noisy Odometry

1: **procedure** NoisyOdometry(q_{t-1}, q_t, q^n_{t-1})
2: $\delta_{rot1} \leftarrow \text{Atan2}\,(y_t - y_{t-1}, x_t - x_{t-1}) - \theta_{t-1}$
3: $\delta_{trans} \leftarrow \sqrt{(x_{t-1} - x_t)^2 + (y_{t-1} - y_t)^2}$
4: $\delta_{rot2} \leftarrow \theta_t - \theta_{t-1} - \delta_{rot1}$
5: $s_{rot1} \leftarrow \text{Sample}\left(\epsilon_{\alpha_1\delta^2_{rot1}+\alpha_2\delta^2_{trans}}\right)$
6: $s_{trans} \leftarrow \text{Sample}\left(\epsilon_{\alpha_3\delta^2_{trans}+\alpha_4\delta^2_{rot1}+\alpha_4\delta^2_{rot2}}\right)$
7: $s_{rot2} \leftarrow \text{Sample}\left(\epsilon_{\alpha_1\delta^2_{rot2}+\alpha_2\delta^2_{trans}}\right)$
8: $\delta^n_{rot1} \leftarrow \delta_{rot1} + s_{rot1}$
9: $\delta^n_{trans} \leftarrow \delta_{trans} + s_{trans}$
10: $\delta^n_{rot2} \leftarrow \delta_{rot2} + s_{rot2}$
11: $x^n_t \leftarrow x^n_{t-1} + \delta^n_{trans}\,\text{cos}(\theta^n_{t-1} delta^n_{rot1})$
12: $y^n_t \leftarrow y^n_{t-1} + \delta^n_{trans}\,\text{sin}(\theta^n_{t-1} delta^n_{rot1})$
13: $\theta^n_t \leftarrow \theta^n_{t-1} + \delta^n_{rot1} + \delta^n_{rot2}$
14: **return** q^n_t

Turtlebot 3

The planning algorithm presented in this thesis is designed to be applicable to various types of mobile robots. However, for the purpose of our simulations, we specifically concentrate on the differential drive architecture. In this regard, we have chosen the Turtlebot3 robot platform (depicted in Figure 4.11) as our preferred option. The Turtlebot3 is widely recognized for its user-friendly nature, making it ideal for educational and research purposes. Moreover, it benefits from extensive documentation and is an open-source project. Notably, the Turtlebot3 is equipped with a diverse range of sensors, including wheel encoders, an Inertial Measurement Unit (IMU), and a LIDAR scanner.

Simulated Environment

To faithfully replicate real-world scenarios, our objective is to reconstruct the 2D grid world environments presented in section 4.1 in a 3D format. To accomplish this, we leveraged an open-source tool called image2gazebo [77], which facilitates the conversion process from a 2D binary matrix representing an occupancy map to a 3D model of the map. This conversion is achieved by extruding the obstacles indicated on the map into the third dimension. The output of this process is a Standard Tessellation Language (STL) file that contains the meshes[8] of the resulting 3D model. Subsequently, the STL file can be combined with a Spatial Data File (SDF) model, which incorporates physical properties such as collisions, textures, and gravity. This SDF model can then be interpreted by the Gazebo visualization tool. Additionally, this conversion plugin offers the flexibility to specify parameters such as the height of the walls and the size of the cells.

It is important to note that the conversion process from 2D to 3D for simulations is opposite to what the robots undergo during the execution phase. In practice, the robot lives within a 3D world and, through its sensors, reconstructs a 2D representation of the surrounding environment, which is then used for planning operations.

[8] A 3D mesh is the structural build of a 3D object model consisting of a collection of vertices, edges, and faces.

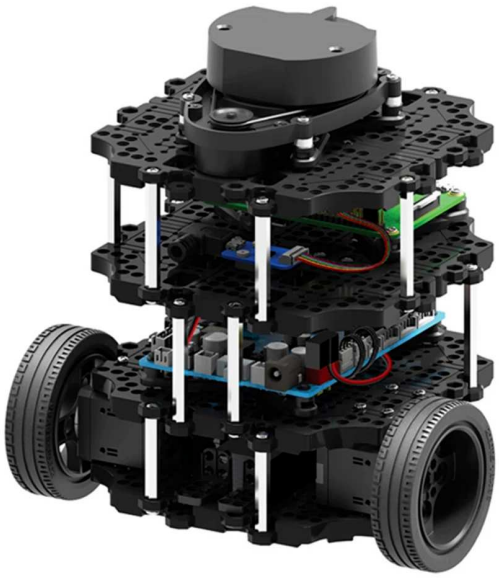


Figure 4.11: Turtlebot3 - model Burger.

To complete the simulation framework, we integrated the online available 3D model of the Turtlebot3, which includes plugins to simulate its various sensors and actuators. Additionally, we custom-designed the object being transported to match the scale of the robots used in the experiments. Figure 4.12 illustrates the process of converting the 2D grid world into a 3D environment and the 3D model of the MRS under test.

Middleware Interface with ROS

To orchestrate the planning and control components with the simulated robot hardware, we employed ROS, a middleware framework. At the core of ROS lies its powerful message-passing interface, which enables seamless communication and interaction between diverse components. While ROS encompasses a range of communication protocols, its fundamental approach is rooted in the publish/subscribe paradigm. Each component operates autonomously within a dedicated node (process), and in order to exchange information with other nodes, it must publish its data to a designated topic. Correspondingly, to receive data, a node must subscribe to the topic of interest. In the context of our scenario, ROS assumes the critical role of facilitating both intra and inter-robot communication, while also serving as the interface between the robotic system and the visualization tool.

In terms of intra-robot communications, we have devised a main control node that assumes responsibility for both planning and control operations. This node subscribes to the relevant robot sensors topics, enabling the reception of perception data. Upon processing this data through our planning algorithm, the controller then transmits its directives to the actuators by means of publishing them on a dedicated topic. Turning our attention to inter-robot communications, their purpose lies in the synchronization and coordination of the multiple robots within the MRS. This synchronization is paramount to ensure the simultaneous movement and maintenance of formation among the robots. Additionally, our planning algorithm mandates precise knowledge of the positions of all robots within the MRS. Consequently, a periodic exchange of this positional information becomes imperative. While simulations may not differentiate between intra and inter-robot communication channels, in practical implementations, a distinction arises. Intra-robot communication

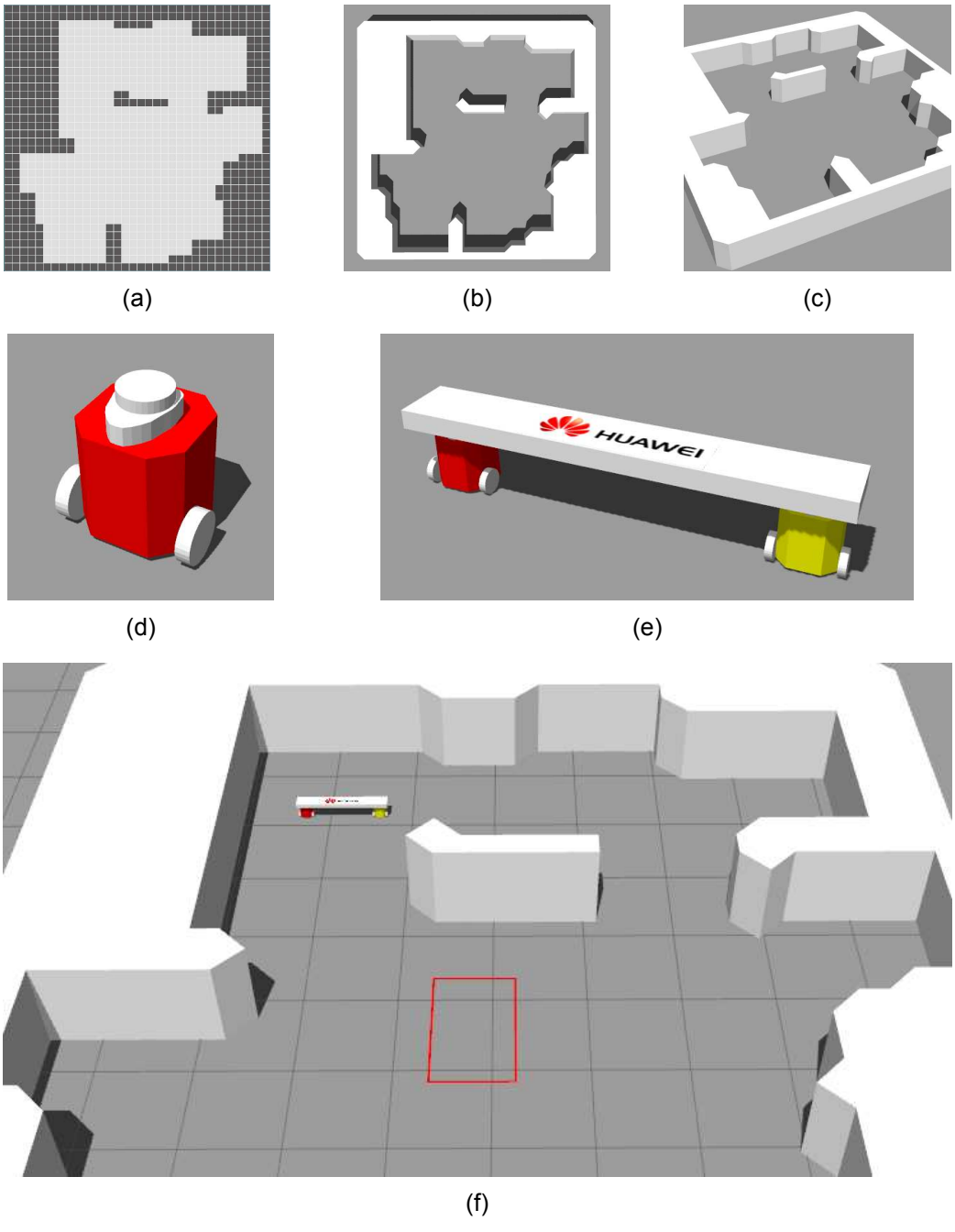


Figure 4.12: Simulated 3D environment and MRS model. The 32x32 2D grid world environment (a) is the input for the image2gazebo conversion tool. The resultant 3D environment is presented from both a top view (b) and a perspective view (c). Each cell within the environment has dimensions of 30x30cm, thereby resulting in an overall environment size of approximately 10x10m. In (d), the 3D Turtlebot3 model is depicted, while (e) showcases the 3D MRS model which includes two Turtlebot3 robots and the transported object. The entirety of the 3D scenario, encompassing the 3D environment, the MRS model, and the target area (depicted as a red rectangle) situated at the center of the map, is illustrated in (f).

is typically realized using a wired connection system. On the other hand, for inter-robot communication wireless technologies are commonly employed.

An overview of the communication's links between the various components is shown in Figure 4.13 while Table 4.5 reports the main steps executed within the main control node.

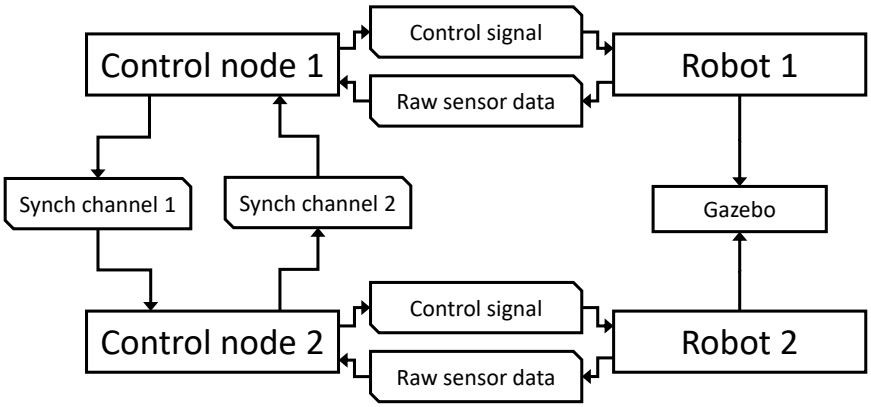


Figure 4.13: Intra and inter-robot communication scheme. The bi-directional intra-robot communication is represented by the control signal and raw sensor data exchanged by the control node and the robot hardware. Additionally, the presence of two synch. channels enable inter-robot communications, allowing the robots to synchronize and coordinate their actions. Lastly, for visualization purposes, Gazebo collects the robot's current state.

Table 4.5 Control node

while goal not reached **do**
  $a^{(self)} \leftarrow$ policy.predict(obs)
  publish $a^{(self)}, q^{(self)}$ to the synchronization channel
  wait until $a^{(other)}, q^{(other)}$ are received from the other robot
  compute $q_f^{(c)}$ using the RotoTranslate procedure
  compute the center path $q_r^{(c)}(s)$ given $q_i^{(c)}$ and $q_f^{(c)}$
  compute the robot path $q_r(s)$ given $q_r^{(c)}$ and (r, ψ)
  compute the robot trajectory $q_r(t)$ given $q_r(s)$ and $s(t)$
  compute the reference velocities $v_r(t)$ and $\omega_r(t)$ given $q_r(t)$
  for $= 0$ to T **do**
    compute pose error e_t given $q(t)$ and $q_r(t)$
    compute the control signal u_t given e_t, $v_r(t)$, and $\omega_r(t)$
    compute the control velocities v_t and ω_t given e_t, u_t, $v_r(t)$, and $\omega_r(t)$
    publish v_t and ω_t to the robot's actuators
  update obs with new robot's positions and new sensed environment's obstacles

Integration of the Localization Algorithm

The parameters associated with the motion model and the AMCL algorithm are often unknown and require fine-tuning based on the results obtained from simulation or real-world experiments. In our case, we have adopted the parameter values reported in study [78], which applies the same models and algorithms presented here to an AGV. Furthermore, Figure 4.14 illustrate how the feedback loop of the tracking controller, specifically the intra-robot communication scheme depicted in Figure 4.13, incorporates the generation of corrupted measurements and the utilization of the sensor-based localization algorithm.

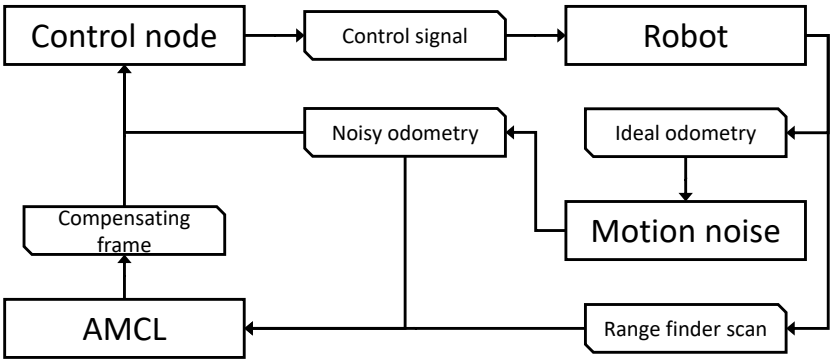


Figure 4.14: ROS communication scheme of the intra-robot control loop enhanced with sensor-based localization algorithm.

5 Results & Discussion

In this chapter, we present the outcomes of the implemented solutions. Firstly, we provide an extensive analysis of the results obtained from the RL planning algorithm, emphasizing its implications in comparison to the existing SoA approaches. Secondly, we showcase the outcomes derived from the path generation and trajectory tracking controller. Lastly, we present the findings obtained from the conducted sensing experiments.

5.1 Training and Performance Results of the RL Planner

Before exploring the multi-agent scenario and realistic indoor environments, we deem it crucial to replicate the experiments outlined in the original VPN article [2]. This preliminary step is essential to validate our adapted implementation of the algorithm before proceeding with cooperative robot scenarios. The authors of the article primarily present the algorithm in a mathematical framework, providing limited details pertaining to its implementation. Consequently, reproducing the results becomes a challenging task. Nevertheless, we consider the results presented by the authors as the standard against which we seek to compare our own implementation.

Recall that our measurement of training time is in steps, which correspond to a single observation-action-reward iteration, namely, the atomic unit in RL settings. For the sake of completeness, we will also provide the total number of episodes required to train the agents. However, note that there is no standardized relationship between these two variables, as the specific implementation parameters can greatly influence the correlation.

All of our training experiments were conducted on an Intel Core i9-12900K processor with 64 GB of main memory. In an effort to accelerate the training process, we explored the option of running the agent's NN on a GPU, specifically the NVIDIA GeForce RTX 4090. However, it is important to emphasize that RL algorithms cannot be directly executed on accelerators. While NNs can be parallelized and effectively executed on GPUs [79], RL environments are primarily designed to run on CPUs as it happens in our case. Consequently, we did not observe significant advantages from utilizing the GPU for training. The bottleneck arises from the cost associated with data transfer between the CPU and GPU, which offsets any potential benefits gained from GPU acceleration.

5.1.1 Single-Agent Scenario

Following the approach proposed in the article, we conduct an analysis using a grid world of varying dimensions, featuring uniformly randomly placed obstacles, and a single agent. Specifically, we evaluate environments ranging from 16×16 to 64×64 in size. The obstacle probability is set at $p = 0.3$, meaning that approximately 30% of the cells within the grid will be occupied by obstacles.

Figure 5.1 illustrates the progression of the training metrics described in subsection 4.2.3 as a function of the training steps. The results demonstrate that the training algorithm successfully converged, leading to a nearly-optimal solution. The plot depicting the policy loss exhibits a decreasing trend, indicating the convergence of the function towards zero, as expected. Furthermore, the entropy loss function displays a similar trend, indicating that the agent is gaining confidence in the choice of actions. Likewise, the plots related to the state-value function approximation exhibit satisfactory convergence, with the value loss (Equation 4.13) steadily decreasing and approaching zero, as expected. Additionally, the explained variance exhibits an upward trend, approaching the unitary value. Following

the completion of the training, the explained variance reached an impressive value of 97.8%, reflecting the high fidelity of the approximated state-value function.

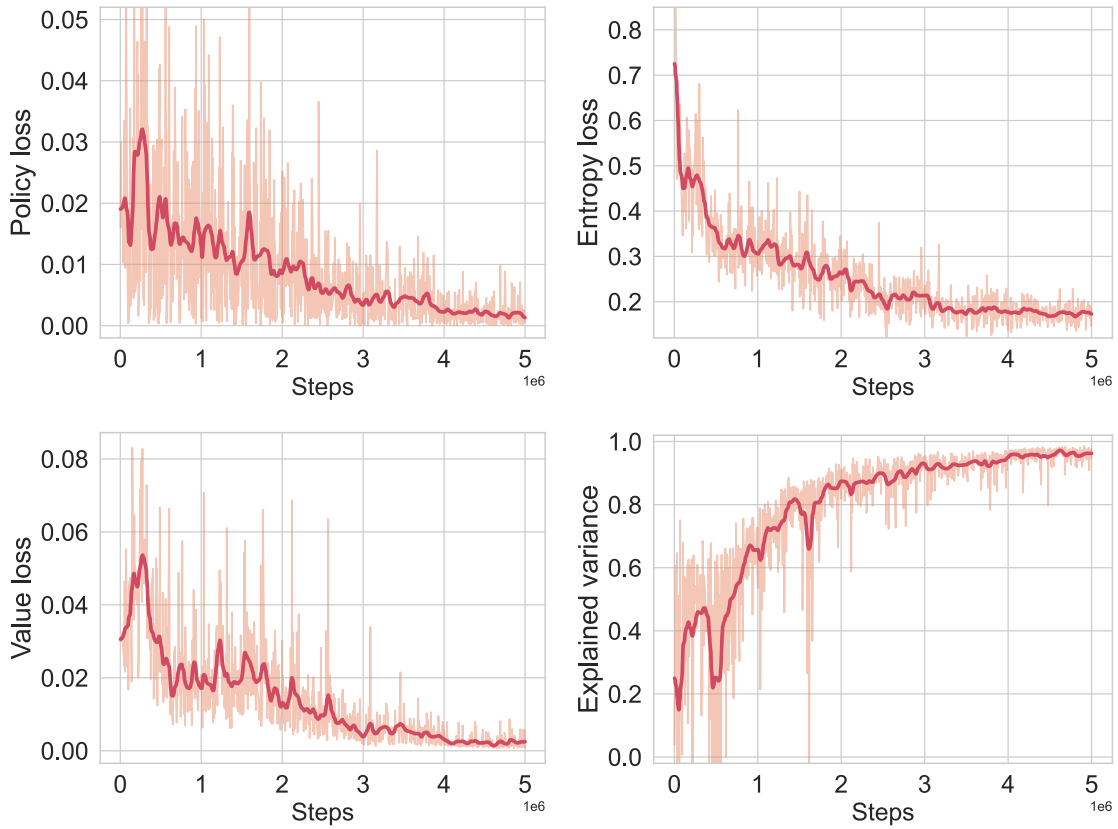


Figure 5.1: The plot illustrates the outcomes of the single-agent training process. Specifically, we present the policy and value loss, along with the entropy loss and the explained variance. The light red background data represents the actual recorded data, while the bold red line is employed for visualization purposes, obtained by applying a Gaussian kernel to the underlying data.

While training metrics provide valuable insights into the learning process, metrics directly measuring the agent's performance are even better indicators. Figure 5.2 depicts the mean episode length and the mean episode rewards throughout the training phase. Additionally, Figure 5.2 presents the optimal episode length, the relative difference metric, and a graph showcasing the updates made by the automatic CL to the maximum spawning distance.

The plotted data highlights the presence of two different training phases. The first phase spans from steps 0 to 2 million, during which the automatic CL algorithm is actively engaged. Throughout this phase, the distance between the agent and the goal is incrementally increased, facilitating a gradual progression in the agent's learning process. The second phase, stemming from step 2 million, marks the agent's transition to the final stage of the curriculum, where it steadily improves its performance towards the optimum. The division between these two phases is clearly discernible in Figure 5.2 (bottom right), which visualizes the points at which the curriculum is updated. In this specific experiment, the target maximum distance between the goal and the agent was set to 64 cells.

The first phase can be further subdivided into two distinct stages. The initial stage, span-

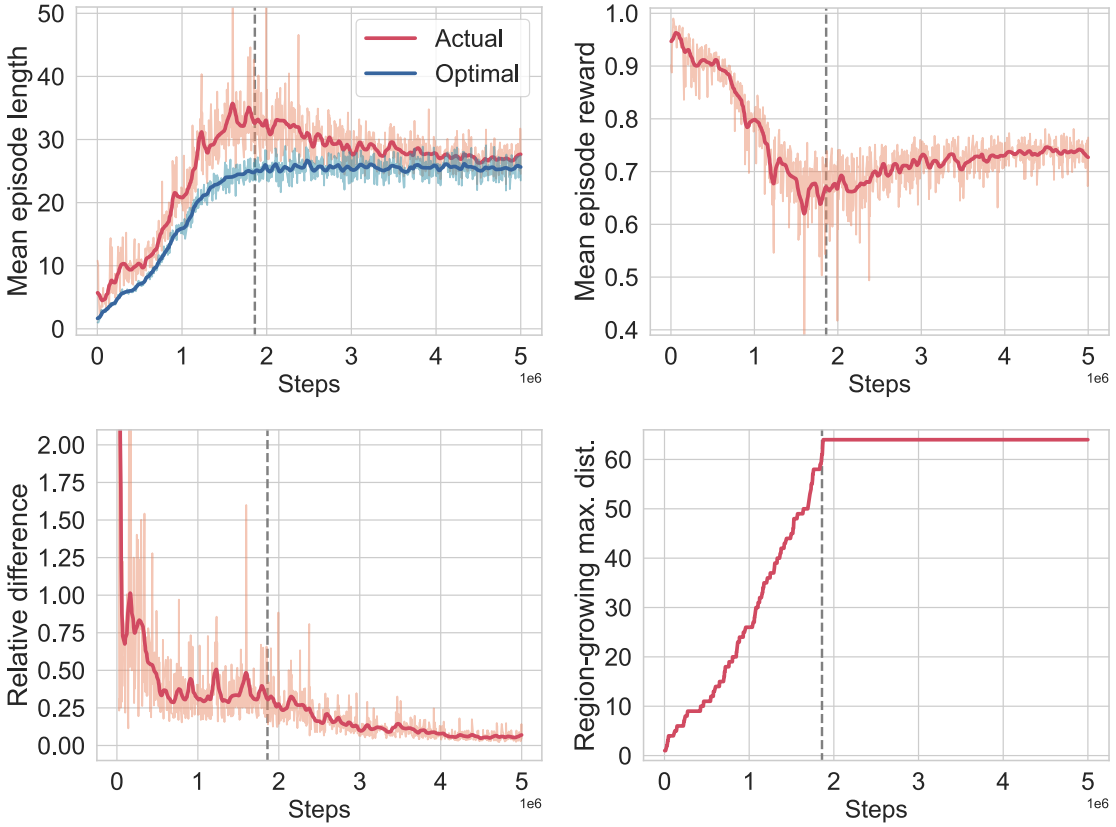


Figure 5.2: The plots showcase the performance metrics of the single agent throughout the training phase. Specifically the mean episode length contains the agent's mean episode length (in red) and the optimal mean episode length (in blue). The dashed grey line indicates the transition between the two training phases.

ning from 0 to 100,000 steps, is characterized by the agent transitioning from random movements to understanding the objective of reaching the goal. This is evident from the mean episode length plot, which initially exhibits a relatively high value, despite the goal being only one cell away. Once the agents grasp the objective, the CL protocol is employed to gradually increase the difficulty level until the target task is achieved. The curriculum undergoes updates when the relative difference metrics fall below 0.2. This threshold indicates that, on average, the agent's path is more than 20% longer than the optimal path when the curriculum is considered complete. This level of performance is suboptimal, as the agent's paths should ideally closely align with the optimal path.

The second phase of training is therefore primarily focused on optimizing the length of the agent's path. Over the course of the subsequent 3 million steps, the relative difference metric consistently decreases, resulting in the agent's path being less than 5% longer than the optimal path. One alternative approach would be to set a lower curriculum update threshold right from the beginning, ensuring that the agent's path is already close to the optimal one upon completing the curriculum sub-tasks. However, adopting this approach would not necessarily expedite the training process, as each stage of the curriculum would likely require more time to achieve the desired level of performance.

Lastly, note that the mean episode reward plot exhibits a similar trend to that of the mean episode length, albeit inverted and re-scaled.

Table 5.1 presents the results in terms of win rate and the relative difference between our findings and those reported in the VPN article. While our results demonstrate the great performance of the algorithm, it should be noted that the exact definition of these metrics used by the authors remains somewhat unclear. For instance, a variation in the threshold that determines a successful episode (in our case, 100 steps) would yield different win rates and subsequently different values for the $RelDiff$ metric. However, despite these potential variations, our findings indicate that the training process converges, and the overall performance achieved aligns with our intended objectives, thus validating our implementation for our scope and purpose.

Train map size	Test map size	Training duration step, episodes, time	Win rate SoA	Win rate Our	RelDiff SoA	RelDiff Our
16x16	16x16	2.5M, 70K, 1.1h	94.4	99.8	0.2	0.09
	32x32		n.a.	98.6	n.a.	0.45
32x32	32x32	5M, 150, 5.7h	68.8	99.6	0.4	0.13
	64x64		n.a.	92.9	n.a.	0.49
64x64	64x64	10M, 280K, 18.2h	53.2	93.4	0.5	0.21

Table 5.1: Comparison of after-training performance metrics of the single-agent with the SoA. In our study, the win rate is averaged over 1000 episodes, and the $RelDiff$ metric is computed exclusively for successful episodes. The authors of the SoA article did not provide information regarding the training time and steps, but only specified the number of episodes: 30,000 for all map sizes. Additionally, the agent trained on 16×16 and 32×32 maps was tested on larger maps (32×32 and 64×64, respectively) to evaluate its generalization capabilities.

5.1.2 Cooperative Scenario

In order to address the cooperative transportation scenario, we initially seek to replicate the scenario proposed in the current state-of-the-art study on cooperative carrying using DQN [13]. The replication of the environment, utilizing grid world models is depicted in Figure 4.1b. This specific scenario provides a straightforward evaluation of the robot's ability to cooperate, as cooperation is the sole means by which the robots can reach their objective. In our scenario, the goal is situated in the lower right corner, and spans a 5×5 area, while the MRS is randomly spawned within the left half of the map.

To address this scenario, we employed pre-trained agents. Specifically, we initially trained the agents individually, following the approach described in the previous section, on maps featuring randomly positioned obstacles. Subsequently, utilizing the NN weights obtained from this initial training phase as a starting point, we changed the scenario to the cooperative one. We named this approach single-to-multi-robot manual CL. The outcomes of the second training stage are presented in Figure 5.3.

A notable distinction compared to the single-agent scenario is the convergence speed. The algorithm achieves convergence in a mere 500,000 steps. This quantity of steps corresponds to approximately a couple of hours of training, signifying a substantial enhancement over the DQN approach, which, as indicated by the authors' findings, requires several days to complete the training procedure. This relatively rapid convergence can be attributed to the agents being pre-trained and the map being smaller and static. Nonetheless, this outcome serves as a compelling demonstration of our algorithm's capability to acquire and adapt to cooperative dynamics.

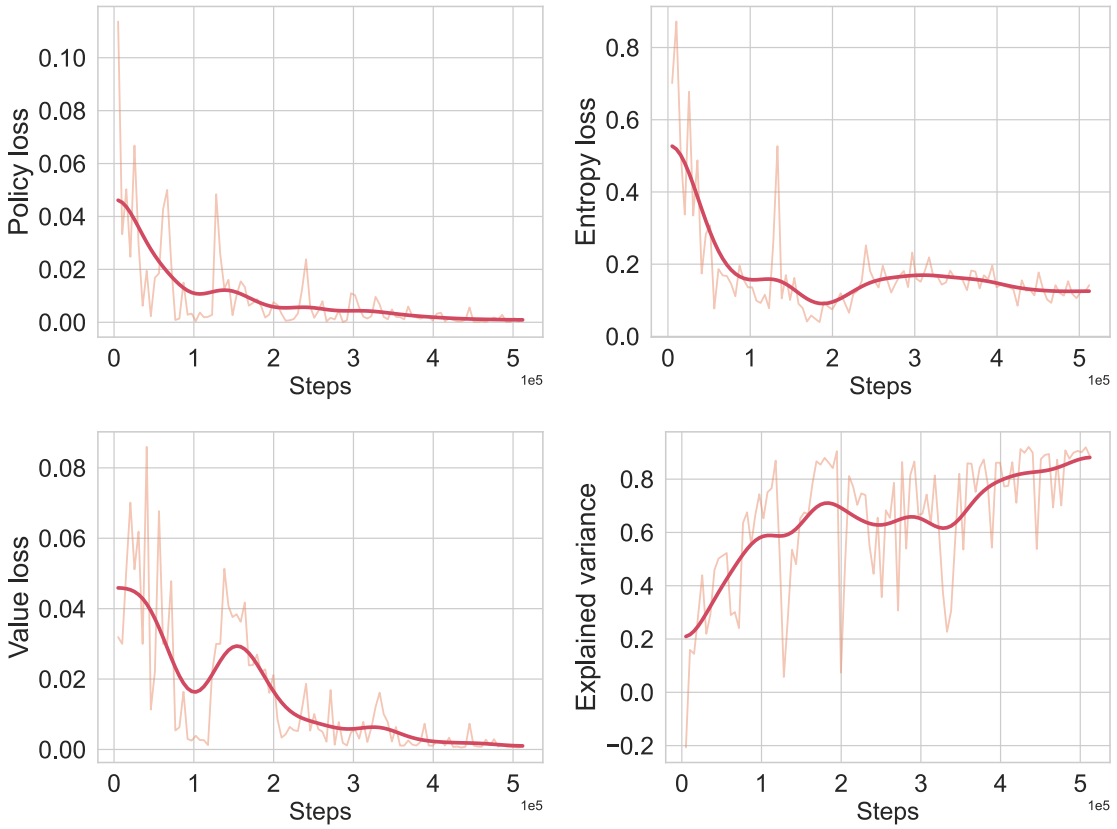


Figure 5.3: The plot illustrates the training metrics of the cooperative multi-agent scenario in the presence of a static environment.

5.1.3 Indoor Scenario

In line with the previous scenario, also in this context, we employ the single-to-multi-robot curriculum. However, the cooperative stage of the training process proves to be considerably more challenging due to the dynamic nature of the environment. In fact, the environment is randomly generated at each episode using the ArenaBench tool. Moreover, the presence of moving obstacles, constituting 5% of the total cells, further adds to the complexity as their positions change at each step.

Figure 5.4 presents the outcomes of the second stage of the training process. Analogous to the initial single-agent scenario, the plots within the figure can be partitioned into two distinct sections. The first segment corresponds to the automatic region-growing curriculum, wherein the complexity of the scenario progressively escalates. The subsequent part of the figure is dedicated to the refinement of the agent's policy.

In this case, the threshold for the $RelDiff$ metric must be set significantly higher. The rationale behind this adjustment lies in the fact that the MRS functions as a solitary agent for the purpose of calculating the optimal path length, hence neglecting the intricate manoeuvres the MRS might need to undertake. Furthermore, the optimal path computation occurs at the beginning of each episode, thereby failing to account for the presence of moving obstacles. For this specific experiment, we determined the threshold to be 2.0, accounting for these considerations.

In order to gain deeper insights into the cooperative aspects of the training process, we have employed the re-formulation of the ΔQ metric tailored to V-value-based agents.

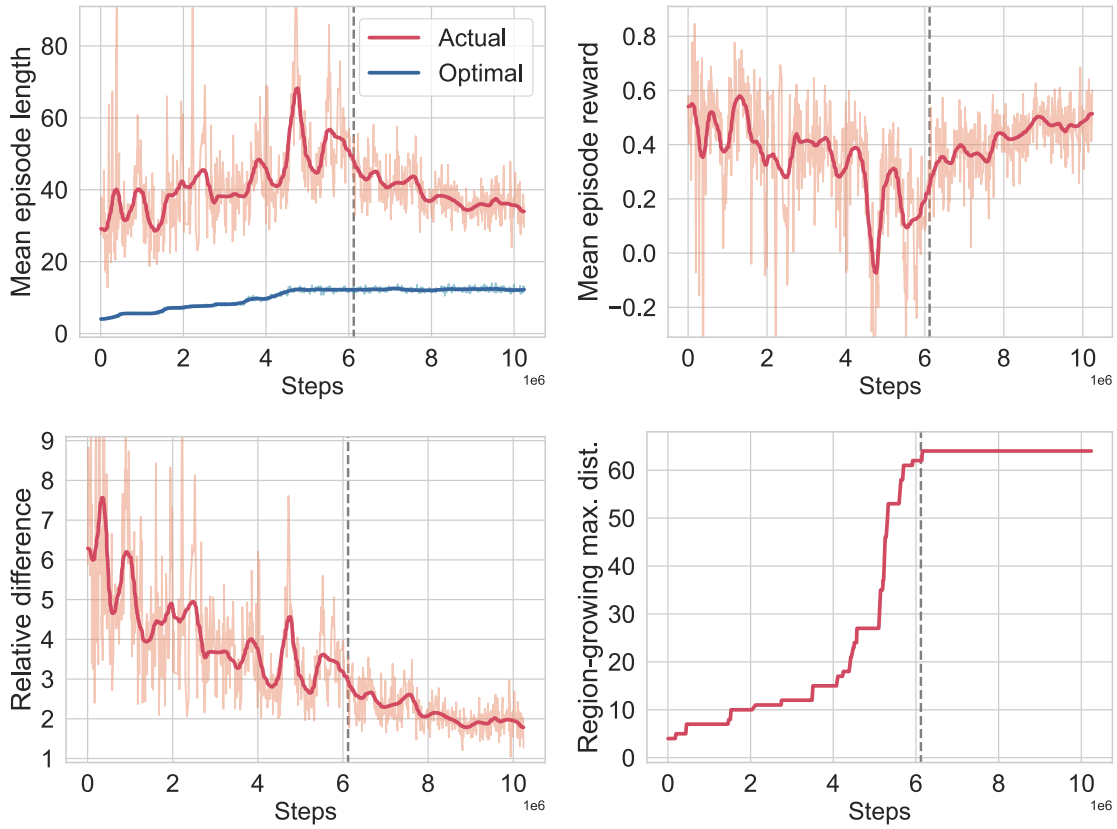


Figure 5.4: The plot presents the training metrics of the cooperative multi-agent scenario in the context of an indoor dynamic environment.

The resulting values are depicted in Figure 5.5. Although the initial phase of the metric is not particularly informative due to its susceptibility to the rapidly changing region-growing curriculum, the subsequent phase effectively corroborates the observations made by the original ΔQ metric's authors. Starting from the 6 millionth step and continuing until the end, the plot exhibits a consistent downward trend, which indicates a progressive enhancement in the level of cooperation between the robots.

Through empirical observation, we have discovered a seemingly counter-intuitive behaviour exhibited by the agent when trained with randomly moving obstacles. Specifically, the MRS demonstrates a preference for maintaining a considerable distance from these dynamic obstacles, despite this choice resulting in a longer path towards the goal. At first glance, it may appear that the agents are not effectively learning. However, their strategic choice is justified by the potential hindrances posed by closely approaching a randomly moving obstacle. Such proximity could significantly impede the manoeuvrability of the MRS. Consequently, the agents exhibit a preference for taking locally sub-optimal steps, ultimately leading to superior global outcomes.

5.1.4 Integration with the Path Generation Algorithm

This section is dedicated to presenting the outcomes achieved through the integration of the RL planner with the path generation algorithm. An exemplification of this integration is depicted in Figure 5.6. The devised RL algorithm formulates plans based on the grid world representation of the environment, and subsequently, its directives are seamlessly translated into continuous paths in the real-world domain.

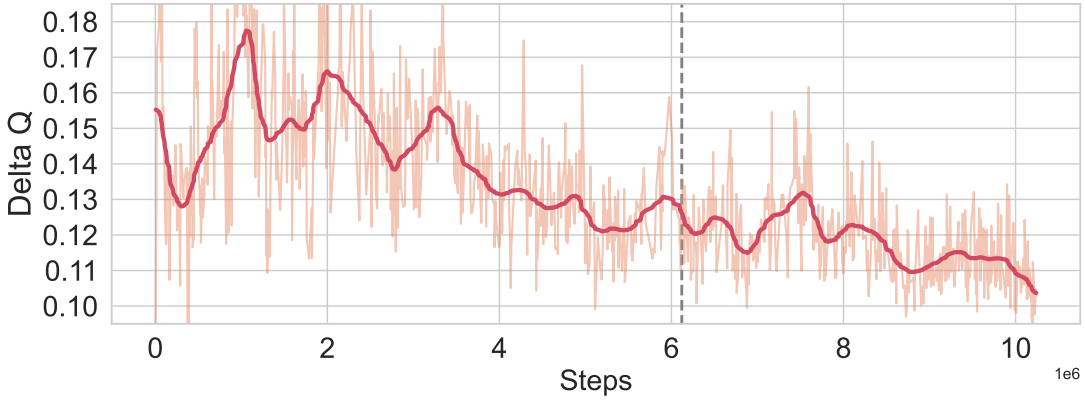


Figure 5.5: The plot depicts the trend of the ΔQ metrics during multi-agent training in the indoor dynamic environment.

5.1.5 Integration with 6G Radio Coverage Constraints

Figure 5.7 illustrates the outcomes obtained from the pathloss computation. It demonstrates the integration of virtual obstacles into the environmental map, showcasing the areas where the radio signal strength is insufficient. Additionally, the figure depicts the planned trajectory of the MRS as it strategically avoids these low-coverage areas.

5.1.6 Comparison with the SoA

In comparison to the current SoA approach that utilizes DQN, our proposed approach possesses the capability to plan in dynamically changing environments. During the execution phase, if the environment undergoes modifications, each planning agent within our framework adapts its actions accordingly. This dynamic planning ability is facilitated by the input map that serves as the interface between the real world and the agent. In contrast, the DQN approach relies on agents ”memorizing” the surrounding environment structure through collisions with walls. Consequently, if the environment changes, all the acquired knowledge becomes obsolete, necessitating a restart of the training process. In contrast, our agents have acquired the ability to take optimal actions that lead towards the goal based on the current state of the environment. It is worth noting that in real-world scenarios, an a priori map of the environment may not always be available. However, assuming the robot can sense the obstacles in the vicinity of its current position, our algorithm can still guide the robot towards the goal. This may not be achieved in an optimal manner, but rather in a similar way to how a flood-fill path-finding algorithm would operate [80]. It is important to acknowledge the trade-off between the level of a priori information about the map and the optimality of the generated paths. In cases where no information is available, the transportation tasks are accompanied by an additional exploration task.

A limitation of our approach is that the planning process takes place within a discretized world, which inherently restricts the accuracy of the environment map. While discrete maps are commonly employed in robotics, the drawback of our algorithm lies in the size of the map that the algorithm can effectively process. In our experiments, we utilized a 32×32 grid world configuration, resulting in block sizes of 30×30cm to represent a 10×10 meter room. This level of granularity may be insufficient to accurately capture the details of the environment. Although it is possible to increase the resolution of the grid world, such enhancement would come at the cost of significantly increased training time. Moreover, the algorithm’s output, which comprises discrete actions, also contributes to the lack of smoothness in the resulting path executed by the robot.

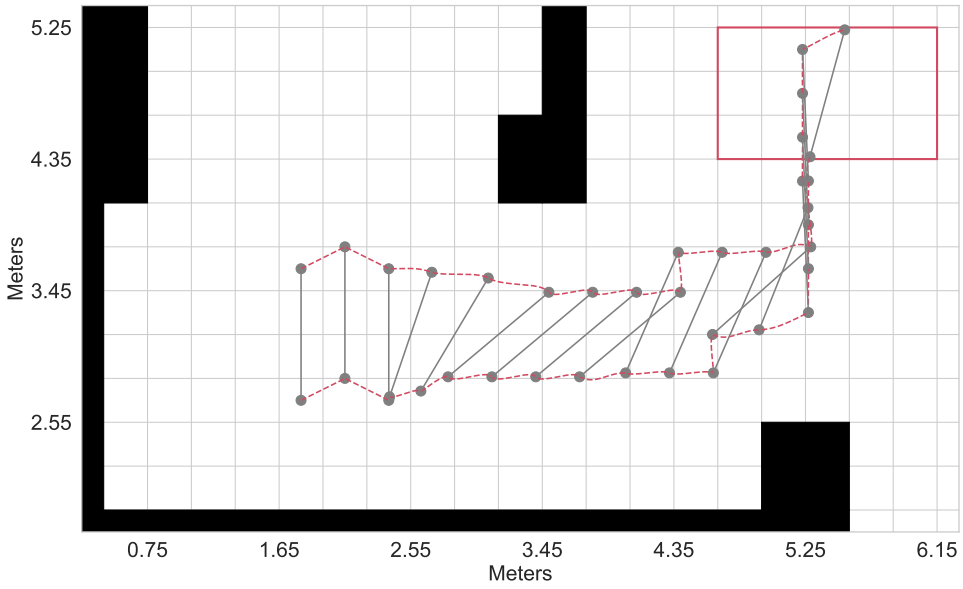


Figure 5.6: An exemplification of the RL planner integrated with the path generation algorithm, resulting in a continuous global path guiding the MRS from its initial pose (located at the bottom left) towards the goal area (positioned at the top right).

5.2 Tracking Controller Performance

To validate the performance of the trajectory tracking controller we performed experiments using the developed simulation framework. Specifically, we use the simulated Turtlebot3 robot and made it follow a trajectory designed with the Cartesian polynomial method, such that it emulates the paths that the robot will follow when part of the MRS. To isolate the controller from other factors, we took advantage of the perfect knowledge of the robot pose the simulation environment provides. Figure 5.8 shows the results. To further test the stability of the controller we also tested the cases where the initial robot's pose is different from the reference one.

To assess the effectiveness of the trajectory tracking controller, we conducted experiments utilizing our custom simulation framework. In particular, we employed the simulated Turtlebot3 robot to follow a trajectory generated using the Cartesian polynomial method. In order to isolate the controller's performance from other factors, we leveraged the precise knowledge of the robot's pose that the simulation environment provided. The outcomes of these experiments are presented in Figure 5.8 . To further evaluate the stability of the controller, we also conducted tests where the initial pose of the robot differed from the reference pose.

The obtained results demonstrate the capability of the controller to asymptotically track the desired trajectory, even when the initial conditions differ from the reference pose. It is important to note that the controller implemented in the simulation is a discretized version of the controller described in subsection 4.3.2. To achieve this discretization, we utilized the forward Euler method with a sampling interval of 10ms.

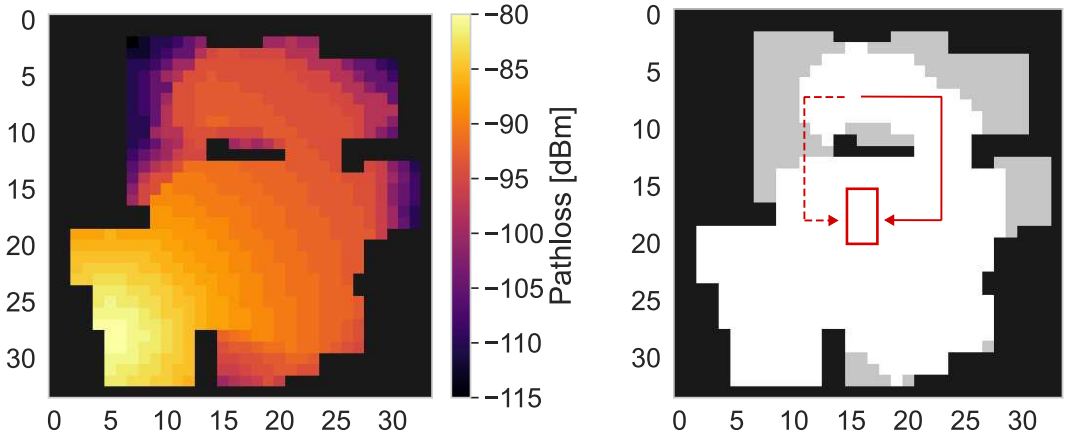


Figure 5.7: On the left side, the computation of pathloss values is performed utilizing the preexisting map of the environment. The access point is positioned in proximity to the bottom left corner of the map. On the right side, the outcomes are displayed after applying a threshold to the pathloss values (in this instance, set at -95 dBm). Virtual obstacles are depicted in a light grey shade. The Multi-Robot System is initially positioned in the upper portion of the map, and in order to reach the designated goal (red rectangle), it strategically opts for a longer but safer route (solid line).

5.3 Localization Results and Experiments

The outcomes of applying the localization metrics described in subsection 4.3.4 to the scenario depicted in Figure 5.6 are illustrated in Figure 5.9. This figure presents a comparison among three different scenarios: the ideal case where the robots possess perfect knowledge of their true position, the scenario where odometry-based localization is employed with simulated noisy odometry using the procedure outlined in Table 4.4, and the utilization of the AMCL algorithm with LIDAR measurements.

The plotted data clearly demonstrates that relying solely on odometry-based localization leads to system failure. In this case, the global position error reaches approximately 30cm. This means that the robot may be dangerously close to colliding with obstacles without the planning algorithm being aware of this proximity. Likewise, the local position error also reaches similar values, around 20cm, which undoubtedly compromises the stability of the transported object. It is important to note that in our scenario, where the transported object is approximately 1 meter in length, an acceptable threshold for the local error would be around 5 or 10cm. Conversely, when utilizing the AMCL algorithm, both the global and local error remain below the 3cm threshold. This represents a remarkable achievement. It is worth mentioning that even in the case of perfect localization (ideal scenario), both metrics do not report zero error. This discrepancy arises due to the non-ideal response of the control system in following the reference signal.

Figure 5.10 presents a comparison between the outcomes obtained by applying the AMCL algorithm with LIDAR measurements and the same algorithm utilizing emulated 6G sensing devices. In order to emulate the performance of these sensors, we took advantage of the shared underlying measurement model with LIDAR sensors. To achieve this emulation, we amplified the noisy variance of the LIDAR measurements by factors of 5 and 10, thereby downgrading its accuracy. This comparison allows us to assess the effectiveness and robustness of the AMCL algorithm when operating with different sensing devices.

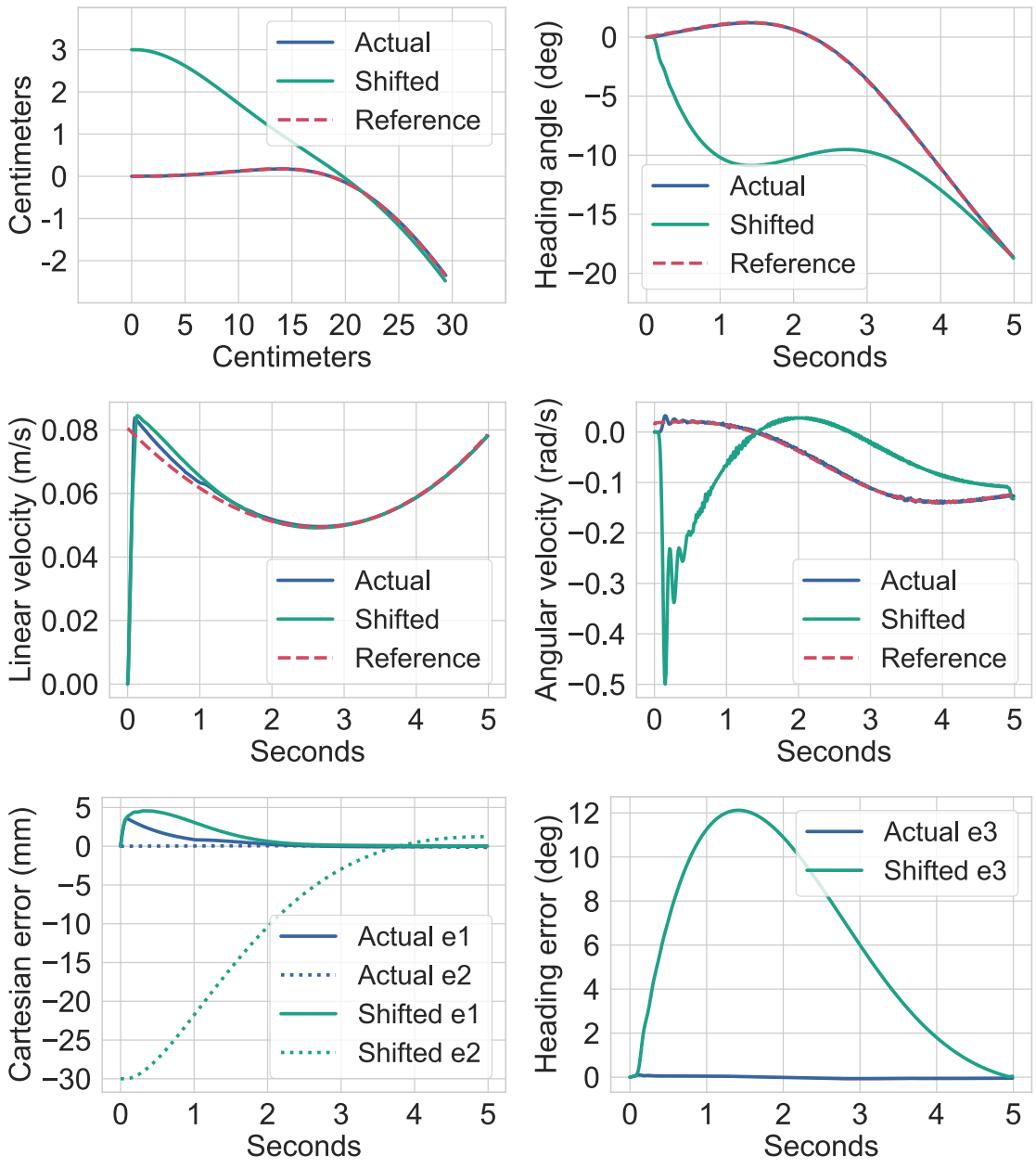


Figure 5.8: Controller performances in tracking a reference trajectory. The reference trajectory and velocities are shown in red, while the actual robot trajectory and velocities are shown in blue. Additionally, we report in green the robot's trajectory and velocities for the case where the initial condition does not match the reference one (the robot shifted on the y-axis of about 3cm). The Cartesian trajectories in the 2D operational plane and the heading angle trajectory as a function of time are shown in the top plots. The reference linear and angular velocities as well as the actual velocities are reported in the middle. The Cartesian (e_1 solid line, e_2 dotted line) and angular error (e_3) are depicted on the bottom.

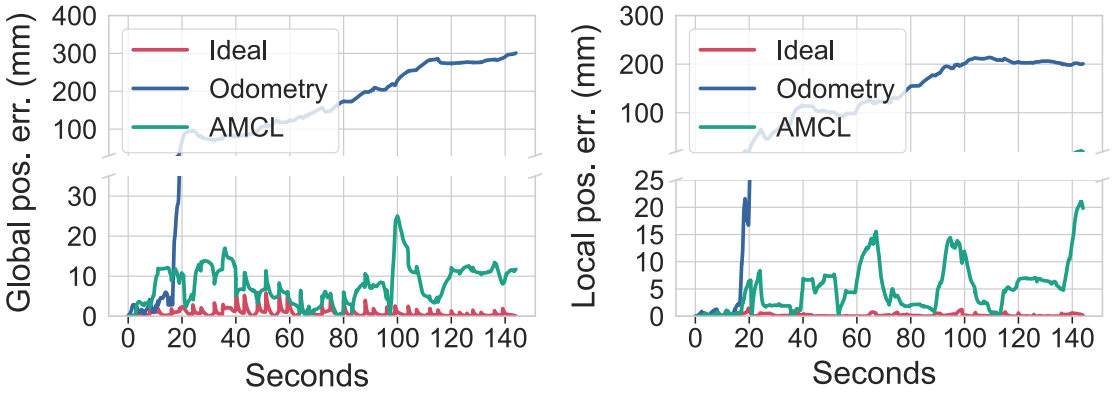


Figure 5.9: Global (on the left) and Local (on the right) position error metrics comparing different localization strategies.

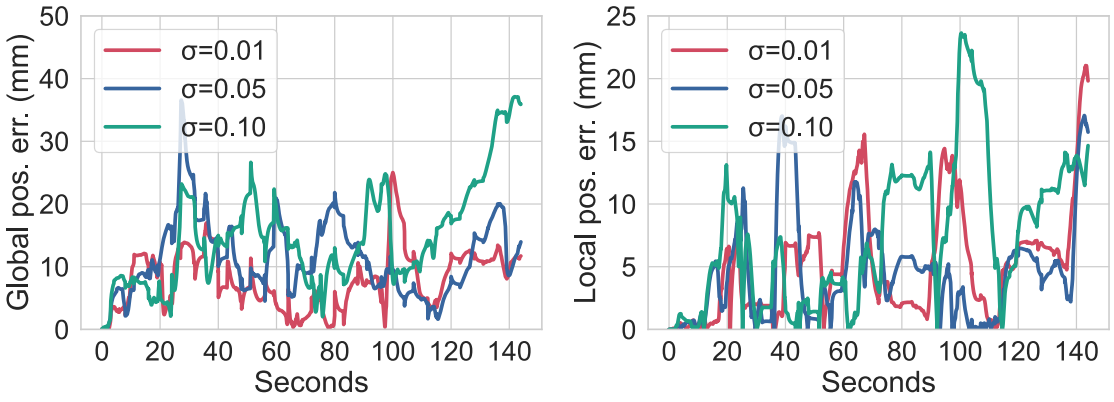


Figure 5.10: Global (on the left) and Local (on the right) position error metrics comparing the AMCL localization algorithm with different measurements noise levels.

The results demonstrate that despite the diminished accuracy in measurements, the localization performance remains largely unaffected. Thus this experiment represents a first step in proving the viability of such sensing devices in robotic scenarios.

6 Conclusions & Future Work

6.1 Conclusions

In this thesis, our primary focus revolved around the design of multi-robot motion planning for the cooperative carrying task. Our specific contribution lies in proposing an adapted version of the VPN algorithm [2], with the underlying belief that RL algorithms can outperform traditional approaches in the context of MRSs. Our research yielded two key improvements over the current SoA. Firstly, we demonstrated that the theoretical framework underlying VPNs can effectively address multi-agent scenarios. Secondly, compared to the SoA RL algorithm for cooperative MRSs planning, namely the DQN approach presented in [13], our solution exhibits superior adaptability in dynamic environments with moving obstacles, without requiring re-training procedures.

To validate our algorithm, we conducted a proof-of-concept implementation using modern RL software frameworks. The performance of our algorithm was rigorously evaluated by comparing it against current SoA approaches. The comprehensive results obtained from these evaluations demonstrate the effectiveness of our algorithm, not only in terms of overall performance but also in terms of training time and across various evaluation metrics. Of particular significance is our extension of the ΔQ metric, which serves as additional evidence that the agents successfully acquired cooperative behaviours through the learning process.

Additionally, our RL framework serves as a 2D visualization tool, however, by recognizing that real-world robots operate in a three-dimensional space, we dedicated substantial efforts to create a simulation platform that accurately reflects this reality. We provided a detailed description of the development process of this platform, encompassing the construction of the 3D environment, the modelling of the MRS based on the Turtlebot3 robot, and the seamless orchestration of the various components through ROS. In the process of building this simulation platform, we seized the opportunity to investigate other crucial elements in robot navigation, specifically path generation algorithms and trajectory tracking controllers. Our research successfully demonstrated the effectiveness of the proposed multi-robot trajectory generation algorithm in maintaining a consistent inter-robot distance, a crucial requirement for the carrying task.

To enhance the realism of the simulation, we implemented a sensor-based localization algorithm, the AMCL. We conducted a comparative analysis between this algorithm and odometry-based localization, and evaluated the accuracy based on the global and local position error metrics. The results of our study demonstrated that the integration of LIDAR-based localization through AMCL significantly improves reliability and accuracy, ensuring the successful completion of the mission.

Lastly, in anticipation of the growing integration of telecommunication and robotics as a cohesive functional system, we presented an innovative concept that leverages 6G signal connectivity, using E-band spectrum as communication technology. With this technique, we enabled the robot to intelligently select routes that prioritize stronger network connectivity, utilizing E-band communication as our experimental platform. Furthermore, in line with the ongoing development of 6G ISAC technologies, we conducted preliminary tests to evaluate the feasibility of utilizing 6G sensing devices for robot localization. Remarkably, these tests revealed that the performance of 6G sensors closely aligns with that of the current standard in the field, i.e. LIDARs, when used for localization purposes.

6.2 Future Work

One of the most significant trade-offs we encountered during the design of the RL motion planning algorithms was the adoption of a coarse grid map to represent the world, coupled with the restriction of agent actions solely to adjacent cell movements. This decision led to a drawback in which the map fails to capture the smaller details present in the actual environment, while the constrained output of the agents underutilizes the robot's mobility capabilities. To address these limitations, forthcoming research should focus on revising the network architecture to encompass a larger set of actions, and on further optimizing the training algorithm to enable fast learning on high-resolution maps.

Moreover, the multi-robot path generation algorithm we have developed suffers from a similar limitation, wherein the generated paths exhibit local smoothness, i.e. between consecutive checkpoints, but lack a global optimization step. Consequently, the MRS must readjust its alignment at each checkpoint before advancing to the subsequent one. To address this issue, future investigations should focus on the integration of the subsequent checkpoints into the path generation algorithm. This enhancement will enable the algorithm to consider the future targeted direction, thereby fostering globally smooth paths.

Finally, an additional area of focus would involve the expansion of the developed cooperative robotic simulator to incorporate enhanced telecommunication components. Including telecommunication aspects would introduce realism to the data exchange mechanism between robots, encompassing factors like delays and packet loss. Moreover, a robotic simulation platform would provide dynamic scenarios with stringent telecommunication and perception requirements. These scenarios would serve as benchmarks for ongoing 6G development, thereby facilitating the evaluation and advancement of novel ISAC devices. The reciprocal interaction between robotics and telecommunication simulations presents a unique opportunity for mutual enhancement, benefiting both fields and contributing to the progress of cutting-edge technologies in a symbiotic manner.

Bibliography

[1] *World Robotics Industrial Robots and Service Robots*. https://ifr.org/worldrobotics/. 2022.

[2] Nantas Nardelli et al. "Value Propagation Networks". In: *CoRR* abs/1805.11199 (2018). arXiv: 1805.11199. url: http://arxiv.org/abs/1805.11199.

[3] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. "A Survey and Analysis of Multi-Robot Coordination". In: *International Journal of Advanced Robotic Systems* 10.12 (2013), p. 399. doi: 10.5772/57313. eprint: https://doi.org/10.5772/57313. url: https://doi.org/10.5772/57313.

[4] Yulin Wang. "What Are Cooperative Robots and Collaborative Robots?" In: *IDTechEX* (2022). url: https://www.idtechex.com/en/research-article/what-are-cooperative-robots-and-collaborative-robots/28376#:~:text=As%20a%20relatively%20new%20definition,between%20human%20operators%20and%20themselves..

[5] Elio Tuci, Muhanad H. M. Alkilabi, and Otar Akanyeti. "Cooperative Object Transport in Multi-Robot Systems: A Review of the State-of-the-Art". In: *Frontiers in Robotics and AI* 5 (2018). issn: 2296-9144. doi: 10.3389/frobt.2018.00059. url: https://www.frontiersin.org/articles/10.3389/frobt.2018.00059.

[6] Jun Ota. "Multi-agent robot systems as distributed autonomous systems". In: *Advanced Engineering Informatics* 20.1 (2006), pp. 59–70. issn: 1474-0346. doi: https://doi.org/10.1016/j.aei.2005.06.002. url: https://www.sciencedirect.com/science/article/pii/S1474034605000509.

[7] Michael W Otte. "A survey of machine learning approaches to robotic path-planning". In: *University of Colorado at Boulder, Boulder* (2015).

[8] B. Siciliano et al. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2010. isbn: 9781849966344.

[9] Zichen He, Jiawei Wang, and Chunwei Song. "A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures". In: *CoRR* abs/2108.13619 (2021). arXiv: 2108.13619. url: https://arxiv.org/abs/2108.13619.

[10] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2021. isbn: 9781292401133. url: https://books.google.com.hk/books?id=B4xczgEACAAJ.

[11] Zhi Yan et al. "Metrics for performance benchmarking of multi-robot exploration". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 3407–3414. doi: 10.1109/IROS.2015.7353852.

[12] Nelson Munoz Ceballos, Jaime Valencia Velasquez, and Nelson Ospina. "Quantitative Performance Metrics for Mobile Robots Navigation". In: Mar. 2010. isbn: 978-953-307-076-6. doi: 10.5772/8988.

[13] Lin Zhang et al. "Decentralized Control of Multi-Robot System in Cooperative Object Transportation Using Deep Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 184109–184119. doi: 10.1109/ACCESS.2020.3025287.

[14] Sutton R. S. and Barto A. G. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Massachusetts; London, England, 2018. isbn: 9780262193986. url: https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf.

[15] Tanmay Gangwani. *Value Iteration, Policy Iteration and Policy Gradient*. Oct. 2019. url: https://yuanz.web.illinois.edu/teaching/IE498fa19/lec_16.pdf.

[16] Luíza Caetano Garaffa et al. “Reinforcement Learning for Mobile Robotics Exploration: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–15. doi: 10.1109/TNNLS.2021.3124466.
[17] C. J. C. H. Watkins. “Learning from Delayed Rewards”. PhD thesis. King’s College, Oxford, 1989.
[18] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: (2013). cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. url: http://arxiv.org/abs/1312.5602.
[19] Bruno Bouzy and Guillaume Chaslot. “Monte-Carlo Go Reinforcement Learning Experiments”. In: *2006 IEEE Symposium on Computational Intelligence and Games*. 2006, pp. 187–194. doi: 10.1109/CIG.2006.311699.
[20] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8 (1992), pp. 229–256.
[21] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. url: http://arxiv.org/abs/1801.01290.
[22] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].
[23] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. url: http://arxiv.org/abs/1707.06347.
[24] David Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. Beijing, China: JMLR.org, 2014, pp. I–387–I–395.
[25] Yang Cheng, Mark Maimone, and Larry Matthies. “Visual odometry on the Mars Exploration Rovers”. In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 1. 2005, 903–910 Vol. 1. doi: 10.1109/ICSMC.2005.1571261.
[26] Muhammad Khan et al. “Investigation of Widely Used SLAM Sensors Using Analytical Hierarchy Process”. In: *Journal of Sensors* 2022 (Jan. 2022), pp. 1–15. doi: 10.1155/2022/5428097.
[27] Huijuan Zhang et al. “Localization and navigation using QR code for mobile robot in indoor environment”. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2015, pp. 2501–2506. doi: 10.1109/ROBIO.2015.7419715.
[28] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2005. isbn: 9780262201629. url: https://books.google.com.hk/books?id=2Zn6AQAAQBAJ.
[29] Rudolph Emil Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.
[30] Brian D. O. Anderson. *Optimal Filtering*. eng. Dover Books on Electrical Engineering. Newburyport: Dover Publications, 2012. isbn: 0-486-13689-2.
[31] Yoon-Gu Kim et al. “Localization strategy based on multi-robot collaboration for indoor service robot applications”. In: *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2013, pp. 225–226. doi: 10.1109/URAI.2013.6677348.
[32] Feng Gu et al. “Experimental study of vision sensor based multiple robots active cooperative observation using multi-RFRs testbed”. In: *2011 IEEE International Conference on Robotics and Biomimetics*. 2011, pp. 2417–2418. doi: 10.1109/ROBIO.2011.6181665.
[33] Zhenyi Chen et al. “6G Mobile Communications for Multi-Robot Smart Factory”. In: *Journal of ICT Standardization* 9.03 (Dec. 2021), pp. 371–404. doi: 10.13052/

jicts2245-800X.934. url: https://journals.riverpublishers.com/index.php/JICTS/article/view/7121.

[34] Alireza Bayesteh et al. “Integrated Sensing and Communication (ISAC) — From Concept to Practice”. In: *Communications of HUAWEI RESEARCH* (2022). url: https://www.huawei.com/en/huaweitech/future-technologies/integrated-sensing-communication-concept-practice.

[35] Marina Lotti et al. *Radio SLAM for 6G Systems at THz Frequencies: Design and Experimental Validation*. 2022. arXiv: 2212.12388 [eess.SP].

[36] Carlos De Lima et al. “Convergent Communication, Sensing and Localization in 6G Systems: An Overview of Technologies, Opportunities and Challenges”. In: *IEEE Access* 9 (2021), pp. 26902–26925. doi: 10.1109/ACCESS.2021.3053486.

[37] *6G: The Next Horizon: From Connected People and Things to Connected Intelligence*. Cambridge University Press, 2021. doi: 10.1017/9781108989817.

[38] *6G and Robotics*. Jan. 2023. url: https://one6g.org/new-one6g-position-paper-on-6g-robotics/#.

[39] Soon-Jo Chung et al. “A Survey on Aerial Swarm Robotics”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 837–855. doi: 10.1109/TRO.2018.2857475.

[40] Hyeonbeom Lee, Hyoin Kim, and H. Jin Kim. “Planning and Control for Collision-Free Cooperative Aerial Transportation”. In: *IEEE Transactions on Automation Science and Engineering* 15.1 (2018), pp. 189–201. doi: 10.1109/TASE.2016.2605707.

[41] Vojtech Spurny et al. “Cooperative Transport of Large Objects by a Pair of Unmanned Aerial Systems using Sampling-based Motion Planning”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2019, pp. 955–962. doi: 10.1109/ETFA.2019.8869298.

[42] Hyeonbeom Lee, Clark Youngdong Son, and H. Jin Kim. “Collision-Free Path Planning for Cooperative Aerial Manipulators Under Velocity and Curvature Constraints”. In: *IEEE Access* 7 (2019), pp. 171153–171162. doi: 10.1109/ACCESS.2019.2946273.

[43] Stephanie Kamarry Alves De Sousa et al. “Two-Layers Workspace: A New Approach to Cooperative Object Transportation With Obstacle Avoidance for Multi-Robot System”. In: *IEEE Access* 10 (2022), pp. 6929–6939. doi: 10.1109/ACCESS.2022.3140857.

[44] A. Yamashita et al. “Motion planning of multiple mobile robots for Cooperative manipulation and transportation”. In: *IEEE Transactions on Robotics and Automation* 19.2 (2003), pp. 223–237. doi: 10.1109/TRA.2003.809592.

[45] Vinayak Honkote et al. “Design and Integration of a Distributed, Autonomous and Collaborative Multi-Robot System for Exploration in Unknown Environments”. In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. 2020, pp. 1232–1237. doi: 10.1109/SII46433.2020.9025810.

[46] Junyan Hu et al. “Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning”. In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14413–14423. doi: 10.1109/TVT.2020.3034800.

[47] Jianing Chen et al. “Occlusion-Based Cooperative Transport with a Swarm of Miniature Mobile Robots”. In: *IEEE Transactions on Robotics* 31.2 (2015), pp. 307–321. doi: 10.1109/TRO.2015.2400731.

[48] Yili Fu, Han Li, and Yulin Ma. “Path Planning of Cooperative Robotics and Robot Team”. In: *2006 IEEE International Conference on Robotics and Biomimetics*. 2006, pp. 1250–1255. doi: 10.1109/ROBIO.2006.340107.

[49] Jyun-Yu Jhang, Cheng-Jian Lin, and Kuu-Young Young. “Cooperative Carrying Control for Multi-Evolutionary Mobile Robots in Unknown Environments”. In: *Electronics*

8.3 (2019). issn: 2079-9292. doi: 10.3390/electronics8030298. url: https://www.mdpi.com/2079-9292/8/3/298.

[50] Ying Wang and C.W. de Silva. “Cooperative Transportation by Multiple Robots with Machine Learning”. In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 3050–3056. doi: 10.1109/CEC.2006.1688694.

[51] Hao Zhang et al. “H2GNN: Hierarchical-Hops Graph Neural Networks for Multi-Robot Exploration in Unknown Environments”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3435–3442. doi: 10.1109/LRA.2022.3146912.

[52] Tohru Kawabe and Kou Nakamura. “Cooperative Carrying Task Control based on RHC for Mobile Robots”. In: *International Journal of Circuits, Systems and Signal Processing*. Vol. 4. 3. 2010, pp. 129–136.

[53] Devika Mohan and A. Vivek. “Navigation of two wheeled mobile robots cooperatively carrying an object”. In: *2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*. 2017, pp. 1–7. doi: 10.1109/ICCPCT.2017.8074218.

[54] Alpaslan Yufka, Osman Parlaktuna, and Metin Ozkan. “Formation-based cooperative transportation by a group of non-holonomic mobile robots”. In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. 2010, pp. 3300–3307. doi: 10.1109/ICSMC.2010.5642400.

[55] Mahitthidetch Udomkun and Poj Tangamchit. “Cooperative behavior-based control of decentralized mobile robots on an overhead box carrying task”. In: *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. Vol. 2. 2008, pp. 633–636. doi: 10.1109/ECTICON.2008.4600513.

[56] S. Furuno, M. Yamamoto, and A. Mohri. “Trajectory planning of cooperative multiple mobile manipulators”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 1. 2003, 136–141 vol.1. doi: 10.1109/IROS.2003.1250618.

[57] Linh Kästner et al. “Arena-Bench: A Benchmarking Suite for Obstacle Avoidance Approaches in Highly Dynamic Environments”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 9477–9484. doi: 10.1109/LRA.2022.3190086.

[58] Eric Heiden et al. “Bench-MR: A Motion Planning Benchmark for Wheeled Mobile Robots”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4536–4543. doi: 10.1109/LRA.2021.3068913.

[59] Aviv Tamar, Sergey Levine, and Pieter Abbeel. “Value Iteration Networks”. In: *CoRR* abs/1602.02867 (2016). arXiv: 1602.02867. url: http://arxiv.org/abs/1602.02867.

[60] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[61] Xueguang Lyu et al. “Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning”. In: *CoRR* abs/2102.04402 (2021). arXiv: 2102.04402. url: https://arxiv.org/abs/2102.04402.

[62] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6382–6393. isbn: 9781510860964.

[63] Sanmit Narvekar et al. “Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey”. In: *CoRR* abs/2003.04960 (2020). arXiv: 2003.04960. url: https://arxiv.org/abs/2003.04960.

[64] Gyuho Eoh and Tae-Hyoung Park. “Cooperative Object Transportation Using Curriculum-Based Deep Reinforcement Learning”. In: *Sensors* 21.14 (2021). issn: 1424-8220. doi: 10.3390/s21144780. url: https://www.mdpi.com/1424-8220/21/14/4780.

[65] Gyuho Eoh and Tae-Hyoung Park. “Automatic Curriculum Design for Object Transportation Based on Deep Reinforcement Learning”. In: *IEEE Access* 9 (2021), pp. 137281–137294. doi: 10.1109/ACCESS.2021.3118109.

[66] Danny Tseng. *Riding the Wave of 5G, a Millimeter at a Time*. 2021. url: https://developer.qualcomm.com/blog/riding-wave-5g-millimeter-time.

[67] ETSI 3GPP. *5G; Study on channel model for frequencies from 0.5 to 100 GHz (3GPP TR 38.901 version 17.0.0 Release 17)*. 2022. url: https://www.etsi.org/.

[68] Marco Mezzavilla et al. “End-to-End Simulation of 5G mmWave Networks”. In: *IEEE Communications Surveys and Tutorials* 20.3 (2018), pp. 2237–2263. doi: 10.1109/COMST.2018.2828880.

[69] Greg Brockman et al. “OpenAI Gym”. In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. url: http://arxiv.org/abs/1606.01540.

[70] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. url: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[71] Sainbayar Sukhbaatar et al. “MazeBase: A Sandbox for Learning from Games”. In: *CoRR* abs/1511.07401 (2015). arXiv: 1511.07401. url: http://arxiv.org/abs/1511.07401.

[72] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for Gymnasium*. 2018. url: https://github.com/Farama-Foundation/Minigrid.

[73] Eric Liang et al. “Ray RLLib: A Composable and Scalable Reinforcement Learning Library”. In: *CoRR* abs/1712.09381 (2017). arXiv: 1712.09381. url: http://arxiv.org/abs/1712.09381.

[74] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. url: http://jmlr.org/papers/v22/20-1364.html.

[75] David A. Lizárraga. “Obstructions to the Existence of Universal Stabilizers for Smooth Control Systems”. In: *Mathematics of Control, Signals and Systems* 16.4 (Mar. 2004), pp. 255–277. issn: 1435-568X. doi: 10.1007/s00498-003-0140-x. url: https://doi.org/10.1007/s00498-003-0140-x.

[76] Steven Macenski et al. “The Marathon 2: A Navigation System”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[77] Salih Marangoz. *image2gazebo*. https://github.com/salihmarangoz/image2gazebo. 2020.

[78] Wallace dos Reis et al. “An extended analysis on tuning the parameters of Adaptive Monte Carlo Localization ROS package in an automated guided vehicle”. In: *The International Journal of Advanced Manufacturing Technology* 117 (Nov. 2021), pp. 1–21. doi: 10.1007/s00170-021-07437-0.

[79] Vivienne Sze et al. “Efficient Processing of Deep Neural Networks: A Tutorial and Survey”. In: *CoRR* abs/1703.09039 (2017). arXiv: 1703.09039. url: http://arxiv.org/abs/1703.09039.

[80] Semuil Tjiharjadi and Erwin Setiawan. “Design and Implementation of a Path Finding Robot Using Flood Fill Algorithm”. In: *International Journal of Mechanical Engineering and Robotics Research.* 5 (Jan. 2016). doi: 10.18178/ijmerr.5.3.180-185.

A Actor-Critic Additional Material

A.1 Derivation of the REINFORCE Gradient Estimator

Recall the RL objective function:

$$E^{RL}(\theta) = \int P(S_0) V_{\pi_\theta}(S_0) dS_0 = \mathbb{E}_{\pi_\theta}[G_0] \tag{A.1}$$

Let's unroll the expectation:

$$\mathbb{E}_{\pi_\theta}[G_0] = \sum_{S_0,\dots,S_T} \sum_{A_0,\dots,A_{T-1}} \sum_{R_1,\dots,R_T} \prod_{t=0}^{T-1} P(S_{t+1}, R_{t+1}|S_t, A_t) \prod_{t=0}^{T-1} \pi_\theta(A_t|S_t) \sum_{k=0}^{T-1} \gamma^k R_{k+1} \tag{A.2}$$

let's define:

$$P_{0:\tau} = \prod_{k=0}^{\tau} P(S_{k+1} R_{k+1}|S_k, A_k) \tag{A.3}$$

$$Q^\theta_{0:\tau} = \prod_{k=0}^{\tau} \pi_\theta(A_k|S_k) \tag{A.4}$$

$$\sum_{S,A,R} = \sum_{S_0,\dots,S_T} \sum_{A_0,\dots,A_{T-1}} \sum_{R_1,\dots,R_T} \tag{A.5}$$

therefore:

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}[G_0] &= \sum_{S,A,R} P_{0:T-1} Q^\theta_{0:T-1} \sum_{k=0}^{T-1} \gamma^k R_{k+1} \\
&= \sum_{k=0}^{T-1} \sum_{S,A,R} P_{0:T-1} Q^\theta_{0:T-1} \gamma^k R_{k+1} \\
&= \text{by summing the probabilities that do not depend on other terms} \\
&= \sum_{k=0}^{T-1} \sum_{S,A,R} P_{0:k} Q^\theta_{0:k} \gamma^k R_{k+1}
\end{aligned} \tag{A.6}$$

By differentiating over the policy parameters θ we obtain:

$$
\begin{aligned}
\frac{\partial E^{RL}}{\partial \theta} &= \sum_{k=0}^{T-1} \sum_{S,A,R} P_{0:k} \gamma^k R_{k+1} \frac{\partial Q^{\theta}_{0:k}}{\partial \theta} \\
&= \text{using } \frac{\partial h(x)}{\partial x} = h(x) \frac{\partial \log(h(x))}{\partial x} \\
&= \sum_{k=0}^{T-1} \sum_{S,A,R} P_{0:k} \gamma^k R_{k+1} \frac{\partial \log Q^{\theta}_{0:k}}{\partial \theta} Q^{\theta}_{0:k} \\
&= \sum_{k=0}^{T-1} \sum_{S,A,R} P_{0:k} Q^{\theta}_{0:k} \gamma^k R_{k+1} \frac{\partial}{\partial \theta} \sum_{\tau=0}^{k} \log \pi_{\theta}(A_{\tau}|S_{\tau}) \\
&= \sum_{S,A,R} \sum_{k=0}^{T-1} \sum_{\tau=0}^{k} P_{0:k} Q^{\theta}_{0:k} \gamma^k R_{k+1} \frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} \\
&= \sum_{S,A,R} \sum_{\tau=0}^{T-1} \sum_{k=\tau}^{T-1} P_{0:k} Q^{\theta}_{0:k} \gamma^k R_{k+1} \frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}
\end{aligned}
\tag{A.7}
$$

which means that:

$$
\begin{aligned}
&= \sum_{\tau=0}^{T-1} \sum_{k=\tau}^{T-1} \gamma^k R_{k+1} \frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} \\
&= \sum_{\tau=0}^{T-1} \gamma^{\tau} \sum_{k=\tau}^{T-1} \gamma^{k-\tau} R_{k+1} \frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} \\
&= \sum_{\tau=0}^{T-1} \gamma^{\tau} G_{\tau} \frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}
\end{aligned}
\tag{A.8}
$$

is an estimate of $\frac{\partial E^{RL}}{\partial \theta}$. Hence the gradient estimator is:

$$
\frac{\partial E^{RL}}{\partial \theta} = \mathbb{E}\left[\sum_{\tau=0}^{T-1} \gamma^{\tau} G_{\tau} \frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} \right] \tag{A.9}
$$

A.2 The Baseline Term Does Not Affect the Gradient Estimates

Note that:

$$\begin{aligned}\frac{\partial E^{RL}}{\partial \theta} &= \mathbb{E}\left[\sum_{\tau=0}^{T-1}\gamma^{\tau}(G_{\tau}-b(S_{\tau}))\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}\right] \\ &= \sum_{\tau=0}^{T-1}\mathbb{E}\left[G_{\tau}\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} - b(S_{\tau})\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}\right] \\ &= \mathbb{E}\left[\sum_{\tau=0}^{T-1}\gamma^{\tau}G_{\tau}\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}\right] - \sum_{\tau=0}^{T-1}\mathbb{E}\left[b(S_{\tau})\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}\right]\end{aligned} \tag{A.10}$$

let's focus on the second term:

$$\begin{aligned}\mathbb{E}\left[b(S_{\tau})\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta}\right] &= \sum_{S_{\tau}}\sum_{A_{\tau}}P(S_{\tau})\pi_{\theta}(A_{\tau}|S_{\tau})b(S_{\tau})\frac{\partial \log \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} \\ &= \sum_{S_{\tau}}\sum_{A_{\tau}}P(S_{\tau})b(S_{\tau})\frac{\partial \pi_{\theta}(A_{\tau}|S_{\tau})}{\partial \theta} \\ &= \sum_{S_{\tau}}P(S_{\tau})b(S_{\tau})\frac{\partial}{\partial \theta}\sum_{A_{\tau}}\pi_{\theta}(A_{\tau}|S_{\tau}) \\ &= \sum_{S_{\tau}}P(S_{\tau})b(S_{\tau})\frac{\partial}{\partial \theta}1 \\ &= 0\end{aligned} \tag{A.11}$$

Therefore the baseline term in expectation does not affect the gradient estimator.

B Listings

B.1 Value Propagation Module

```python
def _v_prop(
    self,
    rin: torch.Tensor,
    rout: torch.Tensor,
    p: torch.Tensor,
) -> torch.Tensor:
    """
    Value propagation algorithm

    :param rin: reward matrix
    :param rout: reward matrix
    :param p: propagation matrix
    :return: values, values matrix
    """
    actions = [(0, 1), (2, 1), (1, 0), (1, 2),
               (0, 0), (0, 2), (2, 0), (2, 2)]
    values = torch.zeros_like(rin)

    padded_rin = torch.nn.functional.pad(
        rin, (1, 1, 1, 1, 0, 0), "constant", 0
    )

    for __ in range(self.vi_k):
        padded_v = torch.nn.functional.pad(
            values, (1, 1, 1, 1, 0, 0), "constant", 0
        )

        for h_offset, w_offset in actions:
            shifted_v = padded_v[
                :,
                h_offset : h_offset + self.maze_size,
                w_offset : w_offset + self.maze_size,
            ]
            shifted_rin = padded_rin[
                :,
                h_offset : h_offset + self.maze_size,
                w_offset : w_offset + self.maze_size,
            ]

            nv = p * shifted_v + shifted_rin - rout
            values = values.maximum(nv)

    return values
```

Listing B.1: PyTorch Implementation of the Value Propagation Module

B.2 Actor Network

```python
def _compute_logits(
    self, obs: torch.Tensor, values: torch.Tensor
) -> torch.Tensor:
    """
    Forward of the policy net (logits)

    :param obs: Observation
    :param values: Values matrix
    :return: logits, output of the latent layer of the policy net
    """
    # Retrieve Batch Size
    B = obs.shape[0]

    # concatenate values to observations
    obs_val = torch.cat((obs, values.unsqueeze(dim=1)), dim=1)

    # Neighbor cut-out
    padding = self.neighbor_size // 2
    padded_obs_val = torch.nn.functional.pad(
        obs_val,
        (padding, padding, padding, padding, 0, 0, 0, 0),
        "constant",
        0,
    )

    pos = torch.nonzero(obs_val[:, 1, :, :])[:, 1:]
    if pos.numel() == 0:
        assert False, "Agent not found"

    selected_obs_val = torch.zeros(
        (B, self.in_channels + 1, self.neighbor_size, self.neighbor_size)
    )
    for b in range(B):
        i, j = pos[b, 0] + padding, pos[b, 1] + padding
        selected_obs_val[b, :, :, :] = padded_obs_val[
            b,
            :,
            i - padding : i + 1 + padding,
            j - padding : j + 1 + padding
        ]

    # Policy network
    logits = self.Logit(torch.flatten(selected_obs_val, start_dim=1))
    logits = self.relu(logits)
    return logits
```

Listing B.2: PyTorch Implementation of the Actor Network

University of Padua

Via VIII Febbraio, 2
35122 Padova, Italy

www.unipd.it

Technical University of Denmark

Anker Engelunds Vej, 1
2800 Kgs. Lyngby, Denmark

www.dtu.dk

Huawei Munich Research Center

Riesstraße 25
80992 München, Germany

www.huawei.com