Università degli Studi di Padova

Department of Information Engineering

Ph.D. Course in Information Engineering

XXXV Series

Communication and learning in dynamic networks

Coordinator
Andrea Neviani
University of Padova

Supervisor
Andrea Zanella
University of Padova

Ph.D. Candidate
Federico Mason

Academic Year 2022/2023

To my cousin Chiara and my brother Francesco

Abstract

Nowadays, telecommunication systems are essential to numerous fields, enabling revolutionary applications such as industry 4.0, the Internet of Things, virtual reality, and cloud computing. On the other hand, most telecommunication networks still present a rigid architecture that does not lend itself to managing highly dynamic scenarios with heterogeneous characteristics. In order to address the requirements of future technologies, it is necessary to design new solutions that enable the continuous configuration of services and functions without external intervention. The scientific community is putting strong effort toward such a vision, investigating strategies to provide mobile infrastructures, enable the softwarization of network resources, and deploy cognitive algorithms for orchestrating system operations.

In this thesis, we address the challenge of defining a dynamic network architecture with all these characteristics and, thus, capable of adapting to various working scenarios. The central point of our idea is to distribute network functions among multiple intelligent units that interact with each other to find the optimal system configuration, maximizing performance while ensuring resilience to local failures. To this end, we organize the thesis into three parts, each investigating a different aspect of dynamic network management. Firstly, we define a tracking framework based on the unscented Kalman filter to map a group of autonomous nodes broadcasting their local status while operating in different application scenarios. Hence, we study how to adapt the communication policy of each node in accordance with the estimate of the overall network status. Then, we exploit the reinforcement learning paradigm and implement a distributed learning architecture to allow the single nodes to cooperate and reach a common goal. In this context, we show that adapting the communication and control policy to each other leads to higher performance than the basic scenario in which communication is pre-determined.

Lastly, we investigate new techniques to dynamically manage communication and computational resources in the target scenario, taking advantage of the network slicing and software-defined networking paradigms. To this goal, we distribute the resource allocation tasks among multiple agents, providing a highly flexible architecture that can operate in different contexts without retraining. Our results show that learning algorithms are fundamental for implementing a fully flexible network architecture, capable of detecting changes in the surrounding environment and adapting its functions consequently. At the same time, artificial intelligence requires network designers to devote part of the available resources to support the training of the learning agents managing the network. This competition between immediate performance and innovation capability determines a significant trade-off, to which we devote the last chapter of our research.

Sommario

Al giorno d'oggi, i sistemi di telecomunicazione sono essenziali in innumerevoli settori e permettono l'utilizzo di tecnologie rivoluzionarie come l'industria 4.0, l'Internet delle cose, la realtà virtuale e il cloud computing. D'altra parte, la maggior parte delle reti di telecomunicazioni presenta ancora un'architettura piuttosto rigida che mal si presta a gestire scenari altamente dinamici con caratteristiche eterogenee. Per far fronte ai requisiti delle tecnologie future, è necessario ideare nuove soluzioni che consentano di riconfigurare servizi e funzioni di rete senza un intervento esterno. La comunità scientifica sta facendo un grande sforzo per realizzare questa visione, studiando strategie per fornire infrastrutture mobili, virtualizzare le risorse disponibili e implementare algoritmi cognitivi che gestiscano autonomamente le funzioni di rete.

In questa tesi affronteremo la sfida di definire un'architettura di rete dinamica, avente tutte le caratteristiche descritte e, pertanto, in grado di adattarsi a diversi scenari di lavoro. Il punto centrale della nostra idea è distribuire le funzioni di rete tra più unità intelligenti che interagiscono tra loro per individuare la configurazione ottima del sistema, massimizzando le prestazioni e garantendo al contempo la resilienza a guasti locali. Per raggiungere questo obiettivo, organizzeremo la tesi in tre parti, ciascuna delle quali indaga un diverso aspetto della gestione dinamica delle reti di telecomunicazioni. Nelle prima parte, definiremo un sistema di tracciamento, basato sull'*unscented Kalman filter*, per mappare un gruppo di nodi autonomi che trasmettono il loro stato locale mentre operano in diversi scenari operativi. In particolare, studieremo come adattare la strategia di comunicazione di ciascun nodo di rete alla stima dello condizioni del sistema nel suo complesso. Dopodiché, sfrutteremo il paradigma di *reinforcement learning* e implementeremo un'architettura di apprendimento distribuito che consenta alle singole unità di rete di cooperare e raggiungere un obiettivo comune. In questo contesto, dimostreremo che l'adattamento reciproco delle strategie di comunicazione e controllo porta a prestazioni superiori rispetto a uno scenario di base in cui la comunicazione è predeterminata.

Nell'ultima parte della tesi, studieremo nuove tecniche per gestire le risorse computazionali e di comunicazione nello scenario descritto, sfruttando i paradigmi di *network slicing* e di *software-defined networking*. A tal fine, distribuiremo la gestione delle risorse di rete tra più unità intelligenti, che andranno a costituire un'unica architettura adattabile, capace di operare in contesti diversi senza essere riconfigurata da zero. I nostri risultati mostrano che gli algoritmi di apprendimento sono fondamentali per ottenere un'architettura di rete completamente flessibile, che possa rilevare i cambiamenti nell'ambiente circostante e modificare le proprie funzioni di conseguenza. Allo stesso tempo, l'utilizzo dell'intelligenza artificiale richiede di dedicare parte delle risorse di rete disponibili per l'allenamento degli agenti di apprendimento che gestiscono la rete stessa. Questa competizione tra prestazioni immediate e capacità di innovazione di sistema determina un significativo *trade-off*, a cui dedicheremo l'ultimo capitolo di questa tesi.

Contents

Listing of figures

Listing of tables

Acronyms

Symbols

3D-CTRA 3-Dimensional CTRA. 36, 37, 39, 42, 43, 47, 49, 50, 51, 52, 53, 54, 55, 56, 163
3GPP 3rd Generation Partnership Project. 119
5G 5th Generation. 1, 2, 9, 35, 119, 145
6G 6th Generation. 1, 2, 35

A

A2C Advance Actor Critic. 129, 130, 131, 138
ACK Acknowledgment. 45, 48, 54
Adam Adaptive moment estimation. 108, 138, 154
ADR Adaptive Data Rate. 37, 38, 45, 48, 53, 54, 55, 56
AoI Age of Information. 10, 11, 46, 51, 92, 93, 98, 99
AUV Autonomous Underwater Vehicle. 92, 93, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114

B

BF Bayesian Filtering. 36, 37
BS Base Station. 146, 149, 150, 152, 153, 154

C

CC Closest Communication. 107, 109, 110, 111, 112, 113, 114
CDF Cumulative Distribution Function. 80
C-ITS Connected and Intelligent Transportation Systems. 11, 13, 17
CJCC Centralized JCC. 108, 109, 110, 111, 112, 113, 114
CL Continual Learning. 2, 145, 161
CN Core Network. 146
CNN Convolutional Neural Network. 74, 106
CP-POMDP Cyber-Physical POMDP. 92, 93, 95, 96, 97, 107, 114
CPS Cyber-Physical System. 92
CS Constant Speed. 36, 39, 40, 47, 49, 50, 51, 53, 56
CSCC Channel Sensing Congestion Control. 19, 20, 25, 26, 27, 28, 31, 32
CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. 12, 16, 20, 21, 30
CTRA Constant Turn Rate and Acceleration. 11, 12, 16, 36, 37, 39, 40, 41, 43, 47, 49, 50, 51, 53, 56, 163
CVI Critical Video. 153
CVO Critical Voice. 153

D

DDQL Distributed Deep Q-Learning. 72, 75, 76, 77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88

Dec-POMDP Decentralized Partially Observable Markov Decision Process. 65, 70, 95
DJCC Distributed JCC. 108, 110, 111, 112, 113, 114
DL Downlink. 48
DQN Deep Q-Network. 73, 74, 91, 152, 153
DRL Deep Reinforcement Learning. 3, 64, 91, 105, 120, 121, 122, 128, 129, 133, 134, 135, 136, 139, 140, 141, 142, 143, 144, 145, 146, 149, 151, 152, 157, 169
DSRC Dedicated Short Range Communication. 12, 16

E

ED End Device. 45, 48, 53
eMBB enhanced Mobile BroadBand. 119, 120, 124, 125, 135, 136, 137, 139, 140, 143, 144
ETB Error Threshold Broadcasting. 17, 19, 20, 24, 25, 26, 30, 31, 32, 33

F

FEC Forward Error Correction. 94
FL Federated Learning. 34, 146
FoV Field of View. 67, 68, 70

G

GA Genetic Algorithm. 135
GPS Global Positioning System. 9, 11, 35, 39, 46, 47
GW Gateway. 45, 47, 48, 49, 50, 54

I

IoT Internet of Things. 38, 46, 66, 91, 93, 94

J

JCC Joint Communication and Control. 106, 107, 108, 111, 112, 113, 114

K

KF Kalman Filter. 11, 36, 37, 163

L

LA Look-Ahead. 64, 75, 76, 77, 78, 79, 81, 83, 84, 85, 86, 87, 88, 89
LoRaWAN Long Range Wide Area Network. 36, 37, 38, 44, 45, 46, 48, 53, 56, 57

M

MAC Medium Access Control. 12, 16, 20, 45
MARL Multi-Agent Reinforcement Learning. 64, 65, 66, 89, 92, 94, 95, 106, 107, 160
MC Markov Chain. 21
MDP Markov Decision Process. 64, 70, 92, 96, 148, 151, 167
MEC Mobile Edge Computing. 145, 148, 157
ML Machine Learning. 1, 11, 12, 34, 120, 121, 133, 139, 141, 160, 161, 167

mMTC massive Machine Type Communication. 119

N

NACC Neighbor Aware Congestion Control. 19, 20, 24, 25, 26, 28, 31, 32, 33
NC Networked Control. 106, 107
NCVI Non-Critical Video. 153
NCVO Non-Critical Voice. 153
ND-POMDP Networked Distributed Partially Observable Markov Decision Process. 64, 67, 70, 88, 92
NFV Network Function Virtualization. 119, 138, 143
NN Neural Network. 64, 65, 72, 105, 106, 107, 108, 109, 120, 121, 122, 131, 138, 152, 154, 169
NS Network Slicing. 2, 119, 120, 121, 122, 129, 144, 146, 149, 152, 160
NSE Network Server. 45, 48

O

OC Oracle Communication. 107, 110, 111, 112, 113, 114

P

PAoI Peak AoI. 51, 52, 56
PB Periodic Broadcasting. 17, 19, 20, 24, 25, 26, 30, 31, 32, 33
PF Particle Filter. 11, 37
POMDP Partially Observable Markov Decision Process. 72

Q

QoE Quality of Experience. 66
QoI Quality of Information. 10
QoS Quality of Service. 2, 3, 91, 92, 94, 121, 145, 149, 160

R

RAdam Rectified Adam. 73
RC Random Communication. 107, 110
ReLU Rectifier Linear Unit. 106, 131
RL Reinforcement Learning. 64, 65, 66, 70, 71, 98, 103, 105, 120, 129, 130, 132, 145, 146, 147, 167, 168, 169
RSSI Received Signal Strength Indicator. 38

S

SARSA State-Action-Reward-State-Action. 154, 168
SDN Software Defined Networking. 119, 121, 138, 143
SF Spreading Factor. 37, 45, 46, 48, 49, 50, 53, 54, 56
SLA Service Level Agreement. 120, 121, 122, 123, 124, 125
SNR Signal to Noise Ratio. 13, 38
SUMO Simulation of Urban MObility. 10, 26, 27, 29

T

TL Transfer Learning. 64, 65, 66, 75, 79, 82, 120, 121, 122, 131, 136, 140, 141, 142, 144, 147, 161

U

UAV Unmanned Aerial Vehicle. 7, 35, 36, 37, 38, 39, 40, 44, 45, 46, 47, 48, 49, 51, 52, 53, 56, 61, 63, 64, 65, 66, 67, 68, 70, 71, 73, 75, 76, 77, 78, 79, 82, 83, 84, 87, 88, 89, 159, 160

UKF Unscented Kalman Filter. 11, 16, 18, 19, 25, 39, 44, 47, 48, 52, 163, 164

UL Uplink. 45, 48, 54

UoI Urgency of Information. 94

URLLC Ultra Reliable Low Latency Communication. 9, 119, 120, 124, 125, 135, 137, 139, 140, 141, 142, 143, 144

UT Unscented Transformation. 164

V

VoI Value of Information. 93, 94

1 Introduction

SINCE the end of the last century, humanity has seen a phase of technological progress without precedent. The significant improvement in computational technologies has enabled more powerful Machine Learning (ML) algorithms, which surpassed human abilities in multiple areas. The diffusion of virtual and augmented reality has given birth to new applications for supporting user entertainment and highly specialized tasks such as remote surgery and emergency missions. The Internet of Things has fostered a new world vision, where a unique network virtually interconnects any devices, from house appliances to cars. The rapid emergence of these and other technologies is at the basis of the fourth industrial revolution, allowing new business models where everything is tailored to the specific requirements of the end users. Product customization is a paramount aspect of Industry 4.0 since it makes it possible to maximize users' experience while reducing resource waste. To fully attain such a vision, companies are looking to innovate production processes, making them more efficient and adaptable to customer requests. With this business shift in mind, the scientific community is moving away from one-time problems, developing engineering solutions to address mutable requirements and conditions.

Telecommunication technologies are a fundamental enabler of the industrial revolution that has characterized the last few years. Without a proper telecommunication infrastructure, it will be impossible to design cloud systems and gather the massive amounts of data necessary for ML models' training. Telecommunications are at the basis of the Internet of Things and play a primary role in the development of virtual reality, blockchain, remote robot control, and many other breakthrough applications. However, until a few years ago, most telecommunication networks presented a monolithic architecture that hardly suited these new services. This limitation was overcome by the 5^{th} Generation (5G) and 6^{th} Generation (6G) technologies, which shifted the design focus from merely communication performance to the system's flexibility and adaptability. To this goal, many instruments have been proposed, including mobile communication infrastructure, the virtualization of network resources, and the use of cognitive algorithms.

Upcoming telecommunication networks will not constitute a rigid infrastructure to which application requirements must adapt but will be designed as a service that users can require on-demand according to their current needs. The fulfillment of this vision requires new approaches to virtualize network functionalities and manage the different network components. In this regard, the 5G paradigm considers aerial base stations fundamental for providing communication coverage in remote areas where physical facilities are absent. Besides, the use of drones in critical scenarios makes it possible to define breakthrough applications and improve those already in place. Aerial networks can help emergency operators monitor their intervention during wildfires or earthquakes, and similar systems offer higher data rates in areas with massive user density, such as a stadium. Parallel to the diffusion of drone services, 5G technologies lead autonomous driving toward its highest performance and security levels. Hence, upcoming telecommunication systems will enable inter-vehicle communication to facilitate traffic maneuvers and reduce the probability of car collisions.

Another requirement of future networks is to allow more flexible management of the inner resources and protocols. The increasing heterogeneity of future technologies requires telecommunication systems to provide an ever-increasing number of Quality of Service (QoS) levels to accomplish the specific requirements of each application. The 5G and 6G paradigms made this possible through full virtualization of the network functionalities, including the allocation of communication bandwidth and computational power. This concept has led to the Network Slicing (NS) paradigm, a novel approach that envisages splitting the virtual network resources among different logical networks, named slices. Network tenants can customize each slice to offer specific services and initialize new ones on-demand when different application classes enter the system. Besides dramatically improving user experience, the NS paradigm ensures that the different slices are isolated, which means that the performance degradation in one application class does not affect the others.

Lastly, future telecommunication systems will minimize interventions for network management and reconfiguration. In the past, a minimal change in the network dynamics required tenants to take direct actions to modify the existing infrastructures and eventually add new elements. Thanks to the progress of ML technologies, networks can autonomously reorganize their functions, e.g., actively displacing base stations and changing resource allocation among different slices. Besides reducing the capital expenditure necessary to maintain the network, ML makes it possible to reach performance and efficiency levels beyond human possibility. In particular, the Continual Learning (CL) paradigm allows networks to adapt to new working conditions continuously, even in the case of scenarios that deviate from those seen during the training. On top of that, the adoption of distributed learning approaches further increases the flexibility of the network architecture, enabling a constant integration of new features and services and providing a new level of robustness. Indeed, splitting the control duties among multiple learning units allows the final system to operate even in case of network failures, making the users never experience degraded performance.

The scientific community is putting strong effort toward developing new strategies for managing future networks, considering the necessity of constantly reconfiguring protocols and resources

to address requirements that change over time. In this dissertation, we try to design a series of solutions to attain such a vision, with the final aim of building a dynamic network architecture capable of adapting its physical topology and virtual resources to different working scenarios. The salient point of our research is to distribute network control, associating each task with an intelligent unit that interacts with many others to find the optimal system configuration. At the same time, our distributed learning approaches involve new and fascinating challenges for network design and orchestration.

We divide this dissertation into three parts, each tackling a specific aspect of dynamic networks' management. In the first part, we investigate new communication strategies to track the nodes of mobile networks, considering both terrestrial (Chapter I) and aerial devices (Chapter II). The goal is to design a flexible framework to map the overall network without the support of a centralized architecture but offloading the tracking tasks to all the network nodes. In the proposed approach, each node acts as an independent unit and tunes its communication policy according to the local estimate of the network conditions. Going beyond the current literature, we base the broadcasting of local information not on traditional metrics, such as the Age of Information (AoI), but on the *urgency* of the messages to be transmitted. With this concept, we intend the potential benefits that the message transmissions bring to the final task achieved by the system without considering traditional communication metrics. In addition, we adapt the specific communication strategies of the local devices to channel conditions, showing that usage of more advanced and computationally expensive tracking systems may lead to performance degradation.

After designing a framework to track node trajectories, the second part of this thesis focuses on how to determine the same trajectories to achieve a specific mission. More specifically, we investigate new Deep Reinforcement Learning (DRL) strategies to coordinate mobile nodes, allowing them to change positions for different purposes and applications, such as mapping and monitoring remote areas. Again, we consider a distributed system, where each unit takes decisions autonomously, allowing the overall network to work even in case of single component failures. The drawback of this approach is that the system performance strongly depends on the nodes' estimates of the overall network status. Indeed, without a dedicated channel to exchange local information, each node can only have a partial view of the network and learn a sub-optimal control policy. In this regard, we prove that the relationship between communication and control is critical for performance maximization, showing that a framework with learning-based communication overcomes the benchmark scenario where communication is determined a priori.

Finally, the third part of this thesis extends our distributed control architecture toward an intelligent allocation of network resources. Taking advantage of the NS paradigm, we design a hierarchical system where multiple agents cooperate and allocate resources among different slices, ensuring that each application class complies with the desired QoS. Our approach can adapt to variable network conditions, becoming fundamental for initializing new slices on demand and prioritizing critical applications in case of limited network resources. A paramount aspect of this scenario is the necessity of allocating part of the resources to support the same learning

algorithms that must orchestrate the network. This last problem is not trivial since increasing the training resource speeds up convergence but reduces the number of network resources devoted to the system users. We model the above problem as a trade-off between learning efficiency and resource availability, showing that the optimal solution varies according to the specific scenario we consider. This model opens new fascinating challenges related to network design since the presence of learning agents will be paramount in future telecommunication and, in general, engineering systems.

Part I

Quality-Based Communication Strategies for Node Tracking

Introduction

Over the last few years, the scientific and industry communities have paid great attention to vehicular communication and Unmanned Aerial Vehicles (UAVs) technologies. These systems expect to bring impressive benefits in several fields, from autonomous cars to area surveillance and disaster management. At the same time, vehicular and drone networks may be ineffective without the support of an accurate tracking framework. In this context, the state-of-the-art involves the periodic transmission of the node state updates, thus minimizing the Age of Information (AoI) within the overall system. However, the trajectories of mobile nodes, such as cars and drones, can be erratic and unpredictable, degrading the performance obtained with such an approach. To overcomes such limits, the first part of the thesis defines a new framework to map the evolution of dynamic networks, considering vehicles driving on a city map and a UAV swarm flying in a 3-dimensional environment. In our approach, each node acts independently, ensuring robustness to network failures at the cost of more complex node coordination.

At first, we consider a vehicular network scenario and define a novel communication strategy based on the urgency of local information. Hence, each vehicle does not start new transmission relying on traditional AoI-based metrics but increases or decreases the data rate according to the estimated tracking error. We test the designed system in a simulative scenario based on an actual city map, computing the system performance as the vehicle's capability of predicting the other nodes' trajectories. We further improve the benefits of our strategy by implementing an innovative congestion control mechanism that tunes communication according to the vehicular density in the target area. Our simulations show that the proposed scheme reduces the transmission of redundant messages and improves the overall tracking accuracy by more 20% than conventional approaches.

We implement the same framework in a drone swarm scenario, considering the Long Range Wide Area Network (LoRaWAN) standard to collect the measurements of single drones. To correctly represent Unmanned Aerial Vehicle (UAV) trajectories in a 3-dimensional environment, we design a new motion model that extends the Constant Turn Rate and Acceleration (CTRA) equations used in the vehicular scenario. Hence, we investigate the trade-off of the LoRa technology, showing how to tune the Spreading Factor (SF) and the tracking settings to ensure optimal performance. Our results show that the tracking configuration must adapt to the channel conditions to avoid increasing communication overhead and degrading the overall performance. By carrying our realistic simulations on a public dataset, we prove that our system enables the accurate tracking of dozens of drones at distances up to several kilometers.

2

Broadcasting Strategies for Dynamic Mapping in Vehicular Networks

2.1 Introduction

In recent years, there has been a growing interest in vehicular communications, which have rapidly emerged as a means to support efficient transportation systems [2]. From a safety perspective, vehicular networks can mitigate the severity of traffic accidents by notifying vehicles about dangerous situations, including bad road conditions and approaching emergency vehicles [3]. Besides, they can support various entertainment services, ranging from real-time multimedia streaming to interactive gaming and web browsing [4]. Finally, vehicular communication is the basis of all the innovative applications that enable the function of self-driving cars.

Autonomous car systems present critical latency (less than 10 ms for safety-related services) and reliability (i.e., up to 99.999% for a high degree of automation [5]) requirements, which can be addressed only by the Ultra Reliable Low Latency Communication (URLLC) services of the 5^{th} Generation (5G) paradigm [6]. Besides proving extremely low communication delay, the 5G architecture should guarantee timely vehicle positioning with errors below 1.5 m for where-in-lane positioning, as recommended by the National Highway Traffic Safety Administration (NHTSA) [7]. In a standard scenario, vehicles can determine their positions by onboard Global Positioning System (GPS) receivers, though this system may not always provide the required accuracy [8]. Data fusion techniques [9] can overcome GPS limitations by combining several sources of information into a single solution that is more precise than any individual approach. Besides, inter-vehicular communication can further improve the estimation of the overall net-

The work presented in this chapter was published in *IEEE Transaction on Wireless Communication* [1].

work, allowing nodes to exchange the onboard sensors' measurements and gain cooperative perception of the surrounding environment.

Most advanced vehicular applications require disseminating position updates as timely as possible, i.e., as the onboard sensors generate new measures. The traditional approach is to have each node broadcast periodic updates with its positioning and future trajectory information, monitoring the Age of Information (AoI) and starting new communications consequently. However, vehicular scenarios involve rapid dynamics and unpredictable changes [10], which makes the network topology intrinsically variable and periodic broadcasting strategies inefficient. Long inter-transmission intervals may prevent information dissemination in safety-critical situations, while frequent broadcasting may overload the wireless medium with useless data and increase the number of packet collisions [11]. In this context, it is possible to implement congestion avoidance mechanisms to regulate information distribution as a function of the network load [12]. These techniques dynamically adapt to the number of neighboring vehicles [13] or assign priorities to vehicles based on their operating conditions [14], but usually disregard the final level of positioning accuracy.

Deciding how to regulate the broadcasting of positioning data in a vehicular network is a critical problem that cannot be handled with classical AoI-based solutions. In recent years, the scientific community has investigated more sophisticated mechanisms that explicitly consider the Quality of Information (QoI) [15] to trigger data transmissions. These strategies, however, have been typically proposed for sensor networks and may not be directly applied to the vehicular environment. Some other proposals, e.g., [16, 17, 18], estimate the value of the position information updates and broadcast those that maximize the utility for the target applications. On the other hand, these works do not implement congestion control, and practical validation in real vehicular deployments is still missing.

This chapter addresses the problem of ensuring accurate position estimation in a vehicular network scenario while minimizing the number of communication involved. To this end, we design a broadcasting algorithm that allows every vehicle to estimate the trajectories of all the nodes within its range: the proposed technique activates new communication only if the estimated positioning error is above a predefined threshold. Our approach is computationally efficient and can be executed in real-time, even with the limited onboard computational resources of mid-range and budget car models. Besides, we find a mathematical expression of the packet collision probability as a function of the vehicular traffic density and design a new congestion control mechanism exploiting topology information to reduce the packet collision probability. Compared with traditional channel-based congestion control, our solution can better adapt to fluctuating conditions of the environment, as is typically the case in the vehicular ecosystem.

We investigate the performance of the proposed scheme compared to a state-of-the-art solution that instructs vehicles to broadcast state information at regular intervals. The performance analysis takes place in a set of scenarios generated by Simulation of Urban MObility (SUMO) [19], an open simulator designed to model the traffic of large road networks. The simulation results show that the proposed algorithm can reduce the average position estimation error by more than 10% and its 95th percentile by more than 20% with repsect to a standard

approach minimizing the AoI in the system. Finally, we provide guidelines on the optimal broadcasting strategy for a given set of automotive-specific parameters, including the transmission periodicity and the bandwidth capacity.

We organize the remainder of this chapter as follows: Sec. 2.2 presents a selection of the most relevant related work; Sec. 2.3 introduces our system model; Secs. 2.4 and 2.5 describe our broadcasting and congestion control strategies, and derive the expression for the packet collision probability as a function of the vehicular traffic density; Sec. 2.6 validates our theoretical analysis through simulations; finally, Sec. 2.7 provides the chapter conclusions and suggestions for future work.

2.2 Related Work

In a Connected and Intelligent Transportation Systems (C-ITS) scenario, vehicles are equipped with onboard sensors and use them to gather observations about the surrounding environment. These observations are then broadcast within the network through wireless technologies and are used to implement a tracking system [20, 21] whose target state is the set of positions of all surrounding neighbors. The tracking performance depends on the coordination among the vehicles and how the data is processed. The most common choice is to adopt a Bayesian approach, typically based on the Kalman Filter (KF) [22], the Unscented Kalman Filter (UKF) [23], or the Particle Filter (PF) [24]. For instance, the work of [25] designs a tracking framework based on the UKF and the Constant Turn Rate and Acceleration (CTRA) motion model. Instead, in [26], a particle filter jointly processes route information and digital map data, while the authors of [27] achieve position forecasting by a Hidden Markov Model [28] and a Viterbi approach [29]. We highlight that, in all the Bayesian filters, the performance greatly depends on the algorithm settings, e.g., the process and estimation noise covariances, which must be known *a priori* [30].

While conventional tracking approaches focus on the real-time estimation of the target state, most advanced C-ITS applications require a prediction of vehicles' future trajectories. In a basic scenario, we can achieve long-term forecasting by applying the predictive step of a KF to the last available state estimate. However, this solution is very sensitive to imperfections of the motion model, and more sophisticated approaches were proposed in the literature. The authors of [31] exploit the output of a KF to perform a parametric interpolation of the future path of the target vehicle. Instead, in [32], it is designed a framework based on dead reckoning to improve packet forwarding in a highway scenario. Another approach requires describing trajectory prediction as a time series forecasting problem [9] and then applying Machine Learning (ML) techniques to improve target state estimation. In this regard, the authors of [33] use Support Vector Machines to forecast vehicle motion, enabling the target estimation when there is no GPS availability. Instead, the work of [34] introduces a neural network system trained with historical traffic data and then used to predict vehicles' speeds.

In general, ML approaches guarantee high performance but require a massive amount of data for the initial training, which is often not publicly available. Moreover, ML solutions suffer

from significant computational complexity, and vehicles may not support their implementation. Another possibility is to combine ML with Bayesian algorithms: in [35], the authors present a system using a Hidden Markov Model to estimate vehicle maneuvers and a Support Vector Machine to predict future vehicle trajectories. The work of [36] exploits a Radial Basis Function classifier to compute the inner parameters of a particle filter, which then estimates the long-term trajectory of the target. Besides, in [37], the results of a maneuver recognition system are combined with the output of a Bayesian algorithm using the CTRA motion model.

Regardless of the complexity of the tracking framework, the overall performance may be low if onboard sensor measurements are not sufficiently accurate. Therefore, vehicles share local information to take advantage of cooperative perception and compensate for the low quality of the sensor data. To this end, vehicles exploit the Dedicated Short Range Communication (DSRC) technologies and the Wireless Access in Vehicular Environment (WAVE) standard [38, 39], which is based on a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme. Such a system may suffer channel congestion in scenarios with high vehicular density, leading to the risk of losing the transmitted information because of packet collisions. Defining novel congestion control schemes, which suit the characteristics of modern vehicular networks, is thus a problem of high interest.

Over the years, many researchers have proposed different Medium Access Control (MAC) strategies that adapt inter-vehicle communications to channel conditions. In [40], the authors present a rate-adaption strategy that ensures channel stability also in vehicular scenarios, proving the convergence of the proposed algorithm and providing guidance for the choice of the algorithm parameters. In [41], the hidden terminal problem is avoided by a time-slotted system where each vehicle is assigned a dedicated slot per frame, during which it can alert its neighbors about its future transmissions. Finally, the work of [42] focuses on improving congestion control in road intersections by a distributed ML strategy. In particular, dedicated road infrastructure has the task of deleting redundant communications and assigning specific CSMA/CA parameter settings to different clusters of transmitters.

A solution for reducing channel occupancy is to select the optimal transmission strategy as a function of the instantaneous positioning error of nearby vehicles. For instance, the authors in [43] propose a broadcasting strategy in which vehicles trigger new transmissions whenever the estimates of their neighbors' errors are above a predetermined threshold. However, such analysis is provided only for specific case scenarios, and the framework that predicts future vehicles' states is quite obsolete with respect to current vehicular tracking techniques. Similarly, in [44], the vehicles' transmission rate is tuned according to both the positioning error and the estimated number of packet collisions. Nevertheless, this strategy assumes each vehicle is aware of the number of packet losses at each timeslot. Such a condition does not hold in vehicular scenarios, where most collisions are due to the hidden terminal problem and cannot be sensed directly. Instead, the authors of [45] analyze the communication dynamics causing the hidden terminal problem and address the limits of CSMA/CA by varying the vehicle communication range according to channel occupancy. However, the validity of the approach is proven only in a highway scenario and cannot be generalized to more complex and unpredictable environments.

The adaptation of the vehicles' data rate according to the channel conditions, as well as the implementation of accurate tracking frameworks, are fundamental for the development of future vehicular networks. In order to respond to these necessities, this chapter designs a new vehicular communication strategy that minimizes broadcasting operations while ensuring accurate position estimation. To the best of our knowledge, the proposed system is the first to address the problem of balancing the frequency of vehicular communication as a function of the overall tracking performance and the channel conditions.

2.3 System Model

In this section, we present the system model on which we base our study. First, we theoretically model a C-ITS network as a time-varying Euclidean graph, whose nodes and edges represent vehicles and their communication links, respectively. Then, we define a performance metric that considers both the tracking errors and the vehicle positions. Finally, we describe the tracking system implemented by each vehicle and the communication channel through which vehicles disseminate local information respectively. For the reader convenience, we report the main notation of our model in Tab.2.1

Tab. 2.1: Model parameters.

Parameter	Description	Parameter	Description
t	Slot index	τ	Slot duration
$\mathcal{N}$	Vehicle set	$\mathcal{E}$	Edge set
N	Vehicle number	$\Gamma_{n,m}$	SNR between n and m
Γ_{thr}	SNR threshold	d_r	Communication range
$\mathbf{x}_n$	State of n	$\hat{\mathbf{x}}_{n,m}$	Estimate of $\mathbf{x}_m$ made by n
$\mathbf{P}_{n,m}$	Estimation accuracy of $\hat{\mathbf{x}}_{n,m}$	T_{track}	Maximum tracking time
$\mathcal{N}_n$	Neighbors of n	$\hat{\mathcal{N}}_n$	Estimated neighbors of n
$\mathcal{F}_n(\cdot)$	Error function of n	$\mathcal{F}(\cdot)$	System error function
$d(\mathbf{x}_n, \mathbf{x}_m)$	Distance between n and m	$\lambda_{n,m}$	Proximity parameter
C_{tot}	Maximum subcarrier number	C	Subcarrier number
$T_{\text{per},n}$	Inter-transmission period of n	$E_{\text{thr},n}$	Error threshold of n
$T_{\text{last},n}$	Time from last transmission of n	d_0	Safety distance
K_n	Channel busy ratio of n	$K_{\text{avg},n}$	Average channel busy ratio of n
T_{cbr}	Interval during which K_n is computed	T_{avg}	Interval between $K_{\text{avg},n}$ updates
K_{target}	Target channel busy ratio	ρ_n	Transmission probability of n
$N_{\text{ht},n}$	Number of interfering nodes for n	$\Phi(\cdot)$	Shared communication area function
P_{coll}	Collision probability	P_{thr}	Collision probability threshold
$\mathcal{M}(\cdot)$	Map from E_{thr} to T_{per}	T_{err}	Slots during which the error exceeds E_{thr}

2.3.1 General Model

We represent a C-ITS network as a Euclidean graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, i.e., an undirected graph whose vertices are points on a Euclidean plane [9]. Hence, $\mathcal{N}$ represents the set of nodes, i.e., vehicles, while $\mathcal{E}$ represents the set of edges. We denote by $|\cdot|$ the cardinality operator so that $N = |\mathcal{N}|$ represents the total number of nodes in the system. We say that two vehicles $n, m \in \mathcal{N}$, $n \neq m$, are connected by the edge $< n, m > \in \mathcal{E}$ if the average Signal to Noise Ratio $(\text{SNR})_{n,m}$ between them is higher than a threshold Γ_{thr} that makes correct packet reception possible, i.e., $\mathcal{E} = \{< n, m >: n \neq m, \Gamma_{n,m} > \Gamma_{\text{thr}}\}$. This implies that $< n, m > \in \mathcal{E}$ if and

only if the Euclidean distance between n and m is lower than the communication range d_r. Since the composition of the edge set depends on the node positions, the network topology is time-varying, e.g., new edges can be activated or disabled according to how vehicles move.

In our model, we assume that vehicles move in a two-dimensional space; while not always realistic, this hypothesis does not compromise the accuracy of our analysis. For simplicity, we divide time into discrete timeslots of duration τ, making the system evolve in slots $t \in \mathbb{Z}^+$. To highlight the time dependency of the network, we denote by $\mathcal{G}(t) = (\mathcal{N}(t), \mathcal{E}(t))$ the network graph at slot t. Hence, we define the neighbor set $\mathcal{N}_n(t)$ of $n \in \mathcal{N}(t)$ at slot t as the set of all the vehicles connected to n by an edge in $\mathcal{E}(t)$, i.e., $\mathcal{N}_n(t) = \{m \in \mathcal{N}(t) :< n, m >\in \mathcal{E}(t)\}$.

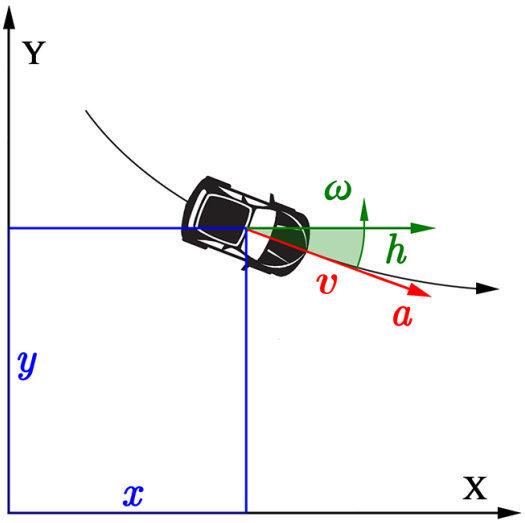


Fig. 2.1: Graphical representation of the vehicle state $\mathbf{x}(t) = \Big(x(t),\, y(t),\, h(t),\, v(t),\, a(t),\, \omega(t)\Big)$ at slot t.

The behavior of each vehicle $n \in \mathcal{N}(t)$ is represented by a 6-tuple $\mathbf{x}_n(t) = \Big(x_n(t),\, y_n(t),\, h_n(t),\, v_n(t),\, a_n(t),\, \omega_n(t)\Big)$, which we call *vehicle state*. In particular, x_n and y_n are the Cartesian coordinates of n on the road topology, h_n is the vehicle's heading direction, v_n and a_n are the vehicle's tangent velocity and acceleration, respectively, and ω_n is the vehicle's angular velocity as exemplified in Fig. 2.1. Consequently, the physical distance between the positions of vehicles n and m at slot t is given by $d(\mathbf{x}_n(t), \mathbf{x}_m(t)) = \sqrt{(x_n(t) - x_m(t))^2 + (y_n(t) - y_m(t))^2}$.

2.3.2 Error Function

In our model each vehicle $n \in \mathcal{N}(t)$ aims at tracking the position of each neighbor $m \in \mathcal{N}_n(t)$ at any time t. For the rest of the work, we call *ego vehicle* any vehicle $n \in \mathcal{N}(t)$ that is tracking a group of neighbors, which are named *target vehicles*. The details of the tracking framework of the ego vehicle are described in Sec. 2.3.3. Hence, we denote by $\hat{\mathcal{N}}_n(t)$ the subset of $\mathcal{N}(t)$ containing the target vehicles, by $\hat{\mathbf{x}}_{n,m}(t)$ the state estimate of $m \in \hat{\mathcal{N}}_n(t)$ maintained by n, and by $\hat{\mathbf{x}}_{n,n}(t)$ the state estimate of n performed by the ego vehicle itself. We highlight that the

real neighbor set $\mathcal{N}_n(t)$ might differ from $\hat{\mathcal{N}}_n(t)$: the ego vehicle could have no knowledge of a vehicle in $\mathcal{N}_n(t)$ (*undetection*), some vehicles in $\hat{\mathcal{N}}_n(t)$ might actually be outside the coverage area of n (*false alarm*), and communication and tracking errors might lead the ego vehicle to consider an actual neighbor to be outside its communication range (*misdetection*).

Under these hypotheses, the performance of the ego vehicle in terms of position estimation accuracy can be assessed by an *error function* $\mathcal{F}_n(t)$, which is a weighted average of the position estimation errors made by the ego vehicle with respect to itself and all its target vehicles. Since computing the ego vehicle's estimation error requires its actual state, no node in the network can know it exactly. Furthermore, each node in the network only has partial and often outdated knowledge about its neighbors. We use the error function to evaluate the performance of different tracking systems in our simulations, as we can compute it offline using our knowledge of the ground truth trajectories. We define the error function as

$$\mathcal{F}_n(t) = \frac{1}{|\hat{\mathcal{N}}_n(t)| + 1}\left(\lambda_{n,n}(t)d(\hat{\mathbf{x}}_{n,n}(t), \mathbf{x}_n(t)) + \sum_{m \in \hat{\mathcal{N}}_n(t)} \lambda_{n,m}(t)d(\hat{\mathbf{x}}_{n,m}(t), \mathbf{x}_m(t))\right). \tag{2.1}$$

In (2.1), $d(\hat{\mathbf{x}}_{n,n}(t), \mathbf{x}_n(t))$ and $d(\hat{\mathbf{x}}_{n,m}(t), \mathbf{x}_m(t))$ represent the error made by n in estimating its own state $n(t)$ and the neighbor state $m(t)$, respectively, and $|\hat{\mathcal{N}}_n(t)|+1$ represents the total number of estimations carried out by n. Finally, $\lambda_{n,m}(t)$ is a scalar value depending on the distance between n and m, computed as

$$\lambda_{n,m}(t) = 1 - \frac{1}{\left(1 + e^{-\epsilon(d(\mathbf{x}_n(t), \mathbf{x}_m(t)) - d_0)}\right)^{1/\nu}}. \tag{2.2}$$

The reader should note that the function in (2.2) fades away as the distance between the ego vehicle and its neighbors grows above d_0. Therefore, the error function in (2.1) is more sensitive to the positioning errors of the nodes closest to the ego vehicle. Hence, we are giving more importance to information updates that may increase the safety of the ego vehicle. The function that describes $\lambda_{n,m}(t)$ is known as Richards' curve [46]: the values of its parameters are listed in Tab. 2.2.

To evaluate the tracking accuracy performance of the whole network, we define $\mathcal{F}(t)$ as the average of $\mathcal{F}_n(t)$ among all vehicles $n \in \mathcal{N}(t)$:

$$\mathcal{F}(t) = \frac{1}{|\mathcal{N}(t)|} \sum_{n \in \mathcal{N}(t)} \mathcal{F}_n(t). \tag{2.3}$$

2.3.3 Tracking System

To minimize the positioning error defined in (2.1), each vehicle $n \in \mathcal{N}(t)$ must estimate its state $\mathbf{x}_n(t)$ and the state $\mathbf{x}_m(t)$ of every other neighbor $m \in \hat{\mathcal{N}}_n(t)$ at any time t. To this goal, the ego vehicle exploits both the information of its onboard sensors and the data received from its neighbors through inter-vehicle communications. We describe the vehicle state evolution with

the CTRA model given in [47], thus considering a constant acceleration $a = \frac{dv}{dt}$ and a non-zero turn rate ω. Besides, we assume the on-board sensors provide a noisy observation $o_n(t)$ at each slot t. The ego vehicle implements a UKF that, at each step t, returns the vehicle state estimate $\hat{\mathbf{x}}_{n,n}(t)$ and the related uncertainty, given by a matrix $\mathbf{P}_{n,n}(t)$. We provide the details of the UKF in Appendix A. In particular, the UKF makes it possible to compute also the target's future trajectory through the *predictive step* of the algorithm.

To allow the ego vehicle to track the surrounding nodes, an extra UKF is deployed for each neighbor $m \in \hat{\mathcal{N}}_n(t)$. Each additional UKF has the task of computing the state estimate $\hat{\mathbf{x}}_{n,m}(t)$ of m and is fed by the information broadcast by such vehicle. At each timeslot t, each vehicle $m \in \mathcal{N}(t)$ can decide to transmit the estimate $\hat{\mathbf{x}}_{m,m}(t)$ and the related covariance matrix $\mathbf{P}_{m,m}(t)$. The time frame by which new transmissions are initiated depends on the selected broadcasting strategy, as described in Sec. 2.4. Each message transmitted by m is received by all the vehicles in $\mathcal{N}_m(t)$ after a communication delay (provided that the transmission is not interfered with, as we will explain later).

Whenever the ego vehicle gets a message from a node not in its current neighbor set $\hat{\mathcal{N}}_n(t)$, it initializes a new UKF with an initial state and uncertainty set equal to those just received. Instead, if the transmitter was known to the ego vehicle, the tracking system is updated by setting $\hat{\mathbf{x}}_{n,m}(t) = \hat{\mathbf{x}}_{m,m}(t)$ and $\mathbf{P}_{n,m}(t) = \mathbf{P}_{m,m}(t)$. In both cases, the predictive step of the UKF allows the ego vehicle to estimate the trajectory of m until a new update is received. Hence, the tracking operations are carried out without transmitting the complete history of observations but only the last computed state estimation, thus saving precious channel resources. We highlight that the UKF algorithm requires using an uncertainty matrix $\mathbf{P}_{m,m}(t)$ to compute the best state estimation. In particular, the transmission of $\mathbf{P}_{m,m}(t)$ is necessary to allow n to predict the future evolution of $\hat{\mathbf{x}}_{n,m}(t)$. If a vehicle n does not receive state updates from a neighbor $m \in \hat{\mathcal{N}}_n(t)$ for a period longer than T_{track}, m is removed from $\hat{\mathcal{N}}_n(t)$.

2.3.4 Channel Access Scheme

Inter-vehicle communications are modeled following the IEEE 802.11p standard, which defines the Physical (PHY) and MAC layer features of the DSRC transmission protocol [38]. The DSRC protocol defines seven different channels at the PHY layer, each containing $C_{\text{tot}} = 52$ subcarriers [39]. In this work, we assume that only a limited number $C \leq C_{\text{tot}}$ of subcarriers is used for broadcasting state information messages, and the remainings are reserved for other applications. In particular, DSRC implements the CSMA/CA scheme at the MAC layer, where nodes listen to the wireless channel before sending.

We consider an ideal 1-persistent CSMA/CA scheme for channel access modeling. In this way, we can successfully arbitrate the channel access among in-range vehicles as long as a single transmission per subcarrier and timeslot are enabled, even in the case of multiple potential transmitters. However, collisions can still occur among out-of-range vehicles that transmit towards the same receiver, an issue known in the literature as the *hidden node* problem. Therefore, the transmission from a vehicle n to a vehicle m will suffer from a hidden terminal collision if

any of m's neighbors that are out of n's range start a transmission that overlaps in time and frequency with n's signal. We also design and implement a congestion control algorithm to reduce the channel collision probability. More details are given in Sec. 2.5.

2.4 Broadcasting Strategies

In this section, we describe the communication strategies used to regulate inter-vehicle communications in our model. In particular, two different solutions are considered, namely Periodic Broadcasting (PB), already implemented by most C-ITS applications, and Error Threshold Broadcasting (ETB), our original proposal.

2.4.1 Benchmark: Periodic Broadcasting

Algorithm 1 PB strategy (Sec. 2.4.1)

Input: $T_{\text{per},n} > 0$, $T_{\text{last},n} > 0$, *new neighbor* $\in$ {*True, False*}
Output: *transmit* $\in$ {*True, False*}
1: *transmit* $\leftarrow$ *False*
2: $T_{\text{last},n} \leftarrow T_{\text{last},n} + 1$
3: **if** $T_{\text{last},n} > T_{\text{per},n}$ **or** (*new neighbor* **and** $T_{\text{last},n} > 2$) **then**
4: *transmit* $\leftarrow$ *True*
5: **end if**
6: **if** *transmit* **then**
7: $T_{\text{last},n} \leftarrow \max\{T_{\text{last},n} - T_{\text{per},n}, 0\}$
8: **end if**
9: **return** *transmit*

In the PB scenario each vehicle $n \in \mathcal{N}(t)$ chooses an *inter-transmission period* $T_{\text{per},n}$. If $T_{\text{per},n}$ remains constant over time, the communication process of n follows an almost regular time-frame. This strategy is a benchmark solution for our analysis as it emulates the broadcast behavior of current vehicular applications, which transmit Cooperative Awareness Messages (CAMs) and Basic Safety Messages (BSMs) at a regular frequency. Reducing $T_{\text{per},n}$ allows any neighbor $m \in \mathcal{N}_n(t)$ to receive the state estimate $\hat{\mathbf{x}}_{n,n}$ of n more frequently, at the expense of increasing the probability of channel access collision. Moreover, we initiate a new transmission each time the ego vehicle senses a new neighbor $m \notin \hat{\mathcal{N}}_n(t)$, and no packets were sent in the previous two timeslots. This feature allows m to immediately update its neighbor set $\hat{\mathcal{N}}_m(t)$, thus reducing the undetection probability. The PB strategy is described in Algorithm 1, where $T_{\text{last},n}$ represents is the time from the last transmission and *"new neighbor"* is a Boolean variable indicating that a vehicle $m \notin \hat{\mathcal{N}}_n(t)$ is detected at slot t.

2.4.2 New Proposal: Error Threshold Broadcasting

In the ETB scenario each vehicle $n \in \mathcal{N}(t)$ chooses an *error threshold* $E_{\text{thr},n}$ and regulates its communication behavior so that the overall position estimation error never exceeds $E_{\text{thr},n}$.

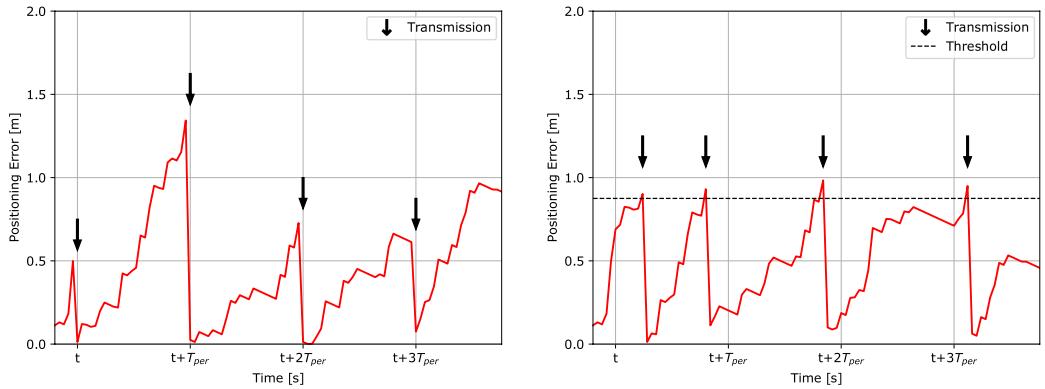
To this goal, the ego vehicle defines an additional UKF replicating the UKF operations of all the neighbor nodes that are tracking the ego vehicle itself. The new filter propagates the ego vehicle's state by using only its *predictive step* with no sensor input, as done by the vehicle's neighbors. Besides, the filter state is updated each time the ego vehicle triggers a new communication, mimicking the operation performed by neighbor vehicles upon receiving the ego vehicle's state update. Hence, in each timeslot t, the ego vehicle knows both the *a posteriori* state estimate $\hat{\mathbf{x}}_{n,n}(t)$, which is the output of its UKF filter, and the *a priori* state estimate $\hat{\mathbf{x}}^p_{n,n}(t)$, which is the output of its purely predictive filter.

Algorithm 2 ETB strategy (Sec. 2.4.2)

Input: $E_{\text{thr},n} > 0$, $T_{\text{last},n} > 0$, *new neighbor* $\in \{\textit{True}, \textit{False}\}$, $\hat{\mathbf{x}}_{n,n}(t) \in \mathbb{R}^6$, $\hat{\mathbf{x}}^p_{n,n}(t) \in \mathbb{R}^6$
Output: *transmit* $\in \{\textit{True}, \textit{False}\}$

transmit $\leftarrow$ *False*
$T_{\text{last},n} \leftarrow T_{\text{last},n} + 1$
if $d(\hat{\mathbf{x}}_{n,n}(t), \hat{\mathbf{x}}^p_{n,n}(t)) > E_{\text{thr},n}$ **or** $T_{\text{last},n} > T_{\max}$ **or** (*new neighbor* **and** $T_{\text{last},n} > 2T$) **then**
    transmit $\leftarrow$ *True*
end if
if *transmit* **then**
    $T_{\text{last},n} \leftarrow \max\{T_{\text{last},n} - T_{\max}, 0\}$
    $\hat{\mathbf{x}}^p_{n,n}(t) \leftarrow \hat{\mathbf{x}}_{n,n}(t)$
end if
return *transmit*

In each timeslot, the two different estimates are compared: if the difference $d(\hat{\mathbf{x}}_{n,n}(t), \hat{\mathbf{x}}^p_{n,n}(t))$ exceeds $E_{\text{thr},n}$, a new transmission is initiated. We observe that, as before, the communication process can vary according to specific events. A maximum *inter-transmission period* $T_{\max}$ is defined to mitigate undetection errors, and the ego vehicle initiates additional transmissions whenever new neighbors are detected. This strategy is described in Algorithm 2, where $T_{\text{last},n}$ and *"new neighbor"* are defined as in Sec. 2.4.1.



(a) PB strategy. **(b)** ETB strategy.

Fig. 2.2: Temporal evolution of the positioning error for the broadcasting strategies.

An intuitive understanding of the PB and ETB dynamics is provided by Figs. 2.2a and 2.2b, which represent the evolution of the position error $d(\hat{\mathbf{x}}_{n,n}(t), \hat{\mathbf{x}}^p_{n,n}(t))$ according to the transmission process in the two cases. In the PB scenario, we can observe that new transmissions are initiated in a regular fashion, regardless of the value of $d(\hat{\mathbf{x}}_{n,n}(t), \hat{\mathbf{x}}^p_{n,n}(t))$. Instead, in the ETB scenario, new transmissions are initiated only when $d(\hat{\mathbf{x}}_{n,n}(t), \hat{\mathbf{x}}^p_{n,n}(t))$ is above a certain threshold. Despite this, the complexity of the ETB strategy is only slightly higher than that of the PB strategy: the ego vehicle needs to maintain an additional UKF performing only the predictive step in each timeslot.

In both the PB and ETB scenarios, the best approach would require any vehicle $n \in \mathcal{N}(t)$ to determine the optimal value of $T_{\text{per},n}$ and $E_{\text{thr},n}$ at any time t. We assume these values to be computed by the network infrastructure through a *brute-force* approach, iterating on all the possible number of available subcarriers, vehicular densities, road characteristics, and other specific parameters. However, this approach is only possible in simulations, as the computational and communication loads would be too large for real scenarios. It is far more practical to use less resource-heavy congestion control techniques, as we will show in the next section.

2.5 Congestion Control

The dynamic nature of the vehicular networks makes the broadcasting strategies described in Sec. 2.4 inefficient without congestion control mechanisms that regulate information dissemination as a function of the network load. In our scenario, congestion control must allow each vehicle $n \in \mathcal{N}(t)$ to independently adjust the values of $T_{\text{per},n}$ and $E_{\text{thr},n}$, so as to adapt the communication process in real-time. In the following, we first design a benchmark congestion control mechanism, named Channel Sensing Congestion Control (CSCC), based on the LIMERIC protocol [40]. This latter is one of the most widely adopted schemes for congestion control [48] and relies on channel sensing, like most state-of-the-art approaches. Hence, we design an alternative congestion control approach, called Neighbor Aware Congestion Control (NACC), that exploits network topology information to reduce the packet collision probability. Finally, we find an expression for the packet collision probability as a function of the network load and specify how to integrate the congestion control schemes with the proposed ETB strategy.

2.5.1 Benchmark: Channel Sensing Congestion Control

In the CSCC scenario, each vehicle $n \in \mathcal{N}(t)$ constantly listens to the wireless channel and estimates the number of resources that it is allowed to use to avoid congestion. Now, let us assume that n is assigned to subcarrier $c_n \in \{0, 1, ..., C-1\}$. In each timeslot, n senses the channel, looking for new packet transmissions performed by neighbor vehicles. We denote by $K_n(t)$ the local channel busy ratio, i.e., the fraction of time during which the channel is busy, considering the last T_{cbr} timeslots. Hence, every T_{avg} timeslots, the value of $K_n(t)$ is smoothed as

$$K_{\text{avg},n}(t) = 0.5 \cdot K_{\text{avg},n}(t) + 0.5 \cdot K_n(t). \tag{2.4}$$

Algorithm 3 CSCC protocol (Sec. 2.5)

Input: $K_{\text{avg},n}(t) > 0$, $\rho_n(t-1) > 0$
Output: $\rho_n(t) > 0$
  if $K_{\text{target}} - K_{\text{avg},n}(t) > 0$ **then**
    $\delta = \min\left(\beta \cdot (K_{\text{target}} - K_{\text{avg},n}(t)), \delta_{\max}\right)$
  else
    $\delta = \max\left(\beta \cdot (K_{\text{target}} - K_{\text{avg},n}(t)), \delta_{\min}\right)$
  end if
  $\rho_n(t) = [(1-\alpha) \cdot \rho_n(t-1) + \delta]_0^1$
  return $\rho_n(t)$

In (2.4), $K_{\text{avg},n}(t)$ is the vehicle average channel busy ratio and represents the channel occupancy sensed by n over the subcarrier c_n. In the CSCC approach, each vehicle $n \in \mathcal{N}(t)$ aims at keeping the value of $K_{\text{avg},n}(t)$ as close as possible to a target value K_{target}. Practically, every T_{avg} timeslots, n evaluates the difference between $K_{\text{avg},n}(t)$ and K_{target} and updates accordingly the value of $\rho_n(t)$, which is the fraction of time used by n to transmit over the wireless channel. This procedure is described in Algorithm 3, where α and β are two tuning parameters, and $[x]_a^b = \min(\max(x, a), b)$, with $a \leq b$. We set α and β according to [40], reporting their values in Sec. 2.6.

When n adopts the PB strategy described in Sec. 2.4.1, the value of $T_{\text{per},n}$ at slot t is updated as $T_{\text{per},n}(t) = 1/\rho_n(t)$. In case n is adopting the ETB strategy described in Sec. 2.4.2, $\rho_n(t)$ should be determined as a function of $E_{\text{thr},n}$, as explained in Sec. 2.5.3.

2.5.2 New Proposal: Neighbor Aware Congestion Control

In the NACC approach, each vehicle $n \in \mathcal{N}(t)$ computes the value of $\rho_n(t)$ as a function of its knowledge about its neighbors' positions. In particular, vehicles can increase or decrease channel occupancy with the aim of minimizing the packet collision probability. To design NACC, we first theoretically model the communication that takes place in a group of vehicles when CSMA/CA is implemented at the MAC layer, as is case of IEEE 802.11p. Then, we describe how a user can estimate the number of neighbors that may potentially result in packet collisions. Finally, we find a relation between the vehicular density sensed by a user and the packet collision probability itself.

In the following, we make some simplifying assumptions for the tractability of the analysis. First, we assume an ideal CSMA/CA mechanism, capable of perfectly arbitrating the channel access among in-range vehicles so that only one node at a time can transmit. However, nodes that are mutually hidden can transmit simultaneously. We consider that the communication channel is error-free and that packets are always received if the distance between the vehicles is below the communication range d_r, except in the case of collisions with hidden nodes. Moreover, we assume that packet capture is impossible: if two packets collide, the receiver cannot decode either. We also assume that vehicles attempt to transmit packets according to a Poisson process and have the same traffic rate, i.e., transmission probability, ρ. Finally, to make the model tractable from a mathematical point of view, we assume that the vehicular density is constant in

the considered map. Deriving the optimal congestion control strategy without these assumptions would require a more cumbersome analysis and some extra signaling overhead for the vehicles. Although the policy obtained under these simplifying assumptions may be suboptimal in a more realistic setting, it still provides a reasonable strategy that, as proved by our simulation results, can achieve better performance than current state-of-the-art solutions.

CSMA/CA Analysis

We saw in Sec. 2.3.4 that vehicles access the channel following an ideal 1-persistent CSMA/CA protocol. We now consider a group of N^* vehicles that share the same subcarrier $c \in \{0, 1, ..., C-1\}$. We assume that all these vehicles are always mutually in-range, i.e., at a distance lower than d_r, and that no other vehicle can interfere with their communication. In the case of multiple vehicles with pending packets, the ideal CSMA/CA algorithm will let just one of them transmit in a time slot. Denoting by $u(t)$ the number of vehicles that want to access the channel during the timeslot t, we can write $u(t) = \max(u(t-1) + a(t) - 1, 0)$, where $a(t)$ is the number of new vehicles attempting a transmission while at most one vehicle managed to transmit in timeslot t_1 (provided that $u(t-1) > 0$). The memoryless nature of the Poisson arrival process allows us to calculate the probability of $a(t)$ given $u(t-1)$:

$$P(a(t) = a | u(t-1) = u, \rho, N^*) = \binom{N^* - u}{a} \rho^a (1-\rho)^{N^* - u - a}. \tag{2.6}$$

Naturally, $P(a(t) = a | u(t-1) = u, \rho, N^*) = 0$ if $a > N^* - u$. Since the probability of accessing the channel is the same for all vehicles, the probability that a specific vehicle trying to access the channel will transmit is $(u(t)+1)^{-1}$.

Assuming that ρ and N^* are constant, the channel dynamics at the end of any timeslot t are completely characterized by the number of users that need to transmit, i.e., the queue size $u(t)$. Hence, we can describe the overall system as a Markov Chain (MC), whose state $u(t)$ is in the set $\mathcal{U} = \{0, \ldots, N^* - 1\}$ and whose transition probability matrix $\mathbf{T}(\rho, N^*)$ is given in (2.7).

$$T_{u,z}(\rho, N^*) = \begin{cases} 0, & z < u - 1; \\ P(a(t) = z - u + 1 | u(t-1) = u, \rho, N^*), & 0 < u < N^*,\ u - 1 \leq z < N^*; \\ P(a(t) = z + 1 | u(t-1) = q, \rho, N^*), & u = 0,\ 0 < z < N^*; \\ P(a(t) \leq 1 | u(t-1) = q, \rho, N^*), & u = 0,\ z = 0. \end{cases} \tag{2.7}$$

The steady-state distribution of u is the left eigenvector $\Pi(\rho, N^*) = [\ \Pi_0(\rho, N^*), \Pi_1(\rho, N^*), ..., \Pi_{N^*-1}(\rho, N^*)]$ of $\mathbf{T}(\rho, N^*)$ with eigenvalue equal to 1, normalized so that it is a valid probability distribution:

$$\begin{cases} \Pi(\rho, N^*)\mathbf{T}(\rho, N^*) = \Pi(\rho, N^*); \\ \sum_{u=0}^{N^*-1} \Pi_u(\rho, N^*) = 1. \end{cases} \tag{2.8}$$

We can compute the eigenvector with well-known algebraic methods and normalize it to get the distribution of u. Hence, we can obtain the probability of different transmission events; in particular, the probability of having no transmissions during a timeslot t is

$$P(u(t-1)=0, a(t)=0|\rho,\mathcal{N}) = \Pi_0(\rho,\mathcal{N}) \cdot P(a(t)=0|u(t-1)=0,\rho,\mathcal{N}). \tag{2.9}$$

Vehicle Position Distribution

We recall that our objective is to minimize the number of packet collisions, which in our model are caused only by the hidden terminal problem. Let us consider that the ego vehicle $n \in \mathcal{N}(t)$ is sending a packet to any vehicle $m \in \mathcal{N}_n(t)$, whose distance from n is $d_{n,m}$. To compute the collision probability of n, we should estimate how many neighbors of m can interfere with the communication: we denote this value by $N_{\mathrm{ht},n}(t)$. Since the vehicular density is constant, we can assume $\mathcal{N}_m(t) \simeq \hat{\mathcal{N}}_n(t)$. Hence, $N_{\mathrm{ht},n}(t)$ can be estimated as

$$\hat{N}_{\mathrm{ht},n}(t) = \frac{|\hat{\mathcal{N}}_n(t)|+1}{C} \frac{\pi d_r^2 - \mathbb{E}\left[\Phi(d_{n,m})\right]}{\pi r^2}. \tag{2.10}$$

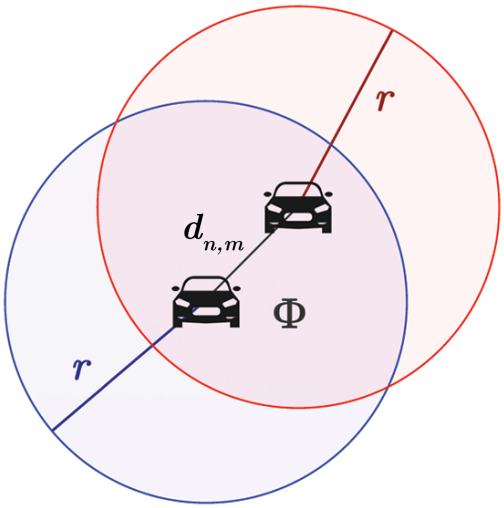


Fig. 2.3: Intersection $\Phi(d_{n,m})$ of the communication ranges of n and m.

In (2.10), $\frac{|\hat{\mathcal{N}}_n(t)|+1}{C}$ is the estimate of the number of vehicles contained in the communication area of m that are using the same subcarriers as n, while $\Phi(d_{n,m})$ is the intersection of the communication areas of n and m. Hence, $\pi d_r^2 - \Phi(d_{n,m})$ is the size of the area within the coverage of m but not of n, i.e., the area from which transmission would be hidden from n, possibly causing a hidden node collision. A graphical representation of this scenario is reported

in Fig. 2.3 while the mathematical expression of $\Phi(d_{n,m})$ is given by

$$\Phi(d_{n,m}) = 2d_r \left(d_r \arccos\left(\frac{d_{n,m}}{2d_r}\right) - \frac{d_{n,m}}{2}\sqrt{1-\left(\frac{d_{n,m}}{2d_r}\right)^2} \right). \tag{2.11}$$

The probability distribution of $d_{n,m}$ is equal to $f_d(d_{n,m}) = \frac{2d_{n,m}}{d_r^2}$. Given (2.11), the mean value of $\Phi(d_{n,m})$ can be computed as

$$\mathbb{E}\left[\Phi(d_{n,m})\right] = \int_0^{2d_r} \Phi(\sigma) f_d(\sigma) d\sigma = d_r^2 \left(\pi - \frac{3\sqrt{3}}{4}\right). \tag{2.12}$$

Replacing $\mathbb{E}\left[\Phi(d_{n,m})\right]$ in (2.10) we finally obtain the expression

$$\hat{N}_{\mathrm{ht},n}(t) = \frac{|\hat{\mathcal{N}}_n(t)|+1}{C}\frac{3\sqrt{3}}{4\pi}. \tag{2.13}$$

Packet Collision Probability

On average, there are $\hat{N}_{\mathrm{ht},n}(t)$ vehicles that can interfere with the communication of the ego vehicle. Hence, according to our channel model, the probability that the transmission will not fail corresponds to the case that none of those $\hat{N}_{\mathrm{ht},n}(t)$ interfering nodes transmits during t. Assuming that the ego vehicle $n \in \mathcal{N}(t)$ has transmission probability ρ_n, while all its interfering vehicles have the same transmission probability $\rho = \rho_n$ and do not interact with other network nodes during t, the packet collision probability P_{coll} of $n \in \mathcal{N}(t)$ at slot t can be derived from (2.9), obtaining

$$P_{\mathrm{coll}}(\rho, \hat{N}_{\mathrm{ht},n}(t)) = 1 - \Pi_0(\rho, \hat{N}_{\mathrm{ht},n}(t))(1-\rho)^{\hat{N}_{\mathrm{ht},n}(t)}. \tag{2.14}$$

Assuming that the ego vehicle uses the same transmission probability as the interfering nodes, each vehicle $n \in \mathcal{N}(t)$ can regulate $\rho_n(t)$ such that its collision probability equals a predetermined threshold P_{thr}, i.e., settings $P_{\mathrm{coll}}(\rho_n(t), \hat{N}_{\mathrm{ht},n}(t)) = P_{\mathrm{thr}}$. This expression can be solved using Eq. (2.14), since the vehicle knows the value of $\hat{N}_{\mathrm{ht},n}(t)$. Therefore, we need to find the solution of:

$$1 - \Pi_0(\rho_n(t), \hat{N}_{\mathrm{ht},n}(t))(1-\rho_n(t))^{\hat{N}_{\mathrm{ht},n}(t)} = P_{\mathrm{thr}} \tag{2.15}$$

Since the steady-state distribution is proper, and the eigenvalues are a continuous function of $\rho_n(t)$, we can prove that a real-valued solution exists. We know that if $\rho_n(t) = 0$, no vehicle ever transmits and $\Pi_0(0, \hat{N}_{\mathrm{ht},n}(t)) = 1$ for any $\hat{N}_{\mathrm{ht},n}(t)$, and if $\rho_n(t) = 1$, all vehicles always try to transmit, and $\Pi_{\hat{N}_{\mathrm{ht},n}(t)-1}(1, \hat{N}_{\mathrm{ht},n}(t)) = 1$ for any $\hat{N}_{\mathrm{ht},n}(t)$. Hence, the intermediate value theorem ensures that the problem has a solution for any $P_{\mathrm{thr}} \in [0,1]$ and, in particular, we can explicitly solve the equation if the number of potential interferers $\hat{N}_{\mathrm{ht},n}$ is small.

In the following, we derive the solution of (2.15) with $\hat{N}_{\mathrm{ht},n} = 2$, which is equivalent to

having just two possible interferers. In doing so, we omit the time dependency for readability's sake. If we apply Eq. (2.7), we obtain:

$$T(\rho_n) = \begin{bmatrix} 1-\rho_n^2 & \rho_n^2 \\ (1-\rho_n) & \rho_n \end{bmatrix} \tag{2.16}$$

Since the state of the Markov model represents the number of nodes waiting to transmit right after a transmission opportunity, both the vehicles cannot be in the queue concurrently, and state 2 is unreachable. We can now solve the system in Eq. (2.8):

$$\begin{cases} (1-\rho_n^2)\Pi_0 - \Pi_0 + (1-\rho_n)\Pi_1 = 0, \\ \Pi_0 + \Pi_1 = 1, \end{cases} \tag{2.17}$$

which results in the following expression for Π_0 and, consequently, for P_{coll}:

$$\Pi_0 = \frac{1-\rho_n}{\rho_n^2 - \rho_n + 1}, \tag{2.18}$$

$$P_{\text{coll}} = 1 - \frac{(1-\rho_n)^3}{\rho_n^2 - \rho_n + 1}. \tag{2.19}$$

Eq. (2.15) then becomes:

$$\rho_n^3 - (2 + P_{\text{thr}})\,\rho_n^2 + (2 + P_{\text{thr}})\,\rho_n - P_{\text{thr}} = 0. \tag{2.20}$$

The solution is now easy to compute: for $P_{\text{thr}} = 0.3$, the optimal transmission rate is $\rho_n \simeq 0.152$. If the number of potential interferers increases, the resolution requires higher-order polynomial equations that cannot be solved explicitly.

Algorithm 4 NACC protocol (Sec. 2.5.2)

Input: $\hat{N}_n(t) \geq 0$
Output: $\rho_n(t) > 0$
$\hat{N}_{\text{ht},n}(t) = \frac{\hat{N}_n(t)+1}{C}\frac{3\sqrt{3}}{4\pi}$
$\rho_n(t) = \rho$ such that $\Pi_0(\rho, \hat{N}_{\text{ht},n}(t))(1-\rho)^{\hat{N}_{\text{ht},n}(t)} = P_{\text{thr}}$
return $\rho_n(t)$

By computing ρ_n, the ego vehicle can tune its channel occupancy according to its estimation of the network load. We report the overall operations of the proposed NACC approach in Algorithm 4. This congestion control strategy makes each vehicle $n \in \mathcal{N}(t)$ change the value of $\rho_n(t)$ according to the vehicular density in its surroundings. In case the vehicle is using the PB strategy described in Sec. 2.4.1, the value of $T_{\text{per},n}$ is updated as $T_{\text{per},n}(t) = \frac{1}{\rho_n(t)}$. Conversely, if the ETB strategy described in Sec. 2.4.1 is implemented, $\rho_n(t)$ will be updated as a function of $E_{\text{thr},n}$, as explained in Sec. 2.5.3. We highlight that, by adjusting the value of $T_{\text{per},n}$ in this way, we violate the assumption regarding the distribution of the packet inter-transmission

time considered in the definition of the system Markov model. Indeed, the time between two subsequent transmissions is constant with the PB strategy, while it depends on the position error evolution in the ETB scenario. This approximation may impair the performance of our congestion control: in particular, we expect to observe a significant performance reduction in the case of the PB strategy.

2.5.3 Implementing Congestion Control for the ETB Strategy

Both the CSCC and the NACC approaches improve the efficiency of the broadcasting strategies described in Sec. 2.4 by adapting the inter-transmission period to the deployment scenario. As stated previously, to combine a congestion control scheme with the ETB strategy, we have to relate the inter-transmission period to the error threshold. Practically, we need to build a map $\mathcal{M}$ such that the transmission period $T_{\text{per}} = \mathcal{M}(E_{\text{thr}})$ yields an average map estimation error close to E_{thr}. Unfortunately, the relation between E_{thr} and T_{per} is subject to multiple factors and cannot be easily modeled. The function $\mathcal{M}$ depends on how the position estimation error of vehicles evolves in time, i.e., on both the road map and the users' behaviors.

To reach our goal, we hence resorted to a pragmatic approach. By simulating a purely predictive UKF in the considered scenario, we derive an empirical estimate of the statistical distribution $P(e_h \leq E_{\text{thr}})$ of the position estimation error e_h after h timeslots since the last update, for any $h \geq 0$. Denoting by H_{thr} the number of timeslots in which the error e_h exceeds the threshold E_{thr}, we can set $T_{\text{per}} = \mathbb{E}[H_{\text{thr}}]$. Now, pretending that the errors e_h can be modeled as independent random variables, the complementary cumulative distribution function of H_{thr} can be expressed as

$$P(H_{\text{thr}} > H) = \prod_{h=1}^{H} P(e_h \leq E_{\text{thr}}), \tag{2.21}$$

from which we easily get

$$T_{\text{per}} = \sum_{H=1}^{\infty} \prod_{h=1}^{H} P(e_h \leq E_{\text{thr}}). \tag{2.22}$$

Equation (2.22) hence provides the desired map $\mathcal{M}$ from the error threshold E_{thr} to the inter-transmission period T_{per}. This function can also be used to determine the value ρ of the broadcast policy ETB, which can be computed as $\rho = \frac{1}{\mathcal{M}(E_{\text{thr}})}$. We highlight that this approach requires that vehicles know the distribution of the position estimation error in the map. Such information can be provided to vehicles by the road infrastructure or pre-programmed into the channel access algorithm (possibly with multiple choices, depending on the road conditions). The investigation of such aspects, however, is left to future work.

2.6 Simulation Settings and Results

In this section, we evaluate the performance of the proposed ETB strategy for broadcasting operations compared to PB. Moreover, we exemplify how the proposed NACC mechanism can improve the performance of the broadcasting strategies by exploiting network topology information with respect to the benchmark CSCC scheme that relies only on channel sensing.

2.6.1 System Settings

We assume vehicles to communicate in the legacy band, i.e., at 5.9 GHz, using the IEEE 802.11p protocol described in Sec. 2.3.4, which is implemented in `python` in our simulations. We use the common piecewise long-distance propagation loss model with Nakagami fading [31], using the parameters derived in [49], which yield a maximum communication range d_r of about 140 m. In order to simulate the packet error rate for a given SNR, we use the analytical model given in [50], with a physical layer rate B_{phy} Mb/s and a packet length $L_{\text{pkt}} = 200$ bytes. In the case of collisions, we assume that both the transmissions fail with no packet capture. The communication delay and the timeslot duration are both set to $\tau = 100$ ms. When congestion control is implemented, T_{per} and E_{thr} are automatically adjusted as a function of the network load. When not implementing a congestion control scheme, instead, the settings of both the PB and ETB strategies must be defined *a priori*, which means that all the vehicles adopt the same T_{per} and E_{thr}, respectively. In our simulations we adopt an exhaustive approach and consider $N_{\text{set}} = 30$ different settings. In particular, we make the inter-transmission period T_{per} vary from 0 to 10 seconds while the error threshold E_{thr} ranges between 0 and 42 meters: the trade-off involves both estimation accuracy and broadcasting overhead.

(a) Openstreetmap scenario. **(b)** SUMO scenario.

Fig. 2.4: Urban scenario considered for the performance evaluation.

For our simulations, we use road map data imported from OpenStreetMap (OSM), an open-source software that combines wiki-like user-generated data with publicly available information. In particular, we consider the OSM map of New York City, as represented in Fig. 2.4a, to characterize a dynamic urban environment. To consider realistic vehicular trajectories, we simulate the mobility of cars using SUMO, as represented in Fig. 2.4b. The vehicles move through the street network according to a `randomTrip` mobility model, which generates trips with random origins and destinations, and speeds that depend on the realistic interaction of the vehicle with the road and network elements. The maximum speed is $v_{\max} = 13.89$ m/s, which is consistent with current speed limits. Given $v_{\max}$, we set $d_0 = 42$ m, which is the distance traveled in 3 seconds by a vehicle running at the maximum speed. In this way, d_0 represents the maximum *safety distance* held in an urban scenario. Following the work in [51], we consider a vehicular density of $\eta = 120$ vehicles/km^2 for medium traffic conditions. Given the total road map area of $A = 0.5168$ km^2, the number of vehicles deployed in the considered scenario is $N = 62$, which corresponds to the cardinality $|\mathcal{N}|$ of the vehicle set.

Tab. 2.2: Simulation settings.

Parameter	Value	Description	Parameter	Value	Description
T_{sim}	10^3 slot	Simulation duration	T_{per}	$\{0, ..., 100\}$ slot	Inter-transmission period
N_{sim}	20	Number of runs	E_{thr}	$\{0, ..., 42\}$ m	Error threshold
τ	100 ms	Timeslot duration	C	{2, 4, ..., 10}	Number of subcarriers
N	62	Number of vehicles	T_{cbr}	10 slot	Interval in which K_n is computed
T_{track}	100 slot	Maximum tracking time	T_{avg}	2 slot	Interval between $K_{\text{avg},n}$ updates
$v_{\max}$	13.89 m/s	Maximum speed	ϵ, ν	0.05, 0.2	Logistic function parameters
d_0	42 m	Safety distance	α	0.1	CSCC speed parameter
A	0.5168 km^2	Area size	β	$(2-\alpha)C/N$	CSCC convergence parameter
η	120 vehicles/km^2	Vehicular density	$\delta_{\min}, \delta_{\max}$	$-1, 1$	Lower and upper bounds of δ
q	1	Process noise parameter	K_{target}	0.68	Target value for K_n
$R_{1,1}$	1.18535 m^2	Accuracy along x	P_{thr}	0.3	Collision probability threshold
$R_{2,2}$	1.18535 m^2	Accuracy along y	L_{pkt}	200 B	Packet length
$R_{3,3}$	0.5 (m/s)^2	Speed accuracy	P_{tx}	16 dBm	Transmission power
$R_{4,4}$	0.39 (m/s^2)^2	Acceleration accuracy	B_{phy}	3 Mb/s	Physical layer rate
$R_{5,5}$	0.09211 rad^2	Heading accuracy	d_r	140 m	Communication range
$R_{6,6}$	0.01587 (rad/s)^2	Turn rate accuracy	C_{tot}	52	Total subcarrier number

The mobility data produced by SUMO are processed by an ad hoc `python` simulator, designed to model both the tracking and the communication processes performed by the system vehicles. As we assessed in Sec. 2.3, the behavior of each node $n \in \mathcal{N}(t)$ can be fully represented by its state $\mathbf{x}_n(t)$. In particular, measurements of the components of $\mathbf{x}_n(t)$ are affected by a non-negligible noise, modeled as a Gaussian process with zero mean and covariance matrix R. The diagonal elements of R are given in Tab. 2.2 and are derived from the models in [52, 53, 54]. Instead, as explained in Appendix A, we define the process noise as a Gaussian process with zero mean and covariance matrix $Q = qI$, where I denotes the identity matrix and q is a scalar value.

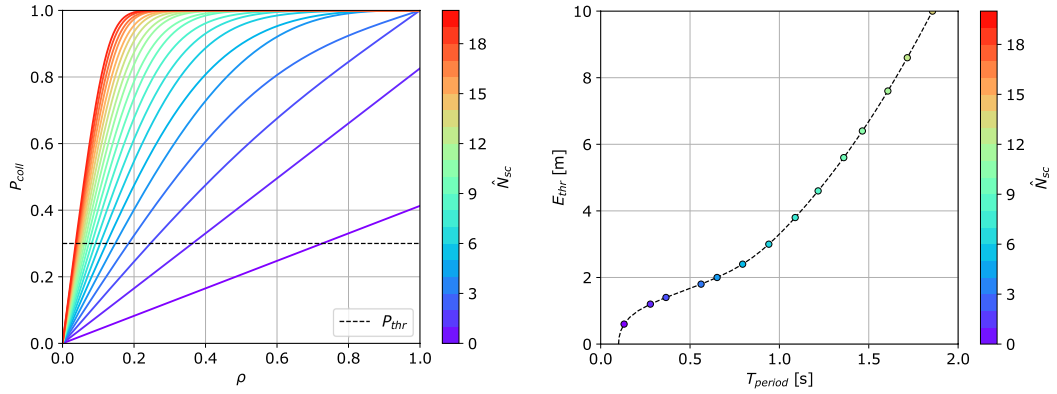
Tab. 2.2 also reports the parameters of the congestion control schemes from Sec. 2.5. For what concerns the CSCC approach, we use the same parameters suggested in [40]: $K_{\text{target}} = 0.68$, so that vehicles aim at occupying the channel about 68% of the time, and $\alpha = 0.1$, which ensures a sufficiently high convergence speed. Finally, we set $\beta = (2-\alpha)C/N$, where C/N represents the maximum number of users sharing the same communication channel, ensuring the algorithm's

convergence for any N. For what concerns our proposed NACC approach, we set the collision probability threshold to $P_{\text{thr}} = 0.3$. We choose this value to allow a fair comparison between the CSCC and the NACC approaches. Indeed, setting $K_{\text{target}} = 0.68$ and $P_{\text{coll}} = 0.3$, we obtain similar values of mean inter-transmission time $\overline{T}_{\text{tx}}$ for each combination of broadcasting strategy and congestion control mechanism.

To evaluate the performance of the proposed broadcasting strategies in the simulations, we take into account four main factors, namely:

- *Average positioning error*, i.e., the average error of the ego vehicle when estimating its position and that of its neighbors, which is given by (2.1);
- *95th percentile of the positioning error*, i.e., the positioning error threshold exceeded only by the worst 5% of the vehicles;
- *Detection error*, i.e., the sum of the misdetection (i.e., unknown vehicles in the ego vehicle communication area) and false detection (i.e., vehicles that are believed to be in the neighborhood but are actually beyond the communication range) event probabilities;
- *Packet collision rate*, i.e., the average number of packet collisions per vehicle and per second that occur because of the hidden terminal problem.

2.6.2 Theoretical Validation



(a) Collision probability. **(b)** Function $\mathcal{M}^{-1}$.

Fig. 2.5: Congestion control analysis.

In Fig. 2.5a, we plot P_{coll} as a function of ρ_n for different values of $\hat{N}_{\text{sc},n} = \lceil \frac{\hat{N}_n+1}{C} \rceil$, which represents the estimate of the number of users in the communication range of the ego vehicle using its same subcarrier. By looking at Fig. 2.5a, we observe that P_{coll} increases with both the transmission probability and the number of interfering neighbors. When using the NACC approach, the ego vehicle sets the transmission probability ρ_n according to both $\hat{N}_{\text{sc},n}$ and P_{coll}.

Hence, the value of $T_{\text{per},n}$ is updated as $\frac{1}{\rho_n}$ while the value of $E_{\text{thr},n}$ is updated as $\mathcal{M}^{-1}\left(\frac{1}{\rho_n}\right)$. In Fig. 2.5b, we represent the function $\mathcal{M}^{-1}$ used for this purpose. In particular, the colored dots represent the different values of $E_{\text{thr},n}$ that are chosen according to both $\hat{N}_{\text{sc},n}$ and P_{coll}.

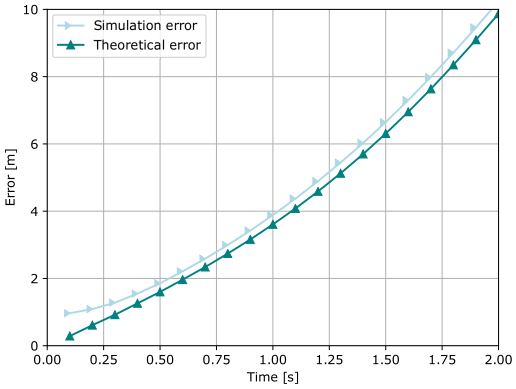


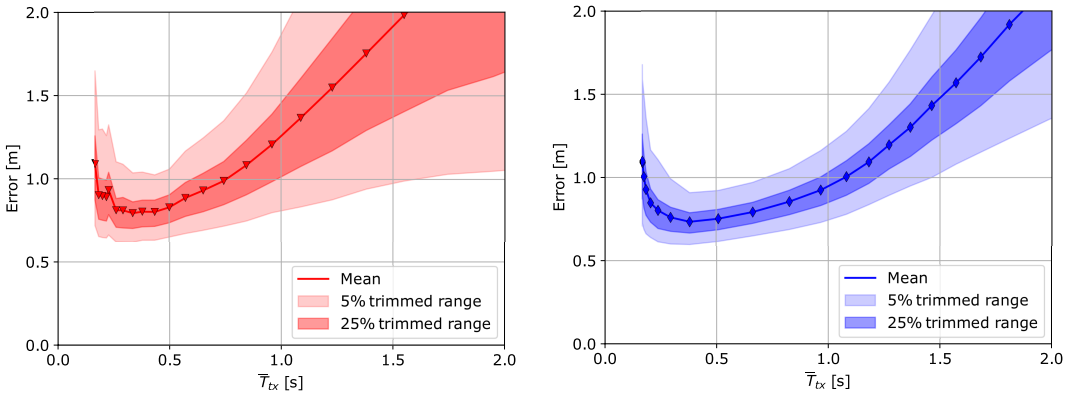
Fig. 2.6: Simulation vs. theoretical positioning error in a highway scenario.

To validate our analysis, we show how the average tracking error evolves considering the output of the Kalman filter and the empirical results obtained through simulation. In the first case, the motion of a vehicle is analytically represented as a rectilinear motion, disregarding the users' driving imperfections, and the vehicle's speed and direction are assumed constant. In the second case, mobility traces are generated using SUMO. In Fig. 2.6, we represent the average positioning error obtained in the two different cases as a function of time. We observe that the two data trends are very similar, thereby validating our theoretical framework. The curves' gap is due to the driving imperfections that cannot be predicted by the considered motion model.

2.6.3 Performance Analysis

In the rest of the section, the performance of the broadcasting strategies and the congestion control schemes that we described in Sec. 2.4 and Sec. 2.5, respectively, are analyzed via simulations according to the evaluation methodology described in Sec. 2.6.1. We derive the results through a Monte Carlo approach, where N_{sim} independent simulations of $T_{\text{sim}} = 1000$ slots are repeated to get different statistical quantities of interest. This approach has the advantage of increasing the realism and generalizability of our analysis compared to the evaluation provided in the previous section. Besides, it makes it possible to estimate the system performance by accounting for realistic channel behaviors and traffic dynamics. All the results are obtained with the communication channel model described in Sec. 2.6.1.

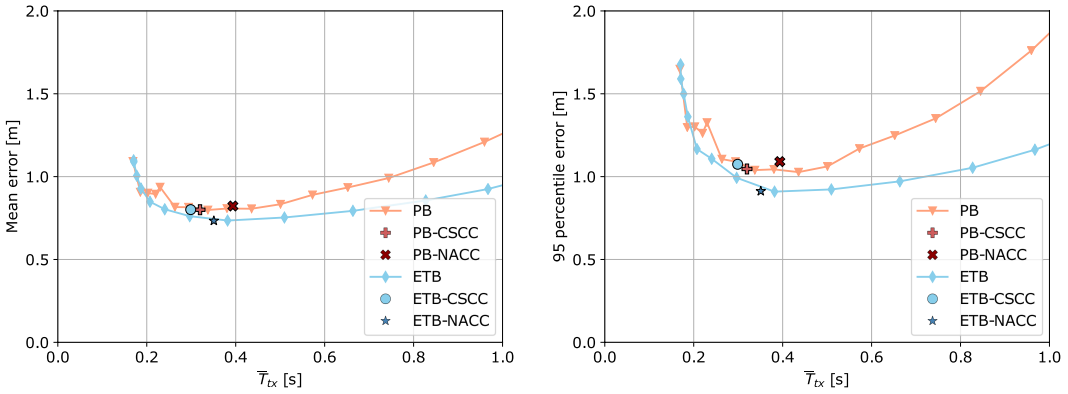
At first, we fix the number of the available subcarriers to $C = 8$. Later, we will verify how different values of C may influence the simulation outcomes. In Fig. 2.7a and Fig. 2.7b, we analyze the statistics of the positioning error as a function of the mean inter-transmission time $\overline{T}_{\text{tx}}$, which is an indicator of the total channel occupancy.



(a) PB strategy. **(b)** ETB strategy.

Fig. 2.7: Positioning error distribution with $C = 8$.

We highlight that $\overline{T}_{\text{tx}}$ does not coincide with the inter-transmission period used in the PB strategy. Indeed, while T_{per} is defined *a priori* and can take all the values within the set $\{0.1 \text{ s}, ..., 10 \text{ s}\}$, $\overline{T}_{\text{tx}}$ is an outcome of the simulation. In particular, in a realistic scenario, $\overline{T}_{\text{tx}}$ never goes below the value of 0.2 s, i.e., two timeslots, because of the channel access contention. From Fig. 2.7a and Fig. 2.7b, we can also observe how the limits of CSMA/CA affect the positioning error: when the number of channel access requests is too high, i.e., $\overline{T}_{\text{tx}} < 0.3$ s, the channel gets congested and, consequently, the performance of the overall system degrades. Indeed, the positioning error can be described by a convex curve, with a minimum for $\overline{T}_{\text{tx}} \approx 0.3$ s; this value represents the level of channel occupancy that guarantees the best position estimation accuracy. By comparing Fig. 2.7a with Fig. 2.7b, we can observe that the ETB strategy outperforms the benchmark PB strategy. In particular, ETB proves to have a slightly lower average error and significantly lower error variance.



(a) Mean positioning error. **(b)** 95th percentile of the positioning error.

Fig. 2.8: Mean and 95th percentile of the positioning error with $C = 8$.

By optimizing the channel access requests, the ETB strategy has a lower positioning error

than the benchmark strategy. Fig. 2.8a and Fig. 2.8b show a direct comparison between the considered broadcasting strategies; in particular, Fig. 2.8a reports the mean error while Fig. 2.8b shows the 95th percentile of the error. In both cases, the ETB strategy ensures better position estimation accuracy for the same level of channel occupancy. The marks in Fig. 2.8a and Fig. 2.8b represent the performance of the congestion control schemes designed in Sec.2.5. Since congestion control can adapt the values of T_{per} and E_{thr} and the communication strategy to the scenario in real-time, we obtain a single outcome for each combination of broadcasting strategy and congestion control approach. First, we observe that all the deployed solutions succeed in maintaining the channel occupancy close to the optimal working point, i.e., $\overline{T}_{\text{tx}} \approx 0.3$ s. Among all the possible solutions, the combination of the ETB strategy with the NACC approach ensures the best performance. This scheme outperforms the classical approach used in the literature, represented by the combination of the PB strategy with the CSCC approach. Our solution obtains a 10% gain when considering the mean error and a 20% gain when considering the 95th percentile of the error, thereby strongly improving the positioning accuracy for safety applications [7, 55].

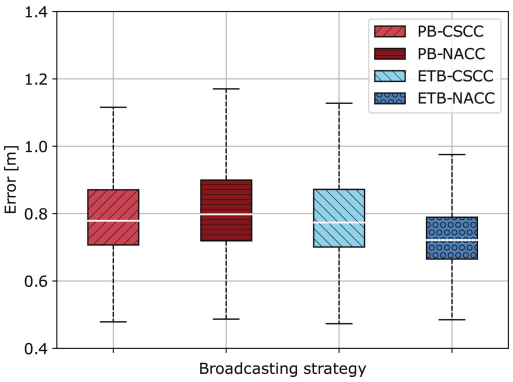
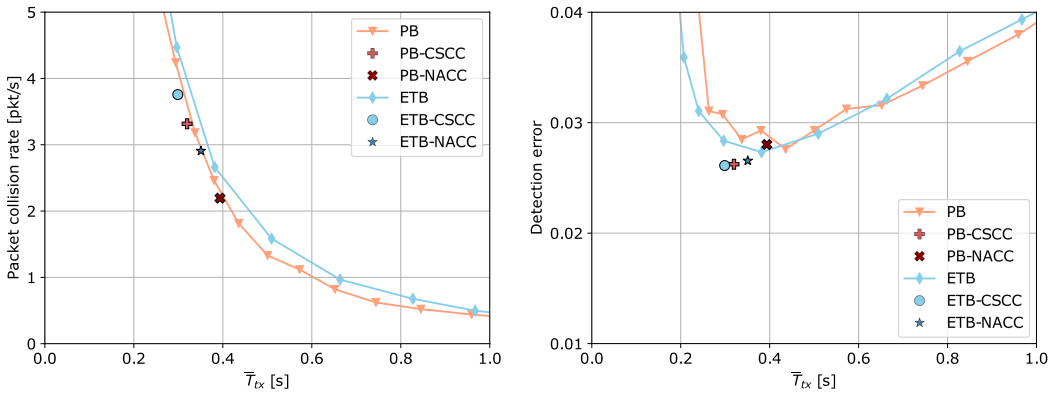


Fig. 2.9: Boxplot of the positioning error with $C = 8$.

The positioning error statistics of the four congestion control solutions are shown in Fig. 2.9, considering the boxplot representation. The box's edges represent the first and third quartiles of the error distribution, the box's center line is the error's median, and the whiskers show the 95% confidence intervals. We can appreciate that our solution is the only technique leading the third quartile below 0.7 m and the confidence interval below 1.0 m: this ensures that positioning estimates are not affected by local variations.

In Fig. 2.10 we show the packet collision rate and the detection error probability. The broadcasting strategies present almost identical trends when implemented without congestion control schemes. As we can observe from Fig. 2.10a, the amount of information that gets lost in the channel significantly increases when $\overline{T}_{\text{tx}} < 0.5$ s, independently of the deployed strategy. This phenomenon explains the degradation of the positioning estimation accuracy that we observe in Fig. 2.7a and Fig. 2.7b. Looking at Fig. 2.10a, the combination of the PB strategy with the NACC approach provides the lowest packet collision rate. This phenomenon occurs because



(a) Packet collision rate.

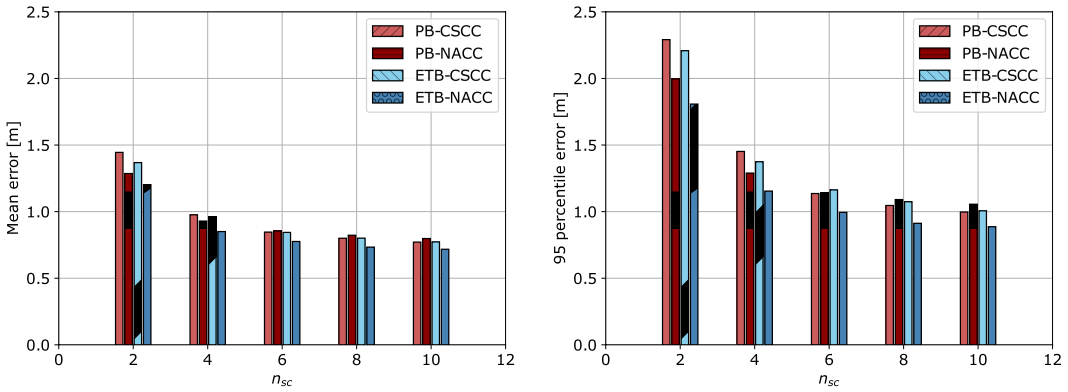
(b) Detection error probability.

Fig. 2.10: Collision and detection statistics with $C = 8$.

PB-NACC is the most conservative approach among those we analyzed, i.e., it minimizes the number of transmissions compared to the other communication schemes. On the other hand, PB-NACC exhibits larger positioning estimation errors, as shown in Fig. 2.9. We highlight that channel congestion does not affect only the positioning error but also the probability of misdetection and false alarm of a neighbor vehicle. Indeed, by looking at Fig. 2.10b, we can observe that the detection error probability quickly increases as soon as $\overline{T}_{\text{tx}} < 0.25$ s. All the schemes using congestion control lead to very similar detection error probabilities. Since the optimal working point is $\overline{T}_{\text{tx}} \approx 0.3$ we conclude that the minimization of the positioning error does not imply an increase in the misdetection and undetection events. Note that, because of the long simulation time, we could perform only 20 runs with each parameter so that the estimate of the detection error probability (which is a rare event) exhibits some statistical oscillations. Nonetheless, all the schemes with congestion control do not deviate from the curves defined by the previous simulations. As already mentioned, the combination of the PB strategy with the NACC approach presents a slightly higher $\overline{T}_{\text{tx}}$ and, therefore, is characterized by a different packet collision rate and detection error probability.

To validate these results in a more general scenario, we analyze the performance of the four possible congestion control schemes with different numbers of subcarriers C. The results of this analysis are reported in Fig. 2.11a and Fig. 2.11b. We observe that the solution combining the ETB strategy and the NACC approach outperforms the other schemes for any value of C, considering both the mean positioning error (in Fig. 2.11a) and the 95th percentile of the positioning error (in Fig. 2.11b). In particular, our solution outperforms state-of-the-art solutions by up to 20% mean error reduction and up to 30% 95th percentile error reduction. For what concerns the other techniques, we observe that the combination of ETB and CSCC performs poorly for $C \leq 4$, while it leads to better results when the number of subcarriers is higher. Conversely, the combination of PB and NACC performs well for $C \leq 4$ but does not fully exploit the available resources when $C \geq 6$.

Overall, we can provide some ideas for configuring optimal broadcasting strategies in vehicu-



(a) Mean positioning error.

(b) 95th percentile of the positioning error.

Fig. 2.11: Mean and 95th percentile of the positioning error as a function of C.

lar deployment. First, Figs. 2.7a and 2.7b demonstrate that the proposed ETB solution delivers better positioning accuracy with respect to a traditional PB approach since state updates are disseminated based on the estimated positioning error of each vehicle. Such an approach proved to reduce the temporal variance of the positioning error, thereby achieving more robust predictions. Second, Fig. 2.8a and Fig. 2.8b illustrate that the proposed NACC strategy outperforms the baseline CSCC strategy in terms of both average positioning error and variance. Such results motivate efforts toward the design of a congestion control scheme that exploits network topology information to regulate the periodicity of broadcasting opportunities. Finally, in Figs. 2.11a and 2.11b, we proved that increasing the number of subcarriers C has the benefit of reducing the channel access collisions due to the hidden node problem. As a consequence, the mean positioning error is reduced by more than 50% when C grows from 2 to 10. In general, among all the possible solutions, the combination of the proposed ETB strategy with the NACC approach ensures the best performance for any subcarrier configuration.

2.7 Conclusions and Future Work

In this chapter, we studied the trade-off between ensuring accurate position information and preventing congestion of the communication channel in vehicular networks. Particularly, we designed an innovative threshold-based broadcasting algorithm that forces vehicles to distribute state information if the estimated positioning error is above a certain error threshold. We also adopted a new congestion control mechanism that adapts the inter-transmission period according to network topology information. We showed through simulations that the proposed approach outperforms a conventional broadcasting strategy, which relies on the periodic transmission of state information and channel sensing since it reduces the positioning error with no additional resources.

As part of our future work, we will test our broadcasting and congestion control frameworks in more complex scenarios, e.g., by considering different road maps and traffic conditions. Be-

sides, we are interested in improving the tracking accuracy of our communication strategy by exploiting the ML paradigm, particularly Federated Learning (FL), as a tool to decide whether and when to transmit state information updates.

3

Remote Drone Tracking via 3D Mobility Models and LoRaWAN

3.1 Introduction

Over the last few years, Unmanned Aerial Vehicles (UAVs) have entered the mainstream, and the number of commercial drones will soon reach millions, according to the U.S. Federal Aviation Administration (FAA) [57]. Their integration in cellular networks, both as end-users and as coverage extension devices [58], is already being discussed. In particular, 5^{th} Generation (5G) and 6^{th} Generation (6G) systems will make use of UAVs of different sizes, from small-scale low-altitude drones to communication satellites [59], heralding the advent of an Internet of Flying Robots [60] in the near future.

Although energy and battery concerns are still critical [61], the use of UAVs can address a wide range of applications, including area surveillance [62], traffic jam detection [63], agriculture operations [64]. Besides, UAVs can be combined with ground-based robots to help them perform complex tasks [65], becoming essential in the Smart City context [66]. However, disaster management and relief is perhaps the most interesting UAVs' use case: drones can easily avoid ground-level obstacles and flooded areas by flying over them, surveying the extent of the damage [67] or helping with search and rescue operations [68] and communications.

As the capabilities of UAVs evolve towards the full support of safety-critical civilian applications as military battlefield operations [69], accurate drone positioning is becoming ever-more important. UAVs often have on-board Global Positioning System (GPS) receivers, and other sensors capturing images or videos [70] and can implement tracking algorithms to monitor their own trajectories [71]. As in vehicular networks, data fusion techniques can significantly improve

The work presented in this chapter was published in *IEEE Transaction on Wireless Communication* [56].

positioning accuracy by combining several measurements into a single and more robust solution. However, in order to track or control a swarm of several UAVs, mission control must be able to follow and even anticipate the drones' trajectories. This process requires the UAVs to frequently report their position to mission control, often with no available infrastructure and over several kilometers.

The accuracy required by positioning algorithms largely depends on the target application. Wildlife monitoring [72] usually deals with counting animals and can accept higher errors as long as the UAVs register images of the packs, flocks, or swarms. Instead, urban pollution monitoring [73], precision agriculture [74], and hydrogeological monitoring [75] can require an accuracy between 10 m and 50 m since they usually involve the reconstruction of 2D or 3D models from multiple images, and the positioning accuracy can determine the quality of the final reconstruction. Finally, formation control usually requires precision well below 10 m, and often even below 1 m [76]: in this case, centralized frameworks are inefficient for operating over several kilometers, and state-of-the-art solutions use distributed optimization with low-range and high-capacity communication technologies.

This chapter considers a scenario with a swarm of UAVs moving independently in a three-dimensional environment and periodically transmitting state information to a control station. Our target is to develop a tracking framework ensuring accurate position estimation, even in the case of occasional state updates. The proposed strategy exploits the Long Range Wide Area Network (LoRaWAN) communication standard, used by the UAVs to transmit short updates over long distances. This scheme allows for the deployment of drones over a wide area with minimal infrastructure, enabling their use in remote regions and disaster missions. However, the low bitrate and limited duty cycle imposed by the LoRaWAN specification in some areas (including Europe) strongly constrains the frequency of positioning updates.

To provide high positioning accuracy despite the LoRaWAN limitations, our framework relies on a Bayesian Filtering (BF) approach and deploys a Kalman Filter (KF) algorithm to track UAVs between updates. In particular, the tracking algorithm uses a novel 3D motion model, called 3-Dimensional CTRA (3D-CTRA), to represent the drone motion in the scenario. This model extends the well-known Constant Turn Rate and Acceleration (CTRA), widely used in vehicular scenarios, by adding a third dimension that allows it to represent even complex banking maneuvers. As a benchmark, we study two simpler models: Constant Speed (CS), which assumes a static heading for the target, and CTRA+, which considers linear motion on the vertical axis. To the best of our knowledge, 3D-CTRA is the first model that represents 3D maneuvers with the same flexibility that CTRA has in the 2D space. Independently on the motion considered, the tracking mechanism is the same: each UAV periodically transmits its state, including the orientation, speed, and acceleration, to a control station that estimates the drone's position. Similarly to the framework described in the first chapter, the control station can use the predictive step of a KF and monitor the overall systems even with sporadic updates from the UAV.

The target scenario presents additional challenges since LoRaWAN uses random channel access, which implies that updates from different UAVs can collide. We can mitigate the commu-

nication interference by exploiting the features of the LoRaWAN protocol as the orthogonality of different Spreading Factors (SFs) [77]. Setting a higher SF increases the communication range at the cost of a lower data rate, which makes the inter-transmission time intervals last several seconds, possibly reducing the tracking accuracy. Determining the optimal SF value is not trivial and strongly depends on the characteristics of the target scenario, including the channel status and the number of transmitters. To this end, we analyze the system performance under various conditions, e.g., when using a single channel with a bandwidth of 250 kHz instead of three 125 kHz channels, as well as with different Adaptive Data Rate (ADR) policies, supporting swarms of different sizes and in different scenarios.

The LoRaWAN technology was originally designed for massive scenarios with thousands of sensors transmitting infrequently. In this chapter, we study its implementation with fewer nodes transmitting at the maximum duty cycle, showing trade-offs never considered in the past literature. We analyze the performance of the proposed system by running extensive ns-3 simulations that exploit the UAV mobility traces from the Mid-Air public dataset. In doing so, we explore the optimization of tracking and communication, proving that the joint configuration of these two aspects can give significant performance gains, enabling accurate positioning over large swarms and long distances.

We organize the rest of this chapter as follows: Sec. 3.2 presents the state-of-the-art on UAV applications, focusing on the tracking frameworks used to estimate drone trajectories; Sec. 3.3 presents the CTRA+ and 3D-CTRA models, including the relative update equations; Sec. 3.4 describes the LoRaWAN standard and the frequency plan needed for our application; Sec. 3.5 describes the simulation settings and results for the different ADR solutions we implemented; finally, Sec. 3.6 presents our concluding remarks and ideas for future work.

3.2 Related Work

The target tracking problem is a well-studied research topic, usually solved by representing the target's motion with some linear models and estimating its state with a BF approach. The best-known BF algorithms used in this context are the KF [22] and the Particle Filter (PF) [24]. Despite these solutions being designed for applications with constant observation over the target, we can achieve long-term forecasting by simply applying the algorithm's predictive step to the last available state estimate. However, this solution does not provide high performance if the motion model is not sufficiently accurate.

As discussed in the first chapter, the tracking problem has been widely explored in 2D vehicular scenarios [1], often using the CTRA motion model [47], which considers the target to have constant acceleration and turn rate. The work of [78] presents a similar model for drones moving horizontally, also defining the Gaussian noise on the motion parameters. A more complex model with several possible maneuvers is in [79], whose authors adapt the CTRA settings to different possible trajectories. Most of the remaining literature works describe drone motion with classical CTRA or simpler models with constant speed [80] or orientation [81].

It is also possible to track UAVs passively, i.e., find their position without any communica-

tions, although at a much shorter range. To this goal, the authors of [82] design a visual system to track a UAV, providing high accuracy over a distance of up to 200 m. This method has no communication overhead but is constrained by the sensing capabilities of the visual system, and it only works if the UAVs are in the line of sight. It is also possible to use the degree of the arrival of external communication signals to allow the swarm to track the overall network topology [83]. In this case, if the UAVs share channel information, they can derive their relative positions like in an antenna array. However, this method only works for a high Signal to Noise Ratio (SNR) and inherently limits the area covered before the swarm loses formation control. We can perform the same operation using Terahertz communications [84], although only for very short distances (below 10 m in the paper).

Although LoRaWAN is a communication technology designed for Internet of Things (IoT) applications [85], we tested its performance in this specific use case to verify if it could adequately support control communication. To extend LoRaWAN capabilities in the scenario with multiple UAVs, we leveraged the ADR mechanism provided by the standard, as described in Sec. 3.4 and Sec. 3.5.1: several different implementations of this mechanism are in the literature. In [86], the authors propose a mechanism based on a water-filling algorithm: by adequately setting the modulation parameters, it is possible to equalize the packets' time on air, increasing the maximum network throughput. However, this solution would not benefit our application since low-bit-rate devices transmit their packets sporadically, leading to poor tracking performance.

To compensate for LoRaWAN limitations, we can deploy ADR techniques, as shown in many literature works, as [87, 88, 89]. In particular, the authors of [87] configure ADR depending on the estimated Received Signal Strength Indicator (RSSI), assuming full knowledge of mobility patterns, which are directly related to the value of the RSSI. Although this approach is useful for predictable trajectories, it does not apply to the case of drones exploring unseen areas or moving towards new objectives. In [89], the proposed ADR algorithms based on Gaussian, and Exponential Moving Average filters, are explored with ns-3 simulations in a scenario of static or slowly moving devices. Under such conditions, however, the devices transmit only one packet per hour, which would not be sufficient in a tracking application. Finally, the work in [88] presents empirical measurements of the performance of the ADR algorithm implemented by a drone network. In this analysis, devices are placed on trucks moving in the center of a city, which may be different from the UAVs use the case for the application requirements, the characteristic of the communication channel, and the mobility patterns.

In the near future, UAV networks will be an essential support for different applications and technologies introduced by Industry 4.0. However, the current literature lacks a tracking system designed specifically for drone swarms, especially for scenarios with critical communication constraints. This work goes beyond such limitations, defining a low-cost drone tracking solution based on an innovative 3D motion model a the LoRaWAN standards, which has never been implemented for this kind of application.

3.3 Tracking Model

In this section, we recall the CS and CTRA models, and we extend them to obtain the system equations for CTRA+ and 3D-CTRA. Besides, we describe the Unscented Kalman Filter (UKF) framework used by the control station to track the UAVs' trajectories. We resume the notation of our model in Tab. 3.1.

Tab. 3.1: Model parameters.

Parameter	Description	Parameter	Description
t	Timeslot index	τ	Timeslot duration
$\mathcal{N}$	UAV set	N	Swarm size
(x, y, z)	UAV position	(θ, ϕ, γ)	UAV orientation
R_{earth}	Earth radius	Δ_{earth}	Earth curvature error
$\hat{\mathbf{x}}_n$	Estimate state of n	$\mathbf{x}_n$	State of n
$\mathcal{F}(\cdot)$	System error function	$\mathcal{F}_n(\cdot)$	Error function of n
R	Measurement noise matrix	Q	Process noise matrix
η	Tilt reduction parameter	B	Communication bandwidth
d	Distance from the gateway	SF	Spreading Factor

3.3.1 Benchmark Motion Models

In the following, we refer to a UAV's *position* as the coordinates (x, y, z) representing its location in space using a Cartesian coordinate system. Instead, the UAV's *orientation* is represented by the three angles (θ, ϕ, γ), which measure the rotation with respect to a horizontal, North-facing position. While conventional systems only track the yaw, i.e., the angle θ between the drone's orientation and the reference direction on the horizontal plane, 3D motion models must also consider the pitch, i.e., the vertical angle ϕ between the drone's orientation and the horizon. Moreover, the target state must include the altitude z as the horizontal position (x, y), resulting in the 5-tuple (x, y, z, θ, ϕ). These parameters are common to all the motion models we implement.

We highlight that none of our models explicitly considers the orientation on the roll axis. While rotorcraft UAVs typically uses motion on the roll axis to turn, we can model the three-dimensional drone as a simple cylinder with no roll rotations, making the representation more compact. Naturally, this does not correctly track the precise orientation of the UAV, but does not affect the future drone position, for which rotations on the roll axis are irrelevant. Furthermore, the models do not consider the curvature of the Earth, but a perfectly horizontal plane, which results in an error Δ_{earth} when measuring the altitude of an object:

$$\Delta_{\text{earth}} = R_{\text{earth}} \left(1 - \cos\left(\frac{d}{R_{\text{earth}}}\right)\right), \tag{3.1}$$

where $R_{\text{earth}} = 6.371 \times 10^6$ m is the average radius of the Earth. This error is negligible for distances below $d = 1$ km, as $\Delta_{\text{earth}} = 7.8$ cm. As we expect the UAVs to cover distances much lower than 1 km between subsequent updates, which come at a rate of 1 Hz, this error can be compensated for by GPS measurements, which do account for the Earth's curvature,

and filtered out. If we were to consider much faster UAVs or less frequent sensor updates, the curvature of the Earth would become a significant factor.

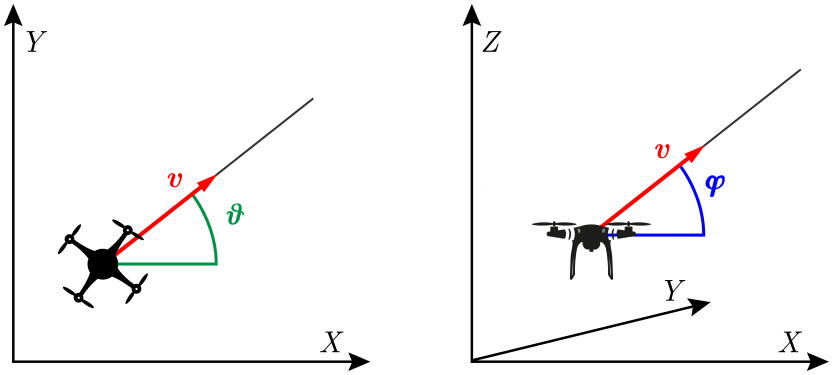


Fig. 3.1: The CS motion model.

The CS model assumes that the target moves with a constant speed v, without any change of direction (see Fig. 3.1). Hence, the turn rate $\omega = \frac{d\theta}{dt}$ and the tilt rate $\psi = \frac{d\phi}{dt}$ are zero, while the model state is given by

$$\mathbf{x}_{\mathrm{CS}}(t) = [x(t)\ y(t)\ z(t)\ \theta\ \phi\ v\ ]^T . \tag{3.2}$$

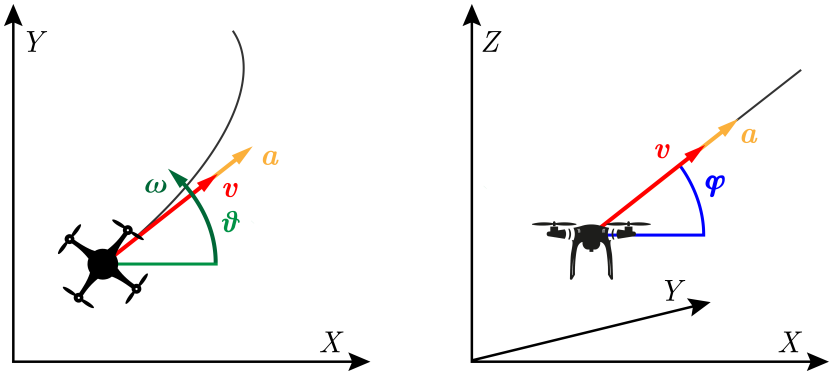


Fig. 3.2: The CTRA+ motion model.

The classical CTRA revises this hypothesis by assuming the target to have a constant tangential acceleration $a = \frac{dv}{dt}$ and a non-zero turn rate ω on a flat plane. The CTRA+ model makes the same assumption, but the plane on which the UAV moves is rotated by a constant pitch ϕ. As above, the tilt rate ψ is zero:

$$v(t) = v(0) + at; \tag{3.3}$$
$$\theta(t) = \theta(0) + \omega t; \tag{3.4}$$
$$\phi(t) = \phi(0), \tag{3.5}$$

where $v(0)$, $\theta(0)$ and $\phi(0)$ represent the initial velocity and orientation of the target. CTRA+ can be described by an Archimedean spiral [90] that evolves on a plane tilted by an angle ϕ with respect to the horizon. A sketch of the resulting model motion is given in Fig. 3.2.

To compute the equations of the CTRA+ model, we first need to separate the components of the target's velocity vector $\mathbf{v}(t)$:

$$v_x(t) = \frac{dx}{dt} = v(t)\cos(\theta(t))\cos(\phi(t)); \tag{3.6}$$

$$v_y(t) = \frac{dy}{dt} = v(t)\sin(\theta(t))\cos(\phi(t)); \tag{3.7}$$

$$v_z(t) = \frac{dz}{dt} = v(t)\sin(\phi(t)). \tag{3.8}$$

Therefore, we can compute the velocity module $v(t)$ as $v(t) = \sqrt{(v_x(t))^2 + (v_y(t))^2 + (v_z(t))^2}$, while the target position is given by the integral of the velocity components over time:

$$x(t) = x(0) + \int_0^t v(\tau)\cos(\theta(\tau))\cos(\phi)d\tau; \tag{3.9}$$

$$y(t) = y(0) + \int_0^t v(\tau)\sin(\theta(\tau))\cos(\phi)d\tau; \tag{3.10}$$

$$z(t) = z(0) + \int_0^t v(\tau)\sin(\phi)d\tau. \tag{3.11}$$

After solving the above equations, we get the following results:

$$x(t) = \cos(\phi)\left[a\left(\frac{\cos(\theta(t)) - \cos(\theta(0))}{\omega} + t\sin(\theta(t))\right)\right. \tag{3.12}$$

$$\left. + v(0)\left(\sin(\theta(t)) - \sin(\theta(0))\right)\right] + x(0); \tag{3.13}$$

$$y(t) = \cos(\phi)\left[a\left(\frac{\sin(\theta(t)) - \sin(\theta(0))}{\omega} - t\cos(\theta(t))\right)\right. \tag{3.14}$$

$$\left. - v(0)\left(\cos(\theta(t)) - \cos(\theta(0))\right)\right] + y(0); \tag{3.15}$$

$$z(t) = z(0) + \sin(\phi)\left(v(t)t - \frac{at^2}{2}\right). \tag{3.16}$$

We note that the procedure is equivalent to 2D glsctra [47] for the x and y components, except for the constant multiplying factor $\cos(\phi)$. Hence, the CTRA+ state is given by:

$$\mathbf{x}_{\text{CTRA+}}(t) = \left[x(t)\ y(t)\ z(t)\ \theta(t)\ \phi\ v(t)\ a\ \omega\right]^T, \tag{3.17}$$

which corresponds to the tuple representing the current attitude, with the addition of the velocity v, the acceleration a, and the turn rate ω.

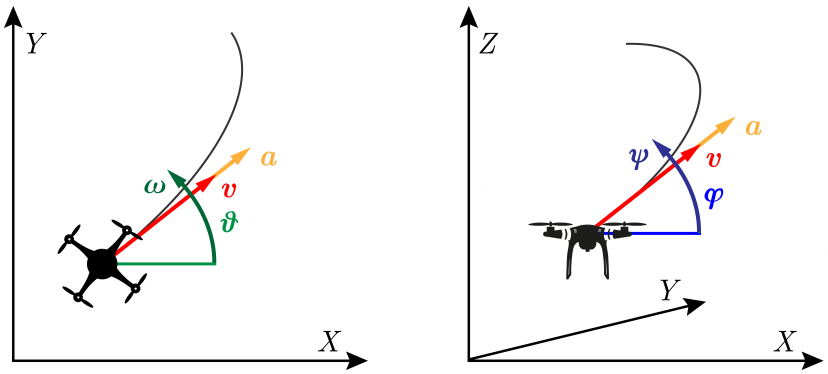


Fig. 3.3: The 3D-CTRA motion model.

3.3.2 The 3D-CTRA Model

The 3D-CTRA model extends the above description by adding a constant tilt rate $\psi = \frac{d\phi}{dt}$. Consequently, the target's movement is represented as the combination of two independent spiraling motions on the horizontal and vertical planes, forming a curved helix. We represent the resulting trajectory in Fig. 3.3. While the evolution of $\theta(t)$ still follows (3.4), the pitch is given by:

$$\phi(t) = \phi(0) + \psi t. \tag{3.18}$$

This complicates the derivation of the motion equations considerably since $\phi(t)$ is now time-dependent. For the sake of simplicity, we report the procedure only for $x(t)$, which is given by the integral of $v_x(t)$ over time. Applying the Werner formula, we obtain

$$x(t) = x(0) + \int_0^t \frac{v(\tau)}{2}(\cos(\theta(\tau) + \phi(\tau)) + \cos(\theta(\tau) - \phi(\tau)))d\tau, \tag{3.19}$$

which can be solved in closed form. The derivations for $y(t)$ and $z(t)$ follow the same steps; the final results are given in (3.20)-(3.22), where we used the compact notation $[F(x)]_a^b = F(b) - F(a)$ to indicate that the primitive function $F(x)$ should be evaluated at the extremes a and b.

$$\begin{aligned} x(t) = \Bigg[& \frac{v(\tau)}{2}\left(\frac{\sin(\theta(\tau)+\phi(\tau))}{(\omega+\psi)} + \frac{\sin(\theta(\tau)-\phi(\tau))}{(\omega-\psi)}\right) \\ & + \frac{a}{2}\left(\frac{\cos(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)^2} + \frac{\cos(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)^2}\right)\Bigg]_0^t + x(0); \end{aligned} \tag{3.20}$$

$$\begin{aligned} y(t) = \Bigg[& -v(\tau)\left(\frac{\cos(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)} + \frac{\cos(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)}\right) \\ & + \frac{a\sin(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)^2} + \frac{a\sin(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)^2}\Bigg]_0^t + y(0); \end{aligned} \tag{3.21}$$

$$z(t) = z(0) + \left[- v(\tau)\frac{\cos(\phi(\tau))}{\psi} + a\frac{\sin(\phi(\tau))}{\psi^2} \right]_0^t. \tag{3.22}$$

The equations (3.3), (3.4), (3.18) and (3.20)-(3.22) define the full non-linear version of 3D-CTRA. We observe that 3D-CTRA model equations are well-defined with some conditions on the tilt ψ: it must be different from 0 and the turn rate ω or its inverse. In those cases, the equations need to be derived separately in order to arrive at a valid mathematical result. First, when $\psi = 0$, the model is equivalent to CTRA+ and the value of $z(t)$ is given by (3.16). Then, when $\omega = \psi$, i.e., the rotations on the two axes have the same period, the values of $x(t)$ and $y(t)$ become:

$$\begin{aligned} x(t) = \Bigg[& \frac{v(\tau)\sin(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)} + \frac{a\cos(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)^2} \\ & + \frac{2v(\tau)-a\tau}{4}\tau\cos(\theta(\tau)-\phi(\tau)) \Bigg]_0^t + x(0); \end{aligned} \tag{3.23}$$

$$\begin{aligned} y(t) = \Bigg[& \frac{a\sin(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)^2} - \frac{v(\tau)\cos(\theta(\tau)+\phi(\tau))}{2(\omega+\psi)} \\ & + \frac{2v(\tau)-a\tau}{4}\tau\sin(\theta(\tau)-\phi(\tau)) \Bigg]_0^t + y(0). \end{aligned} \tag{3.24}$$

The case in which $\omega = -\psi$ produces a similar result, with switched terms:

$$\begin{aligned} x(t) = \Bigg[& \frac{v(\tau)\sin(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)} + \frac{a\cos(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)^2} \\ & + \frac{2v(\tau)-a\tau}{4}\tau\cos(\theta(\tau)+\phi(\tau)) \Bigg]_0^t + x(0); \end{aligned} \tag{3.25}$$

$$\begin{aligned} y(t) = \Bigg[& \frac{a\sin(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)^2} - \frac{v(\tau)\cos(\theta(\tau)-\phi(\tau))}{2(\omega-\psi)} \\ & + \frac{2v(\tau)-a\tau}{4}\tau\sin(\theta(\tau)+\phi(\tau)) \Bigg]_0^t + y(0). \end{aligned} \tag{3.26}$$

Along with the standard equations, the above results complete the model, which is valid for all values of the relevant parameters. Hence, the 3D-CTRA state is given by:

$$\mathbf{x}_{\text{3D-CTRA}}(t) = [x(t)\ y(t)\ z(t)\ \theta(t)\ \phi(t)\ v(t)\ a\ \omega\ \psi]^T, \tag{3.27}$$

which is equivalent to the CTRA+ state as defined by (3.17), with the addition of the tilt rate ψ.

We remind the reader that 3D-CTRA considers constant values for both ω and ψ. This does not reflect the real behavior of an aircraft, as dives and climbs are usually relatively short.

To make the model more realistic, the tracking system multiplies the value of ψ by a factor $\eta \in [0, 1]$ after every prediction step. In other words, the model implicitly assumes that the drone will gradually reduce its tilt rate and stabilize its pitch until it receives an explicit update from the UAV.

3.3.3 Tracking Framework

In our model, each UAV implements a UKF algorithm to monitor its current and future position, as explained in Appendix A. In particular, we assume that the time is discretized into slots $t \in \mathbb{Z}^+$ of τ seconds and that the UAVs are equipped with onboard sensors providing a noisy measurement at each slot. The UKF uses sensor data to estimate the UAV trajectory, assuming it follows any of the motion models $f \in \{\text{CS, CTRA+, 3D-CTRA}\}$. The control station deploys an additional UKF for each of the UAVs and aims to estimate the motion of the overall swarm. To this end, the control station has no local sensors but exchanges information with the UAVs, which periodically disseminate their state estimates.

Similarly to what was done by the ego vehicle in the previous chapter, the control station exploits the *predictive step* of the local UKFs to propagate the state updates, thus predicting the UAV trajectories. In particular, we denote by $\mathbf{x}_n(t)$ the state of an UAV $n \in \mathcal{N}$ at slot t, and by $\hat{\mathbf{x}}_n(t)$ the estimate of $\mathbf{x}_n(t)$ performed by the control station during the same slot. The system performance is given by the tracking error at the control station, computed as

$$\mathcal{F}(t) = \frac{1}{N}\left(\sum_{n\in\mathcal{N}} \mathcal{F}_n(\hat{\mathbf{x}}_n(t), \mathbf{x}_n(t))\right), \tag{3.28}$$

where $\mathcal{N}$ is the set of UAVs, $N = |\mathcal{N}|$ is the swarm size and $\mathcal{F}_n(\hat{\mathbf{x}}_n(t), \mathbf{x}_n(t))$ is the mathematical distance between the actual and the estimated positions of n:

$$\mathcal{F}_n(\hat{\mathbf{x}}_n(t), \mathbf{x}_n(t)) = \sqrt{(\hat{x}_n(t) - x_n(t))^2 + (\hat{y}_n(t) - y_n(t))^2 + (\hat{z}_n(t) - z_n(t))^2} \tag{3.29}$$

As in the previous chapter, we adopt a periodic state broadcasting: after a randomized time at the beginning of the simulation, each UAV sends the estimate of its own state to the control station with a constant inter-transmission period. We add a delay between consecutive transmissions in order to mitigate the interference suffered by UAVs whose communications happen to be synchronized. After it receives an update, the control station updates its UKF with the new information and exploits the predictive step to forecast the UAV's trajectory. Naturally, the errors will compound, causing long-term predictions to become less and less accurate until the next update. To enable the UAV to transmit the UKF parameters even at great distances, we considered the LoRaWAN technology [91], whose details are explained in the following section.

3.4 Communication Model

LoRaWAN is a low power wide area network technology leveraging the LoRa modulation at the PHY layer, which is based on a proprietary chirp spread spectrum technique to transmit over long distances [91]. We can tune the modulation through the SF parameter, which takes values from 7 to 12 and allows to trade coverage range for data rate. Indeed, signals transmitted with higher SF require longer transmission times but are more robust to channel impairments and can reach farther distances, up to several kilometers, in open-air scenarios. Conversely, signals modulated with low SFs cover shorter distances but require a shorter time to be transmitted. Another important feature is the *quasi*-orthogonality between SFs, which allows the receiver to demodulate packets overlapping in both time and frequency, as their power difference respects some conditions [92]. At the Medium Access Control (MAC) protocol layer, LoRaWAN considers a star topology, with three kinds of devices: a Network Server (NSE), End Devices (EDs), and Gateways (GWs). The first is the central controller of the network, while the EDs are peripheral nodes collecting data and transmitting them through the LoRa modulation. Finally, the GWs act as relays between EDs and NSE, i.e., collecting messages from EDs and forwarding them to the NSE through a reliable IP connection, and vice versa.

In our model, we assume that each drone is equipped with a LoRaWAN Class A ED. This class of devices is designed to consume a minimum amount of energy, with nodes staying in sleep mode most of the time, transmitting when necessary, and waking up for the reception in two short windows after each transmission. This feature makes the energy consumption of the LoRaWAN system negligible for the UAVs, considering that even the lightest quadcopter requires 50 to 100 W to hover or fly at low speed when carrying no sensor payload. In our case, we can expect the engines to have a power consumption of tens or even hundreds of Watts, 3 to 4 orders of magnitude higher than the transmission power.

While most of the traffic is expected to be in the Uplink (UL) direction (i.e., from the ED to the NSE), the standard also defines confirmed messages, which require an Acknowledgment (ACK) from the NSE. The protocol also supports the ADR mechanism, through which the NSE can control the transmission parameters of the EDs. By setting the appropriate flag on the transmitted packets, EDs requires the NSE to control their SF and transmission power, according to a policy defined by the NSE. The various ADR algorithms in the literature differ for the metric they try to optimize, i.e., scalability, throughput, and achieved range. In Europe, LoRaWAN works in the unlicensed 868 MHz sub-band and follows duty cycle regulations. In particular, three 125 kHz channels are allocated to UL transmissions and must respect a duty cycle limitation of 1%. Another option considered in this work is to use the available frequency band to allocate a single 250 kHz channel. This scheme does not bring the benefits of frequency orthogonality but reduces the packet transmission time, which is preferable in the case of a system with a single drone.

As we saw in the previous chapter, good tracking performance is possible only when the packet generation (and transmission) frequency is very high. However, when combined with multiple transmitting devices, as in a swarm of drones, this rapidly increases the traffic injected

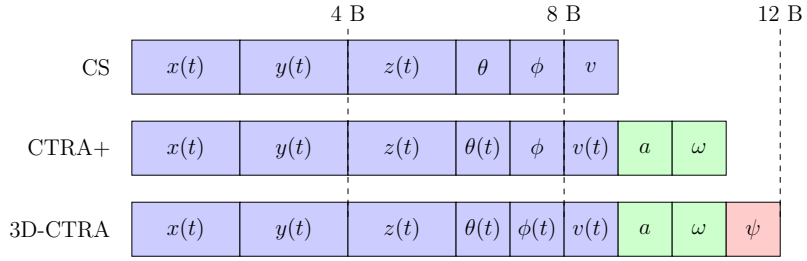


Fig. 3.4: Schematic of the payload format for the three tracking schemes.

into the communication channel, and the system performance suffers from interference. A drone scenario strongly differs from the most common LoRaWAN applications, which involve thousands of sensors transmitting sporadically and allows network designers to approximate the system as an infinite-user ALOHA system. Instead, our system considers a limited number of UAVs transmitting periodically at the maximum allowed duty cycle of 1%. This case is more similar to the finite-user case, which has been studied for slotted systems [93, 94]. What is interesting in our case is not just the throughput and delay [95], as in most previous studies, but also the Age of Information (AoI) [96], and it is not just the AoI that matters, but its effect on the tracking performance. Whether or not LoRaWAN is suitable to sustain this type of IoT application is unclear.

In this chapter, we show that, under certain conditions, LoRaWAN technologies can sustain both control and data traffic when using a correct configuration of the system parameters. A straightforward solution is to leverage orthogonality in the frequency domain (i.e., using three channels instead of a single one) and employing different SFs. However, these choices can also have side effects: the increased packet transmission time due to a narrower channel and the use of higher SFs will reduce the number of transmitted packets because of the duty cycle regulations. These trade-offs is part of the analysis presented in Sec. 3.5.

To partially contain the problem posed by the duty cycle limitation, we compress the system state to reduce the inter-transmission time and improve the tracking performance. In order to minimize the payload size, we can represent the position using 2 bytes, allowing movement in a square box with a size of 13 km while limiting the quantization error to 10 cm, significantly less than the average GPS error. Instead, we represent angles and turn rates using just 1 byte, with

Tab. 3.2: Packet transmission times.

SF	B [kHz]	Packet payload [B]	Transmission time [s]	Transmission interval [s]
7	125	9 11, 12	0.0515 0.0566	5.15 5.66
	250	9, 11 12	0.0257 0.0283	2.57 2.83
8	125	9 11, 12	0.0927 0.1029	9.26 10.29
	250	9 11, 12	0.0463 0.0514	4.63 5.14

a maximum error of 0.7 degrees. Since velocity and acceleration depend on the UAVs' flying capabilities, they can also be represented with just 1 byte, with a negligible loss of precision. Considering that the CS model requires the knowledge of the attitude 5-tuple and the velocity, its minimum payload size is 9 bytes. The CTRA+ state as given in (3.17) requires 11 bytes, and the 3D-CTRA state as given in (3.27) requires 12 bytes. The different payload formats are reported in Fig. 3.4, while the LoRa transmission times for packets with these lengths are reported in Tab. 3.2. We note that the duty cycle rules allow packets to be sent only sporadically, with a transmission period in the order of a few seconds. Transmission times are computed as in [97], considering an additional MAC header of 9 B.

3.5 Simulation Settings and Results

In this section, we describe the dataset providing the drone trajectories and the settings of our simulation scenarios. Hence, we present the results obtained while considering a single drone and an entire swarm, discussing how to configure the communication protocol in the two cases.

3.5.1 System Settings

Tab. 3.3: Simulation settings.

Parameter	Value	Description	Parameter	Value	Description
R_x	0.8274 m^2	Position accuracy along x	R_ϕ	0.0085 rad^2	Pitch accuracy
R_y	0.8274 m^2	Position accuracy along y	R_ω	0.0003 $(rad/s)^2$	Turn rate accuracy
R_z	3.7481 m^2	Position accuracy along z	R_ψ	0.0003 $(rad/s)^2$	Tilt rate accuracy
R_v	0.2500 $(m/s)^2$	Speed accuracy	R_a	0.1521 $(m/s)^2$	Acceleration accuracy
R_θ	0.0085 rad^2	Yaw accuracy	q	0.01	Process noise
τ	100 ms	Timeslot duration	η	0.9	Tilt reduction parameter
B	$\{125, 250\}$ KHz	Communication bandwidth	SF	$\{7, 8\}$	Spreading Factor
N	$\{1, 10, 30, 50\}$	Swarm size	d	$\{0, 1000, 3000\}$	GW distance

To simulate the UAV mobility, we exploit the Mid-Air dataset [98], which contains the flying records of a quadcopter moving in 3 different virtual environments. The dataset includes 54 trajectories of equal length for a total of 79 minutes of flight time. To strengthen our analysis, we randomly combined the different trajectories, thus increasing the duration and variability of the drone traces. The traces so obtained constitute the ground-truth motion of the UAVs, while we synthetically generated noisy data to represent the measurements of the drones' sensors. Each measurement included the position, the attitude, and the velocity and acceleration vectors of the UAV, combining information from GPS, accelerometers and gyroscopes.

According to the UKF parametrization, we model the process noise's covariance of the tracking framework by the matrix $Q = qI$, where I represents the identity matrix and $q = 0.01$. Instead, we model the covariance of the measurement error with a diagonal matrix R, whose elements represent the accuracy of the various drone sensors. The noise matrices and the UKF parameters are reported in Tab. 3.3: the values in R were chosen according to [99, 52, 54]. We highlight that the UKF state dimension and update equations change according to the chosen

motion model. As already stated, the UKF at the control station estimates the target trajectory by exploiting only the predictive step. Hence, after the control station receives a state update, it replaces the filter state with the new information, and the estimation process starts again.

The scenario of interest was studied with the network simulator ns-3 using the `lorawan` module described in [100], with N drones moving in the space according to the mobility traces of [98]. We assume that the drones are equipped with a LoRaWAN interface, which transmits packets at the maximum frequency allowed by the duty cycle. These messages are collected by a GW and forwarded to the NSE. Unless otherwise stated, transmitted packets do not require acknowledgment, and the NSE does not control any communication parameters. We simulate the packet reception according to a log-distance propagation loss model, which also considers shadowing, and to the interference model, which determines the correct reception probability in the case of collisions [101]. Using the data received by the control station and the designed tracking framework, we can compute the system performance, as defined in (3.28).

In the rest of the section, we analyze the positioning error for different tracking and communication scenarios. In particular, we investigate our tracking scheme for different values of the SF and of the position of the GW, analyzing how the communication range affects the tracking performance. We remark that the UAVs move from the initial point, so the distance from the GW to each drone changes over an episode. In particular, changing the position of the GW can give a qualitative idea of the range over which the drones moves.

We always initialize EDs to use the lowest SF setting that allows reliable communications within at least a range d from the GW in the absence of interference. However, each UAV may change its SF according to the considered ADR algorithm. We implemented the following ADR strategies:

- In the *NO ADR* setting, EDs do not set the ADR bit in their UL packets, and implement no autonomous data rate adaptation scheme. When a UAV travels out of the coverage zone, its packets are not received until it comes back in range.

- In the *NS ADR* setting, all decisions regarding changes in the SF of an UAV are taken by the NSE (and conveyed to the device by the GW). In this case, the NSE leverages one of the two receive windows opened by the device after each UL transmission to send a Downlink (DL) packet containing the new SF value to use. The NSE increments the SF when it senses that the UAV is leaving the coverage zone for the current received power of the last communication. Instead, it decrements the UAV's SF when the received power is above receiver sensitivity by a certain margin (set at 3 dB in this work). Note that the NSE sends control messages sent only if a change in the SF is required, otherwise no DL transmission is performed.

- Finally, the *ED+NS ADR* setting has the UAV require confirmation of correct reception for every UL packet it sends. If no ACK is received, the UAV increases the SF on its own. Similarly to what happens in the NS ADR setting, when the UAV comes back in range, the NSE instructs the drone to decrement the SF through a DL communication.

3.5.2 Performance Analysis

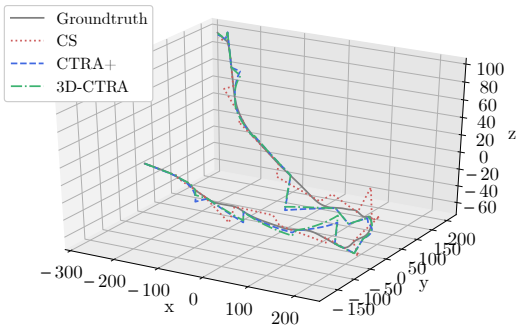


Fig. 3.5: 3D path of a single UAV with $d = 1000$ m, SF 7, and $B = 250$ kHz.

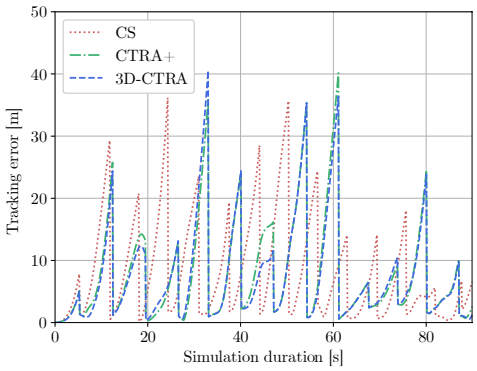


Fig. 3.6: Tracking error of a single UAV with $d = 1000$ m, SF 7, and $B = 250$ kHz.

First, we consider the 90 s drone path shown in Fig. 3.5. In this scenario, we consider the GW to be positioned in $(0, 1000, 0)$, with an initial distance $d = 1000$ m from the drone. The same figure includes the paths estimated by the control station using the CS and 3D-CTRA motion models, considering a communication setup with SF 7 and $B = 250$ kHz. Comparing the different paths, we observe how the CS scheme cannot follow the target, while 3D-CTRA ensures smaller deviations from the actual trajectory. The sharp changes in the estimated paths are due to updates transmitted bu the UAV: if the model estimates the wrong attitude, the predictive model at the control station gradually diverges from the real path. When the control station receives the next packet from the UAV, it resets the state of its filter, resulting in a "jump" in the estimated path. We can better analyze this aspect in Fig. 3.6, which shows the control station tracking error over time for all the considered models: the updates are clearly visible as the tracking error instantaneously drops. We highlight that the error of the CS model rapidly increases every time the drone performs non-linear movements, e.g., at time $t \simeq 20$ s. Instead, the error of CTRA+ and 3D-CTRA presents a smoother trend, with fewer and lower

peaks, even though both models require the transmission of larger packets. The performance of CTRA+ is almost identical to that of 3D-CTRA, but the latter better addresses vertical drone movements (e.g., at time $t \simeq 40$ s).

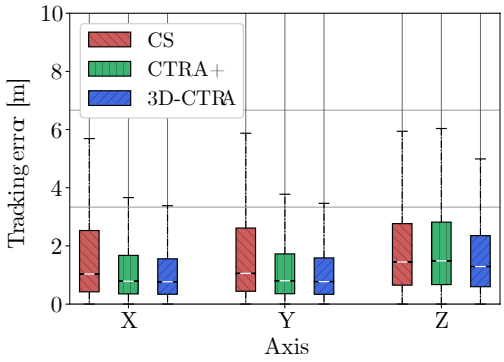


Fig. 3.7: Positioning error with $d = 1000$ m, $N = 1$, SF 7, and $B = 250$ kHz.

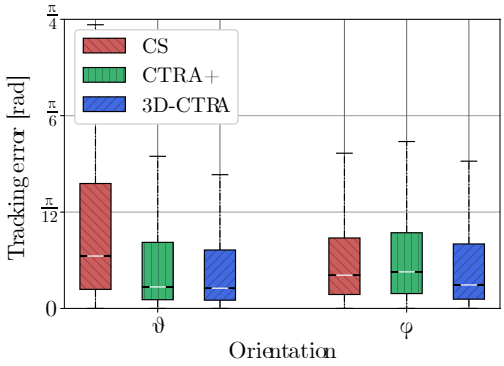
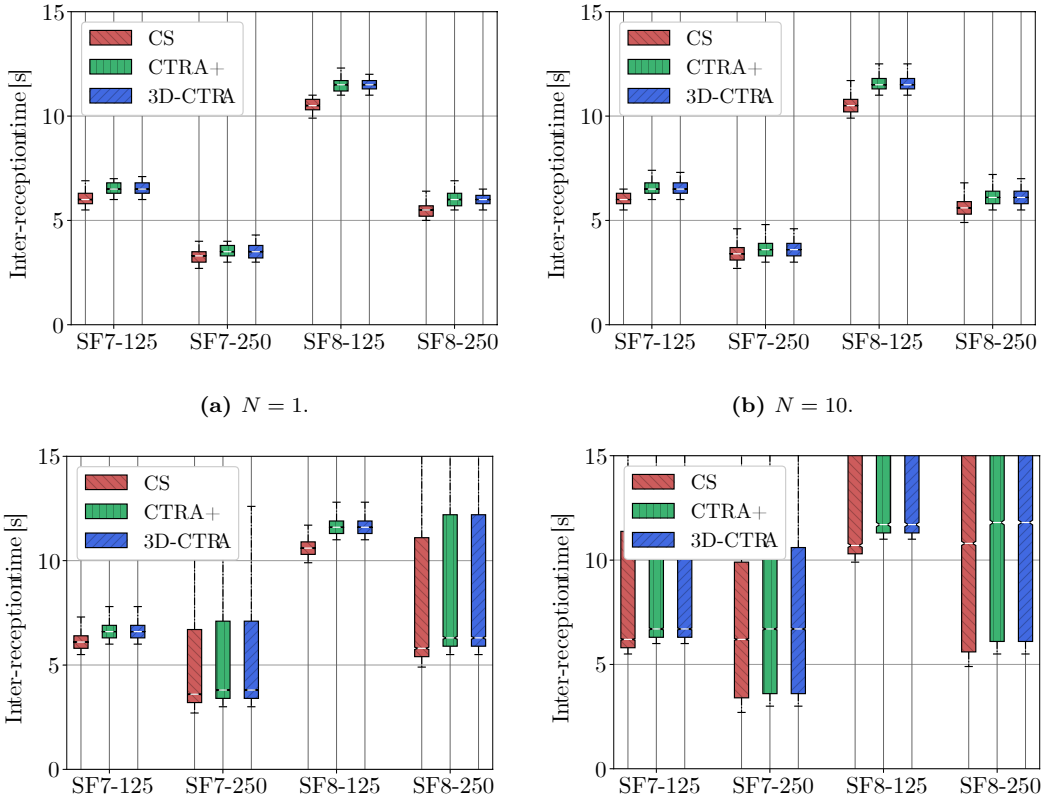


Fig. 3.8: Orientation error with $d = 1000$ m, $N = 1$, SF 7, and $B = 250$ kHz.

In what follows, we evaluate the performance of our system in tracking a single drone starting from a distance $d = 1000$ m from the GW. In particular, we examine the cumulative results over multiple virtual trajectories for a total of 2.5 hours of flight time. Fig. 3.7 shows the distribution of the position error along the three axes with SF 7, and $B = 250$ kHz. In particular, we adopt the boxplot representation, where the white line at the center of the box is the median of the distribution, and the box edges are the 25th and the 75th percentile, while the box whiskers represent the 5th and the 95th percentile, respectively. When considering the X and Y axes, the CS model is outperformed by both CTRA+ and 3D-CTRA, which ensure a richer representation of the drone's movements. In particular, 3D-CTRA shows a slightly lower position error with respect to CTRA+, which uses less information than 3D-CTRA to estimate the drone state. When considering the vertical Z axis, the error of CS is similar to that obtained with CTRA+

since both models consider that the target maintains a constant tilt. On this axis, 3D-CTRA performs best because it is the only one that can accurately track the drone maneuvers in all directions. We have a confirmation of such results in Fig. 3.8, which shows the tracking error on the UAV orientation. It is easy to see that the CS model cannot correctly track the drone orientation on the horizontal plane, as it assumes it will never change. On the other hand, CTRA+ and 3D-CTRA have similar performance, but 3D-CTRA manages to track the orientation better on the vertical plane, as it can represent the movement of the UAV more accurately.



(a) $N = 1$.

(b) $N = 10$.

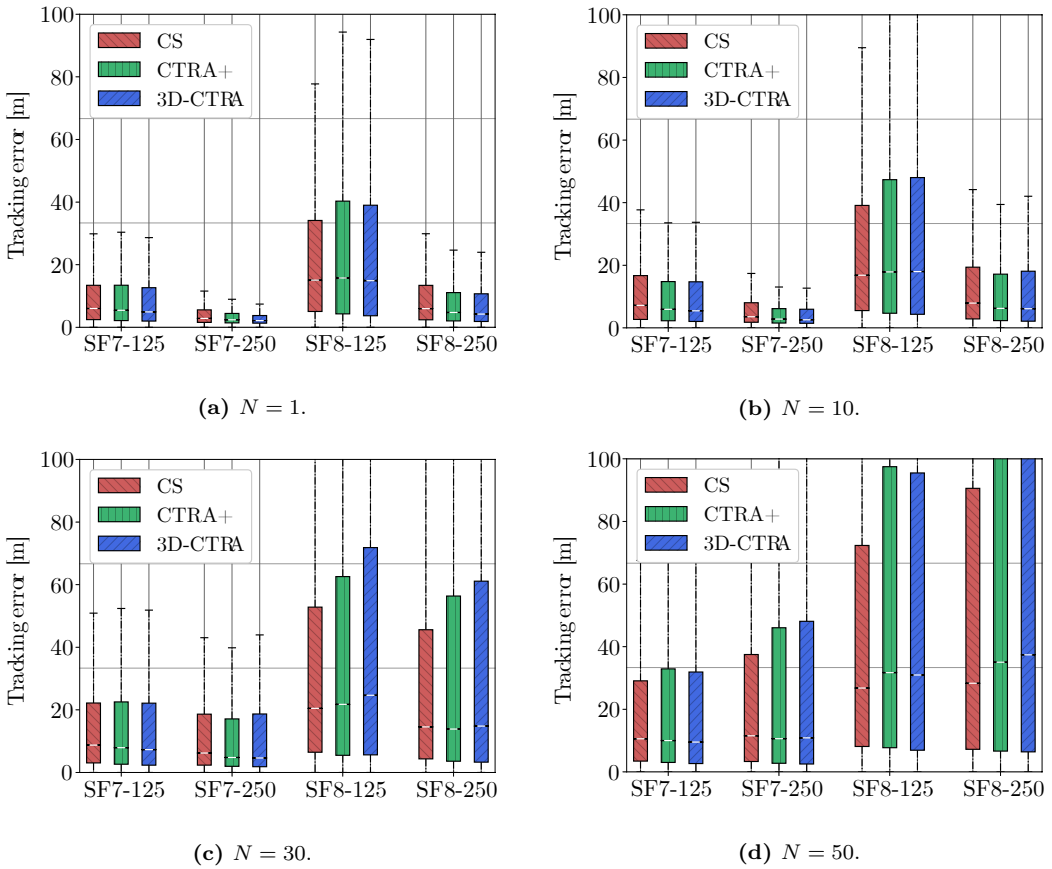
(c) $N = 30$.

(d) $N = 50$.

Fig. 3.9: Inter-reception time with $d = 1000$ m.

Fig. 3.9 shows the distribution of the inter-reception time for a swarm of drones starting 1 km away from the control station, with different tracking systems and communication settings. This parameter almost corresponds to the Peak AoI (PAoI), i.e., the maximum value reached by the AoI before a new update is received, as the transmission delay is much smaller than the inter-reception time. We observe that CS has the lowest inter-reception time since it exploits packets with only six variables. On the other hand, CTRA+ and 3D-CTRA need to transmit more data over the channel and are characterized by similar communication statistics. Furthermore, the

size of the swarm is critical to determining the inter-reception time, as collisions can multiply the time between successful packet receptions. We can easily see that increasing the swarm size from $N = 1$ to $N = 10$ has almost no effect, while a system with $N = 30$ is significantly different. The choice of the bandwidth is also important, as using $B = 250$ kHz allows UAVs to transmit data faster but also reduces the number of orthogonal channels from 3 to 1, increasing the number of drones competing for the same resources. Instead, the increase in the inter-reception time for systems with $N = 30$ and $B = 125$ kHz is imperceptible. The average inter-reception times are often lower for $B = 250$ kHz, as the packet frequency is doubled than the standard case. However, the worst-case performance is much degraded for $N = 30$ and $N = 50$, as shown by the upper halves of the boxes.



(a) $N = 1$.

(b) $N = 10$.

(c) $N = 30$.

(d) $N = 50$.

Fig. 3.10: Tracking error with $d = 1000$ m.

In general, reducing the PAoI corresponds to more frequent UKF updates at the control station, thus improving the tracking performance. However, the relationship between PAoI and tracking error is not linear. Fig. 3.10 shows the tracking error distributions in the same scenario: in this case, choosing $B = 250$ kHz provides a better performance even for $N = 30$, but not for $N = 50$. Interestingly, we can see that 3D-CTRA outperforms the other models when the inter-

reception time is low but CS becomes the best whenever the average inter-reception time goes over 10 seconds. This result is due to the non-linear nature of CTRA+ and 3D-CTRA, which can accurately track the UAVs' movements over a short timespan but suffer from the accumulation of errors over longer intervals. Even without considering turns and maneuvers, a small error in the estimated acceleration can lead to a quadratic increase in the tracking error over time, while a constant speed model has no acceleration and thus will have a somewhat bounded error. Under these conditions, the error for all the tracking models increases significantly and might be too high for some applications.

3.5.3 Spreading Factor Optimization

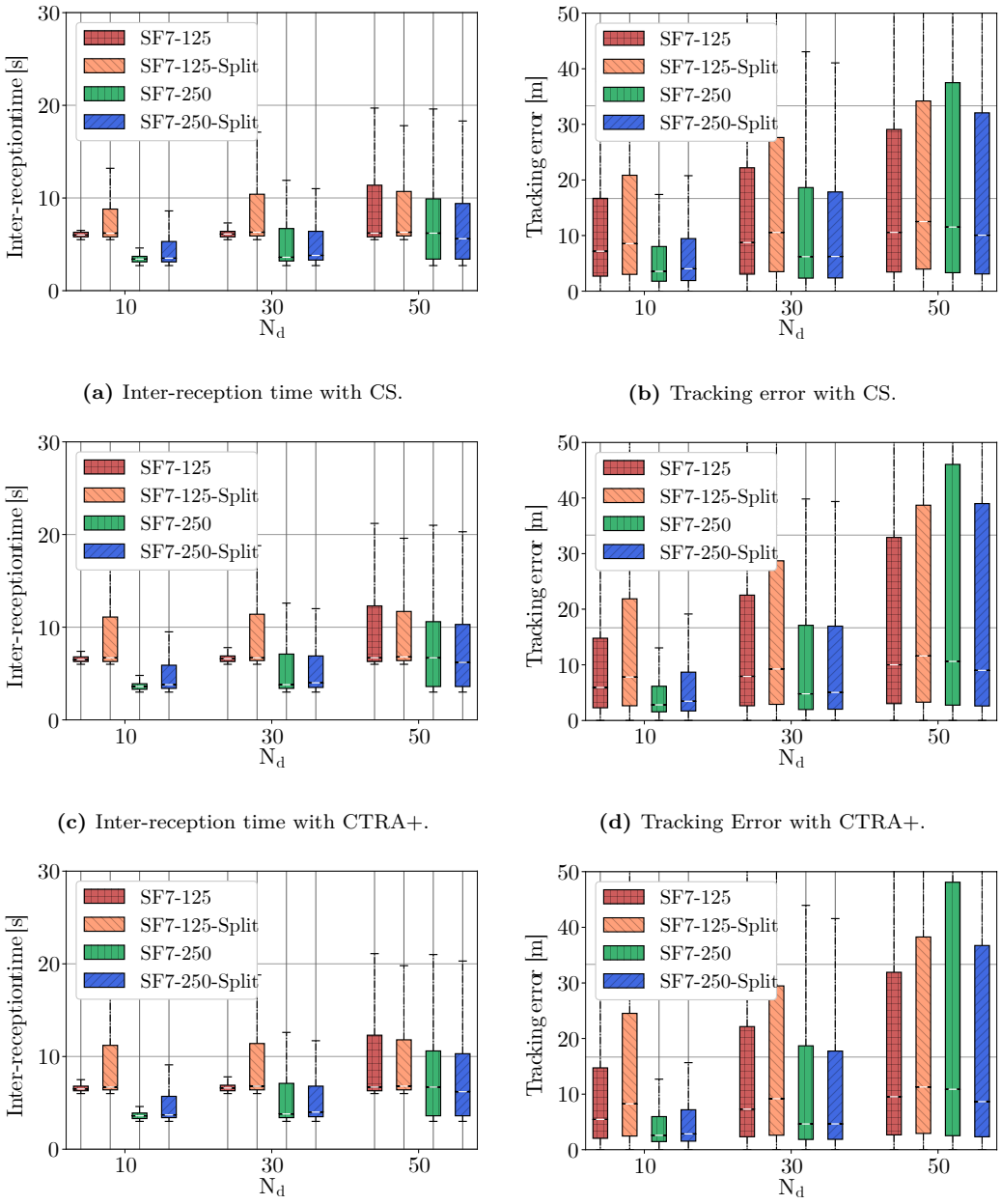
As we discussed above, interference and packet collisions can be significant issues in large swarms. To overcome such a problem and improve tracking performance, we can tune the SF parameter of the LoRaWAN protocol. We recall that packets sent with different SFs are orthogonal and can be received contemporary, providing additional protection from interference. On the other hand, higher SFs can increase the packet transmission time, leading to far longer inter-reception times to respect the duty cycle constraint.

Since inter-reception times of over 10 s severely degrade the tracking performance, we can limit the examined SFs to 7 and 8, i.e., the lowest two. In this case, the optimal choice to reduce congestion would be to have half of the UAVs use SF 7, while the other half use SF 8. This configuration would lead the UAVs using the lower setting to have a significant advantage in terms of data rate, while any collision for SF 8 packets would lead to a very long inter-reception time for the affected drones. In order to balance the two sets, we define a *split* system in which two-thirds of the drones set the SF to 7, while the others use a SF equal to 8.

Fig. 3.11 shows that, for $N = 10$, splitting the UAVs between the two SFs is suboptimal, while it can provide a small performance bonus if $N = 30$ and $B = 250$ kHz. Interestingly, splitting the swarm is not a good choice for the system with $B = 125$ kHz even with a swarm of $N = 50$ drones. Indeed, reducing the bandwidth increases both the number of channels and the time required for transmission; instead, avoiding collisions is not worth the cost of further increasing the inter-reception times for the UAVs with SF equal to 8. We also note that the SF optimization cannot solve the fundamental issue of the more complex models, as the updates are still not frequent enough to avoid divergence, and the CS model is still the optimal for $N = 50$, maintaining the 75th percentile of the tracking error below 30 m.

However, SF adaptation algorithms do not only reduce collisions but also for their original purpose: extending the range of the LoRaWAN network. In Fig. 3.12, we analyze the results obtained using the ADR system we described in Sec. 3.4, for different values of N. If the ADR is not active (NO ADR), drones do not vary their communication settings regardless of the scenario conditions. Instead, the NS ADR and the ED+NS ADR systems allow UAVs to vary the SF according to the policies described in Sec. 3.5.1.

The figure clearly shows that ED+NS leads to a much higher inter-reception times and, consequently, higher error. This is caused by multiple factors: firstly, UAVs using ED+NS



(a) Inter-reception time with CS.

(b) Tracking error with CS.

(c) Inter-reception time with CTRA+.

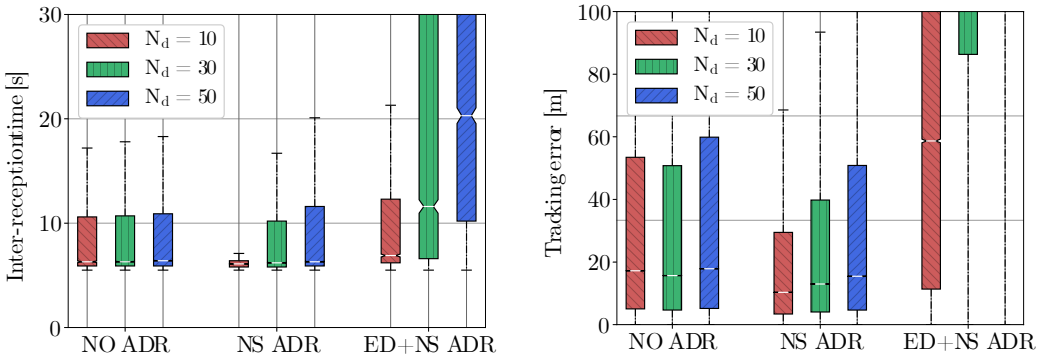
(d) Tracking Error with CTRA+.

(e) Inter-reception time with 3D-CTRA.

(f) Tracking error with 3D-CTRA.

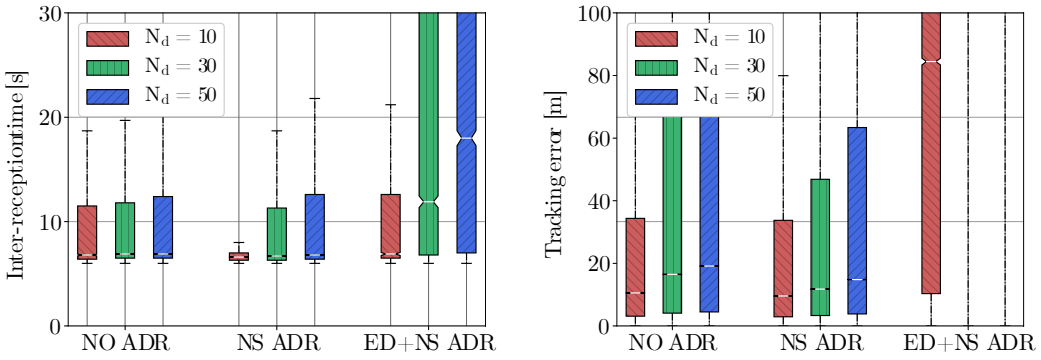
Fig. 3.11: Tracking error for different SF, with $d = 1000$ m.

ADR increases the SF as soon as the GW misses an ACK to the UL message. The fact that ACKs needs to employ the same SF as UL packets further exacerbates this problem, increasing the duty cycle consumption by the GW and further limiting the number of ACKs that can



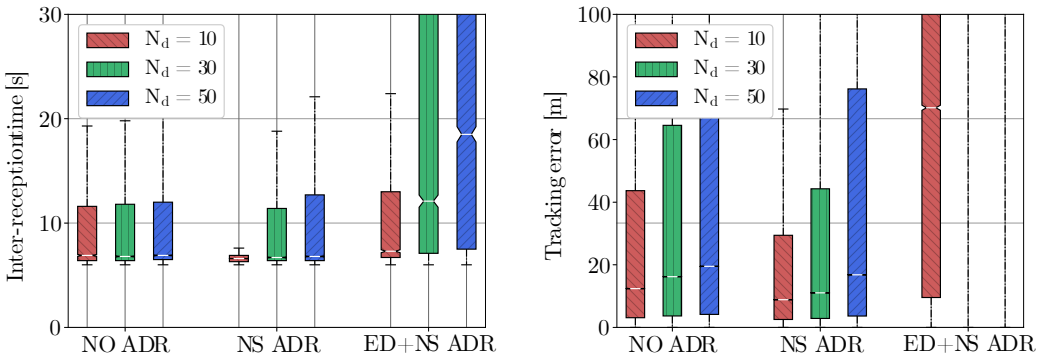
(a) Inter-reception time with CS.

(b) Tracking error with CS.



(c) Inter-reception time with CTRA+.

(d) Tracking error with CTRA+.



(e) Inter-reception time with 3D-CTRA.

(f) Tracking error with 3D-CTRA.

Fig. 3.12: Tracking error for different ADR with $d = 3000$ m.

be sent. As shown in Fig. 3.12, the good inter-reception time performance achieved by the NS ADR scheme results in a better tracking error, which makes the NS ADR scheme the best choice for the specific scenario considered. However, we might implement different ADR policies

to take into account available knowledge of the UAV's mobility patterns, ensuring a smarter configuration of the SF parameter.

3.5.4 Payload Optimization

We can now consider the consequences of piggybacking the tracking update on sensor communications. Naturally, LoRaWAN is a low-bitrate technology, so the sensor updates will have a limited size. We analyze sensor payloads of 16, 32, and 64 bytes, combined with the three tracking systems we have presented. We first observe the inter-reception time, which roughly corresponds to the PAoI for the sensor data, as the transmission takes up less than 1% of the total value. Fig. 3.13a-3.13b show a boxplot for a swarm with $N = 10$, using two different bandwidths, as above. Naturally, inter-reception time grows with the payload, but it is still below 10 s even with the 64 B payload when using $B = 250$ kHz.

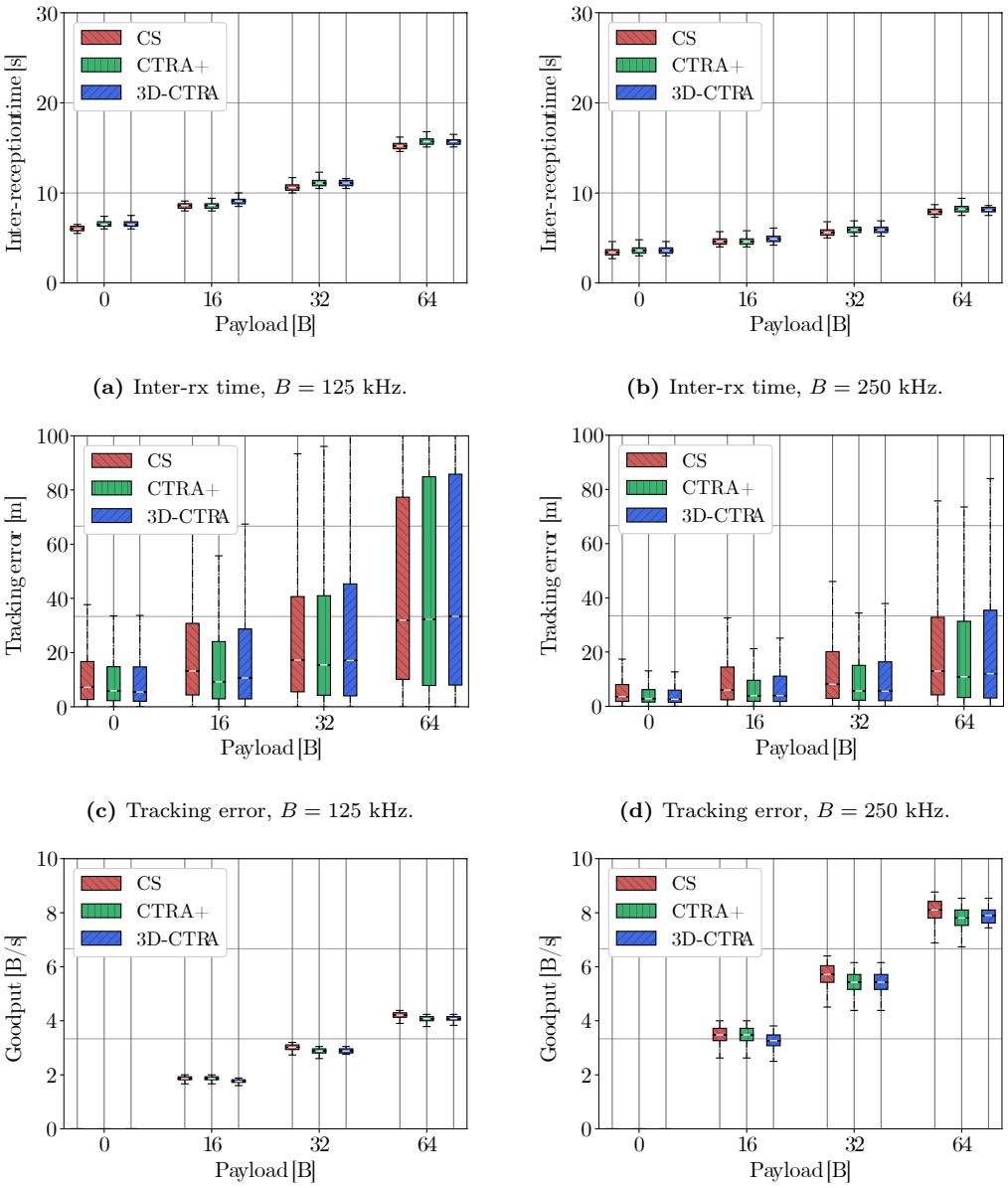
As we discussed above, this has an obvious effect on the tracking error: Fig. 3.13c-3.13d also show the tracking error as a function of the payload size. Focusing on the system with $B = 250$ kHz, the tracking error grows as the payload size increases but stays within an acceptable range. It is interesting to note that, while CS is always the best choice with a non-zero payload and $B = 125$ kHz, CTRA+ is the best choice for small payloads with $B = 250$ kHz, as the inter-reception times are within acceptable bounds to benefit from a more accurate model.

We can also consider the goodput generated for each payload size, which also increases as the payload size grows: Fig. 3.13e-3.13f show the received bytes per second for the sensor application, considering only the payload bits. There is a trade-off between goodput on one side and PAoI and tracking performance on the other: smaller payloads can be delivered more frequently, improving the tracking accuracy but reducing the overall goodput. We also analyzed the packet erasure probability, but we do not report it in the chapter, as it was similar for all payload sizes and tracking schemes, around 5% for $B = 125$ kHz and 15% for $B = 250$ kHz. This difference is due to the larger number of collisions that the drones experience with a single channel with respect to having three orthogonal ones with a lower capacity.

3.6 Conclusions and Future Work

In this chapter, we presented a tracking framework for UAVs, based on a novel 3D-CTRA mobility model and periodic transmissions over LoRaWAN. Our system can estimate drone trajectories with high accuracy even at long distances, and the proposed mobility models significantly outperform standard schemes as CS and CTRA. Despite LoRaWAN's duty cycle limitations, our simulations show that the technology is suited to manage swarms of dozens of drones, provided that an appropriate ADR scheme handles scenarios in which UAVs move out of the coverage area. The accuracy of our system can enable several applications, from urban pollution monitoring to precision agriculture, relatively tolerant to reduced positioning errors.

Several extensions of this work are possible. We can extened the proposed motion model with maneuver and mission-level information, further reducing the tracking error. Moreover, it



(a) Inter-rx time, $B = 125$ kHz.

(b) Inter-rx time, $B = 250$ kHz.

(c) Tracking error, $B = 125$ kHz.

(d) Tracking error, $B = 250$ kHz.

(e) Payload goodput, $B = 125$ kHz.

(f) Payload goodput, $B = 250$ kHz.

Fig. 3.13: Tracking error with $d = 1000$ m, $N = 10$, and SF 7.

will be interesting to explore features that are not part of the LoRaWAN standard up to now, like the use of a different frequency plan or of listen-before-talk instead of applying the duty cycle. Finally, the study of the behavior of swarms, and strategies to avoid packet collision, will enable new applications by improving the tracking accuracy at a low cost.

Part II

Multi-Agent Learning Strategies for Network Control

Introduction

Recent technological progress has promoted autonomous devices such as Unmanned Aerial Vehicles (UAVs) in several scenarios, including area surveillance and factory automation. In these contexts, drones and robots must place themselves in specific locations, exploiting wireless transmission to gain context awareness and build a cooperative perception of the overall system. However, the requirements of most innovative applications may make it hard to achieve efficient node coordination, especially if there is a lack of network resources. To tackle this problem, we extend the dynamic network scenario analyzed in the first part of this work, focusing not on mapping node trajectories but on determining the actions of mobile nodes. In particular, we adopt a goal-oriented approach, where transmission decisions aim to optimize a physical task, and the control strategy is adjusted depending on the communication process.

In the following chapters, we exploit the Deep Reinforcement Learning (DRL) paradigm and define a distributed architecture to govern a group of autonomous nodes that exchange information to coordinate their actions in an implicit fashion. Initially, we consider a basic scenario with a pre-determined communication policy, where a Unmanned Aerial Vehicle (UAV) swarm explores a limited area and identifies locations of interest. Our experiments show that the proposed architecture yields effective strategies that demonstrate robustness to communication impairments and can adapt to different scenarios by taking advantage of the Transfer Learning (TL) paradigm. In particular, our system can overcome a computationally expensive look-ahead algorithm, which takes decisions after estimating the performance of each possible sequence of physical actions.

Then, we address the challenge of combining goal-oriented communication and networked control into a single model, which we call Cyber-Physical Partially Observable Markov Decision Process (CP-POMDP). The proposed model provides high flexibility and can represent any application involving the cooperation between sensors observing an environment and agents taking action over it. To show the benefits of our model, we design a simple reference scenario where an underwater vehicle has to achieve specific missions with the support of a set of buoys. We evaluate the performance of our strategy against multiple benchmark algorithms that exploit a pre-determined transmission policy. Our results prove that joint training of communication and control systems can significantly improve dynamic networks' functionalities, especially in the case of constrained communication resources.

4

Distributed Reinforcement Learning for Drone Swarm Control

4.1 Introduction

The high data rate of modern wireless communications and the increasing computational power of embedded systems, along with the sharp price reduction of commercial Unmanned Aerial Vehicles (UAVs), have encouraged the use of swarms of drones for Smart City services [103]. Thanks to their size, flexibility, and flight ability, drone swarms represent a new solution for many different applications, such as remote surveillance, distributed sensing, wireless coverage extension, and object tracking [104]. Over the past few years, researchers have studied several UAV-based systems [105], addressing the high complexity of swarm coordination and control. In this context, the optimal control strategy depends on the target scenario, which may involve different applications such as surveillance, monitoring, mapping, and tracking [106]. On the other hand, all the above tasks share similarities and require UAVs to move within an area and position themselves over specific locations.

In this chapter, we focus on a scenario where a drone swarm has to localize and reach a set of static targets that occupy random positions in an unexplored area. We consider the UAVs equipped with sensors allowing them to detect targets within a limited range and a radio interface that makes it possible to exchange position information and sensing data. Hence, the UAVs needs to coordinate to find the targets without colliding with each other or possible obstacles. Notably, the problem of identifying fixed targets arises in several practical situations, ranging from the generation of real-time flood maps [107] to the detailed tracking of weeds in agriculture [108]. Besides, an efficient map exploration is of interest even for larger classes of

The work presented in this chapter was published in *IEEE Transactions on Cognitive Communications and Networking* [102].

problems, e.g., considering moving targets or, in general, a dynamic environment. One such example is wildfire monitoring in dry regions [109], which can be effective as long as the UAVs move faster than the spread of the fires.

The nature of our problem, in which actions can have long-term consequences and affect the future evolution of the environment, makes it a suitable application for Reinforcement Learning (RL) [110]. This latter is a powerful mathematical paradigm that makes it possible to compute the optimal strategy to manage a general environment modeled as an Markov Decision Process (MDP). At the same time, the curse of dimensionality [111] makes it difficult to adopt a centralized approach, where a single controller manages all the drones in the network. To address the above issue and design a more scalable system, we can adopt a Multi-Agent Reinforcement Learning (MARL) approach, where many agents interact with the same environment. However, the simultaneous presence of multiple learning units makes the scenario non-stationarity, possibly complicating the system training and the estimation of the optimal policy [112]. In particular, the different degrees of coordination and communication between agents make the configuration of MARL systems a very challenging problem, arousing interest from many application areas.

Our analysis is based on a very general model, using a grid-world representation and making a limited number of assumptions about drone behavior. In particular, we assume that the initial map is not entirely visible and covered by obstacles preventing drone movements. On the other hand, the UAVs exchange information through a radio channel allowing them to achieve a cooperative perception, thus encouraging cooperation and avoiding collisions. The performance depends on the ability of the UAVs to reach the targets, whose positions are initially unknown and dependent on different distributions according to the drone application. To address the described problem, we implement a Networked Distributed Partially Observable Markov Decision Process (ND-POMDP) framework and associate each UAV with a distinct agent, with a partial vision of the overall network status. We compute the optimal control strategy by a Deep Reinforcement Learning (DRL) approach, approximating the policy of each drone with a Neural Network (NN) that determines its local actions. The main challenge is to adapt the drone policy to both the environment dynamics and the behavior of the different learning agents moving within the map.

We compare our learning-based strategy against a Look-Ahead (LA) heuristic, proving that the proposed system can better explore the environment and reach the targets faster. Besides, we test the potential of Transfer Learning (TL), showing that agents trained under specific conditions can adapt to a new scenario much faster than restarting the training from scratch. In doing so, we assess that significant environmental changes, such as varying the map size or drone number, do not compromise our approach. Finally, we show that the proposed system is robust to channel impairments and can handle scenarios where information exchange is possible only if drones are within a limited range, as it occurs in realistic communication settings. Notably, the high flexibility and generality of the learning framework make our setup extendable to even more complicated scenarios.

The rest of the chapter is organized as follows: first, Sec. 4.2 analyzes the related work

in the field; Sec. 4.3 presents the system model and MARL algorithm; Sec. 4.5 reports the experimental setup and the simulation results, including TL experiments; finally, Sec. 4.6 gives the conclusions and presents some possible avenues for future work.

4.2 Related Work

Multi-agent learning is arousing ever-more interest from the scientific community, and an extensive taxonomy of multi-agent solutions was presented in [113]. In order to address a MARL problem, there are four main approaches: (1) designing a single agent that interacts with multiple copies of itself, generating emergent behaviors; (2) implementing communication between agents of the same type to enable agent coordination; (3) defining a cooperation strategy between agents with different behaviors to achieve a common goal; (4) modeling other agents' behaviors and adapting the learned policy to them [114].

The authors in [115] study the first of these four approaches and use the tabular Q-learning algorithm to guide drones to survey an unknown area, showing that even the simplest MARL algorithm can improve the overall reward. Similarly, in [116], a MARL framework is applied to a more complex problem in which a UAV network provides flexible wireless communication. However, in these works, the goal of the MARL algorithm is to optimize resource allocation instead of guiding drones, so a coordinated exploration strategy is missing.

An interesting research direction for MARL is pioneered in [117], which uses NNs to represent and learn more complex Q-functions [118]. At first, the authors study the performance of one network trained for all agents, which then share the same parameters during the execution phase (this is also our approach). Then, the authors propose a differentiable inter-agent learning framework that allows the agents to learn the most meaningful messages to exchange and improve cooperation. This system is suitable for faster training but is limited to low numbers of agents.

Many other works implement RL strategies in the practical scenarios discussed above. In [107], the authors adopt a MARL approach to control a flood-finding swarm of UAVs. However, the model only considers a swarm with a fixed number of drones, and the experimental results are not compared to state-of-the-art heuristics. In [108], the authors consider an improved random walk model to map an agricultural area, solving the problems due to noisy acquisition through a collective perception of the environment. In particular, the possible random walks are biased according to the positions of the targets already discovered and the other network nodes. However, we observe that the model needs to be manually tuned for each setting despite the authors considering swarms of variable sizes,

Another recent study [109] considers wildfire spread monitoring, checking how the fire evolves and spreads on the map from a known starting point. The authors define the problem as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [119] and carry out several experiments, as well as comparisons against a greedy heuristic (similar to the look-head method we studied in this work). The work in [120] focuses on a target-tracking application for disaster scenarios, presenting a model similar to ours but applied to a single drone. Finally, [121]

considers a MARL system with realistic communication, where a swarm of drones needs to get data from an Internet of Things (IoT) sensor network. This problem is quite simplistic, as the position of the targets is known in advance, and the MARL framework only needs to optimize the trajectories.

Finally, MARL approaches also fit models in which UAV connectivity is important. In [122], a framework including RL and game theory enables to plan of the path of two drones that need to save energy and minimize the interference to the ground network while maintaining a cellular connection. Furthermore, the authors of [123] design a centralized RL system to maximize coverage for a swarm of aerial base stations serving mobile users on the ground. A similar approach is considered in [124], which redefines the problem in terms of Quality of Experience (QoE) maximization for the users. For a fuller communication-oriented perspective on the use of RL for UAV networks, we refer the reader to [125].

The above works have similar objectives to ours but either go back to the single-agent setting or have restrictive assumptions. As an example, [109] considers well-known fire patterns, which can be extensively learned, with a known starting point. In our case, the initial positions of the targets and the UAVs are not the same across different episodes, complicating the learning task but making the model more general. Furthermore, unlike previous efforts in the literature, this work exploits the TL paradigm, showing how our model can adapt to scenarios with obstacles, realistic maps, and different swarm sizes. To the best of our knowledge, our work presents the most realistic environment to date, providing a single architecture to address multiple operative scenarios.

4.3 System Model

In the following, we design the system model used to analyze the drone swarm management scenario. The model is based on a grid representation of the environment and makes it possible to represent several UAV applications. We give a list of the notations used in Table 4.1 as a reference to the reader.

4.3.1 Environment

We base the system model on a square grid map of size $M \times M$. Each cell is identified by its coordinates $\mathbf{m} \in \mathcal{M}$, where $\mathcal{M} = \{0, ..., M-1\} \times \{0, ..., M-1\}$. We assume that a swarm $\mathcal{N}$ of UAVs moves within the map to identify a set of target cells $\mathcal{K}$, representing the goal of the UAV application. We denote by $N = |\mathcal{N}|$ and $K = |\mathcal{K}|$ the number of UAVs and targets over the map, where $|\cdot|$ is the cardinality operator. In particular, we denote the position of drone $n \in \mathcal{N}$ and target $k \in \mathcal{K}$ by $\mathbf{x}_n = (x_n, y_n)$ and $\mathbf{z}_k = (x_k, y_k)$, respectively.

Following an RL approach, we discretize the time into slots $t \in \mathbb{Z}^+$, and we consider multiple episodes of $T \in \mathbb{N}$ slots each. At the beginning of each episode, we reinitialize the targets and UAVs positions. The performance for a single episode coincides with the UAVs capability of positioning itself over the targets as fast as possible. To this end, each drone can move by a

Tab. 4.1: Model parameters.

Parameter	Description	Parameter	Description
t	Slot index	T	Slots per episode
$\mathcal{M}$	Coordinate set	M	Cell per map edge
$\mathcal{N}$	UAV set	N	UAV number
$\mathcal{K}$	Target set	K	Target number
$\mathcal{W}$	Obstacle set	W	Obstacle number
$\mathcal{S}$	State space of ND-POMDP	$\mathcal{O}$	Observation space of ND-POMDP
$r_n(\cdot)$	Reward function for the nth UAV	Δ	Action space of ND-POMDP
$\mathbf{x}_n$	Coordinates of the n-th UAV	$\mathbf{X}$	Matrix of UAV positions
$\mathbf{z}_k$	Coordinates of the k-th target	σ	Standard deviation of the target distribution
$\phi(\cdot)$	Target value function	$\mathbf{\Phi}$	Matrix of target values
d_{sparse}	Minimum target distance	Ω	Cells occupied by obstacles
$\mathbf{h}_w$	Size of the w-th obstacle	$\mathbf{l}_w$	Coordinates of the w-th obstacle
$h_{\min}, h_{\max}$	Min and max obstacle size	$\mathcal{H}_w$	Cells occupied by the w-th obstacle
$\omega(\cdot)$	Obstacle location function	$\mathbf{\Omega}$	Matrix of obstacle positions
$\hat{\mathbf{\Phi}}$	Estimated matrix of target values	$\hat{\mathbf{\Omega}}$	Estimated matrix of obstacle positions
$\mathbf{a}_n$	Action of the nth UAV	$\mathcal{A}$	Action space of a single UAV
$\nu(\cdot)$	Invalid action function	$\chi_n(\cdot)$	Collision function for the nth UAV
θ	Penalty for forbidden actions	ψ	Penalty for collisions
F	Observation window size	ζ	UAV field of view
π	Learned policy	$Q(\cdot)$	Q-value function
$e(t)$	Experience sample at slot t	G_n	Cumulative return for the nth UAV
γ	Discount factor	α	Learning rate
B_{size}	Batch size	ρ	Observation tuning parameter

single cell at each slot t, taking the actions determined by a policy π. In particular, we denote by $\mathbf{x}_n(t)$ the position of $n \in \mathcal{N}$ at slot t and by $a_n(t)$ the action taken by n at the same slot. The future drone position $\mathbf{x}_n(t+1)$ depends on both $\mathbf{x}_n(t)$ and $a_n(t)$, as we will better explain in the next section.

We assume that a UAV has a limited Field of View (FoV), i.e., it can only perceive the map cells within a radius ζ. As the swarm explores the environment, each drone discovers new regions on the map and updates its knowledge, consequently. In the basic scenario, we assume that the drone knowledge is shared so that each drone receives the information gathered by the others immediately. We keep these settings constants during the algorithm training, whereas we also consider a scenario with unreliable communications during the testing phase.

When considering realistic communication, we may lose the broadcast messages sent by each UAV due to wireless channel impairments. To this end, we implement the path loss and shadowing model from [126], which is based on actual measurements from air-to-air communication, and a Rayleigh fading model with an error correction code. The physical size of the cells in the map is a critical parameter when the UAVs communicate directly with each other (and not through the network infrastructure on the ground). In particular, increasing the size of the cells impairs performance because of communication range issues. The proposed communication model has an error probability of 50% at approximately 110 m, corresponding to 11 cells, assuming 10 m of cell side, or to 5 cells, assuming 5 m of cell side.

To ensure the scalability of the approach, we assume that the UAV take decisions according to a sub-region of the map. In particular, we consider each UAV to have up-to-date knowledge only inside a $F \times F$ square, with $F \leq M$. The area center is given by the UAV location unless the distance between that cell and the map edges is lower than F. In the second case, the visible region ends at the map edges without modeling the area outside the environment. We

observe that F should not be confused with the FoV ζ, which is generally lower than F. The first represents the area that each UAV exploits to decide its actions; the latter represents the area that each UAV can sense directly.

4.3.2 Target Distribution

To model the targets, we generate K Gaussian distributions with the same covariance matrix $\Sigma = \left(\begin{smallmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{smallmatrix}\right)$. The distribution means coincide with the target coordinates, while the covariances represent the target visibility to the UAVs. In this way, a single distribution does not represent the actual target but rather the full view of the UAVs, which can see the target from afar. We can interpret σ as the distance that enables the target identification: if the value of σ increases, the target is visible from further away. Therefore, we associate each cell $\mathbf{m} \in \mathcal{M}$ with a value $\phi(\mathbf{m})$, which represents the proximity of target for such a location. The cell value is given by the maximum of the Gaussian functions at that point, normalized in such a way that the target locations have values equal to 1:

$$\phi(\mathbf{m}) = \max_{k \in \mathcal{K}} e^{-\frac{1}{2}\left((\mathbf{m}-\mathbf{z}_k)^{\mathbf{T}}\Sigma^{-1}(\mathbf{m}-\mathbf{z}_k)\right)}. \tag{4.1}$$

Under these conditions, the most valuable cells coincide with Gaussian means, representing the target locations.

In this work, we consider two different distributions for the targets, named *sparse* and *cluster*. In both cases, we randomly place the first target on the grid following a 2D uniform distribution, while the others are placed sequentially according to the considered scenario. Hence, the coordinates of the first target can take any value in $\mathcal{M}$ with equal probability. In the sparse scenario, the position $\mathbf{z}_k$ of the k-th target is randomly chosen from the set $\mathcal{M}_k^{\text{sparse}} = \{\mathbf{m} \in \mathcal{M} : ||\mathbf{m} - \mathbf{z}_j||_2 > d_{\text{sparse}},\ \forall j < k\}$, with probability mass distribution

$$P_{\text{sparse}}(\mathbf{z}_k = \mathbf{m}) = \frac{||\mathbf{m} - \mathbf{z}_0||_2}{\sum_{\mathbf{m} \in \mathcal{M}_k^{\text{sparse}}} ||\mathbf{m} - \mathbf{z}_0||_2}. \tag{4.2}$$

In this way, the targets tend to be distributed far from the first, with a minimum distance d_{sparse} between each other.

In the cluster scenario, instead, the position $\mathbf{z}_k$ of the k-th target is randomly chosen from the set $\mathcal{M}_k^{\text{cluster}} = \{\mathbf{m} \in \mathcal{M} : ||\mathbf{m} - \mathbf{z}_j||_2 > 1,\ \forall j < k\}$, with probability mass distribution

$$P_{\text{cluster}}(\mathbf{z}_k = \mathbf{m}) = \frac{1}{(1 + ||\mathbf{m} - \mathbf{z}_0||_2) \sum_{\mathbf{m} \in \mathcal{M}_k^{\text{cluster}}} \frac{1}{(1+||\mathbf{m}-\mathbf{z}_0||_2)}}. \tag{4.3}$$

In this case, the targets tend to cluster around the first but cannot occupy adjacent cells since the minimum distance must be greater than 1.

We provide an example of the two target displacements in Fig. 4.1. We observe that such distributions represent two plausible configurations of targets for different UAV applications. In wildlife monitoring, some species of animals might tend to herd together, while more territorial

ones will have a sparser distribution on the map. Instead, in a battlefield scenario, groups of soldiers act together as a tight formation, while guerrilla-style fighting will involve a sparser distribution of forces.

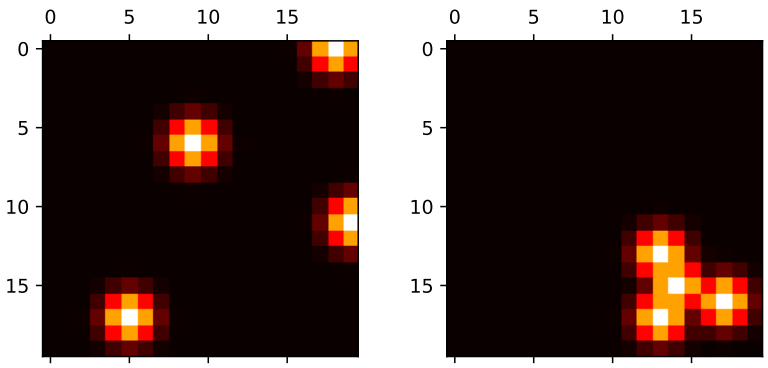


Fig. 4.1: Sparse (left) and cluster (right) target distributions.

4.3.3 Obstacle Distribution

To represent more complex scenarios, we randomly deploy a set of W *obstacles* over the grid map. In an urban area, the obstacles might be tall buildings or designated no-fly zones; in a wild environment, they might correspond to boulders or tall trees. To this end, we define a function $\omega(\mathbf{m})$, which is equal to 1 if $\mathbf{m}$ corresponds to an obstacle and 0 otherwise. We describe each $w \in \mathcal{W}$ by two parameters: the dimension $\mathbf{l}_w$ and corner position $\mathbf{h}_w$. We formally define the wth obstacle as the set

$$\mathcal{H}_w = \big\{(x,y) \in \mathcal{M} : x \in \{\mathbf{h}_{w,1}, \dots, \mathbf{h}_{w,1} + \mathbf{l}_{w,1} - 1\}, y \in \{\mathbf{h}_{w,2}, \dots, \mathbf{h}_{w,2} + \mathbf{l}_{w,2} - 1\}\big\}. \quad (4.4)$$

Therefore, we can define Ω as the union of all the obstacles:

$$\Omega = \bigcup_{w \in \mathcal{W}} \mathcal{H}_w. \quad (4.5)$$

. Hence, we encode the values of Ω in the $M \times M$ matrix $\mathbf{\Omega}$, defined as:

$$\mathbf{\Omega}(\mathbf{m}) = \begin{cases} 1 & \text{if } \omega(\mathbf{m}) = 1; \\ 0 & \text{otherwise.} \end{cases} \quad (4.6)$$

We generate obstacles sequentially, like the targets. For each obstacle $w \in \mathcal{W}$ the dimensions $\mathbf{l}_w$ are drawn uniformly from the set $\{l^{\min}, l^{\max}\} \times \{l^{\min}, l^{\max}\}$. Instead, we pick the corner

position $\mathbf{h}_w$ from a uniform distribution in the set

$$\big\{\mathbf{h} \in \mathcal{M} : \mathcal{H}_w \subset \mathcal{M}, ||\mathbf{m}, \mathbf{z}_k||_2 > 1, \forall \mathbf{m} \in \mathcal{H}_w, k \in \mathcal{K}, d(\mathcal{H}_w, \mathcal{H}_q) \geq 2, \forall q < w\big\}, \tag{4.7}$$

where $d(\mathcal{H}_w, \mathcal{H}_q) = \min_{\mathbf{m}_w \in \mathcal{H}_w, \mathbf{m}_q \in \mathcal{H}_q} ||\mathbf{m}_w - \mathbf{m}_q||_2$ is the distance between $w \in \mathcal{W}$ and $q \in \mathcal{W}$. These constraints ensure that each obstacle is entirely inside the map, not directly adjacent to any targets, and not touching any other obstacles. In particular, our design guarantees that the UAVs can reach the targets from any point on the map. The overall framework allows us to represent many applications, varying the map size, the drone and target numbers, the obstacle shapes, the FoV range ζ, and the target visibility σ.

4.4 Learning Model

In this section, we design a learning framework to coordinate the UAV in our system model. We follow a RL approach, describing the scenario as a ND-POMDP [127], and determining the optimal strategy by using the *double Q-learning* algorithm. We report more details about the RL theory and its methods in Appendix B.

4.4.1 ND-POMDP Formulation

The ND-POMDP framework makes it possible to describe a MDP where the system state is not directly observable and influenced by the actions of multiple agents without central coordination. This approach suits our scenario, where the drone swarm only has limited knowledge and each single UAV takes actions independently. We observe that ND-POMDP is a particular class of Dec-POMDP for which not all agents interact with each other [128]. Convergence to the optimal solution for this kind of problem has been proven for classical reinforcement methods [129], although not for deep models. Hence, as in most works in the literature, we use a benchmark to evaluate the performance of our scheme.

Formally, an ND-POMDP is identified by a 5-tuple, composed of a state space $\mathcal{S}$, an agent space $\mathcal{N}$, a joint action space $\Delta = \mathcal{A}^N$, an observation space $\mathcal{O}$, and a reward map $r : \mathcal{S} \times \Delta \to \mathbb{R}^N$, where $N = |\mathcal{N}|$. In our scenario, the complete system state s is constituted by five matrices: $\mathbf{X}$, $\mathbf{\Phi}$, $\mathbf{\Omega}$, $\hat{\mathbf{\Phi}}$, and $\hat{\mathbf{\Omega}}$. In particular, $\mathbf{X}$ is a $F \times F$ matrix $\mathbf{X}$, whose the elements of $\mathbf{X}$ are set to 1 in the locations occupied by the drones and to 0 otherwise. Instead, $\mathbf{\Phi}$ and $\mathbf{\Omega}$ are two $F \times F$ matrices representing the targets and obstacle displacement. We compute the values of their elements according with the functions $\phi(\mathbf{m})$ and $\omega(\mathbf{m})$ described in the previous section.

The remaining matrices represent the UAV knowledge of the environment. In particular, the elements of $\hat{\mathbf{\Phi}}$ are equal to $\phi(\mathbf{m})$ if $\mathbf{m}$ has been explored and 0 otherwise. Instead, the elements of $\hat{\mathbf{\Omega}}$ are equal to $\omega(\mathbf{m})$ if $\mathbf{m}$ has been explored and 0 otherwise. At the beginning of each episode, the UAVs do not know the actual target and obstacles displacements. Before exploring the map, the drones initialize the cells of $\hat{\mathbf{\Phi}}$ to 1 and those of $\hat{\mathbf{\Omega}}$ to 0. The observation $o_n \in \mathcal{O}$ of $n \in \mathcal{N}$ is given by $\mathbf{X}_n$, $\hat{\mathbf{\Phi}}_n$, and $\hat{\mathbf{\Omega}}_n$, defined as the $F \times F$ subsets of $\mathbf{X}$, $\hat{\mathbf{\Phi}}$ and $\hat{\mathbf{\Omega}}$ centered in $\mathbf{x}_n$.

At each slot t, each UAV can either stay over the same cell or move to one of the four adjacent cells. At the same time, obstacles are impassable in our environment definition, and the UAVs cannot move outside the map. The action space of each UAV is defined as $\mathcal{A} = \{(0,0), (0,1), (1,0), (0,-1), (-1,0)\}N$. The joint action for the swarm is a vector $\mathbf{a} \in \Delta$, which contains the individual UAVs' actions, denoted as $\mathbf{a}_n$ for drone $n \in \mathcal{N}$. We define the function $\nu(\mathbf{x}_n, \mathbf{a}_n)$, which returns 1 if the action is valid (i.e., it does not lead the UAV to fly outside the map or into an obstacle) and zero otherwise:

$$\nu(\mathbf{x}_n, \mathbf{a}_n) = \begin{cases} 1, & \text{if } \mathbf{x}_n + \mathbf{a}_n \in \mathcal{M} \wedge \omega(\mathbf{x}_n + \mathbf{a}_n) = 0; \\ 0, & \text{otherwise.} \end{cases} \tag{4.8}$$

Therefore, we can update the position of each drone in the following way:

$$\mathbf{x}_n(t+1) = \mathbf{x}_n(t) + \mathbf{a}_n(t)\nu(\mathbf{x}_n(t), \mathbf{a}_n(t)). \tag{4.9}$$

Fig. 4.2 shows an example of the system state at the beginning and in an advanced stage of an episode. In this case, we have two drones and four targets in a 20×20 map with no obstacles. The left map shows the drones' positions (in yellow), the observed map is in the center, and the actual map is on the right. In the figure, we associate darker cells with lower values and brighter cells with higher values. It is easy to see how the swarm gains knowledge as the drones explore the map and look for targets. Hence, the UAVs found two targets relatively quickly, and a significant portion of the grid remained unexplored.

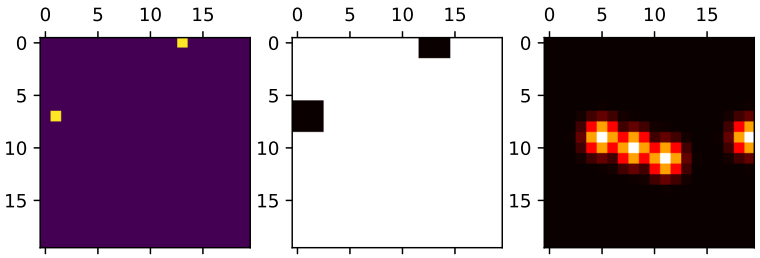
In the example, we set $F = M = 20$, i.e., making drones take decisions according to the overall map. If the communication range equals or exceeds the map side, i.e., $F \geq M$, the observed state o for all UAVs corresponds to the left and center maps. On the contrary, if $F < M$, the observation for each UAV would include a different map region.

According to our RL approach, we need to define a reward function to encourage drones toward their goal. The reward depends on the system state s and the joint action vector $\mathbf{a}$, representing the overall swarm motion. In our scenario, each UAV receives a reward equal to 1 if it is directly above a target, $-\theta$ if it tries to go outside the map or to cross an obstacle, $-\psi$ if it is in the same cell as another drone, and 0 in any other case. The parameter θ makes the drone avoid erroneous actions that would take them outside the map or crash into obstacles. Instead, the value of ψ affects the distance that drones keep from each other: if ψ is low, the drones get close to each other if the targets are very close. Naturally, if there is a collision risk when the drones are in the same cell, the value of ψ should be high.

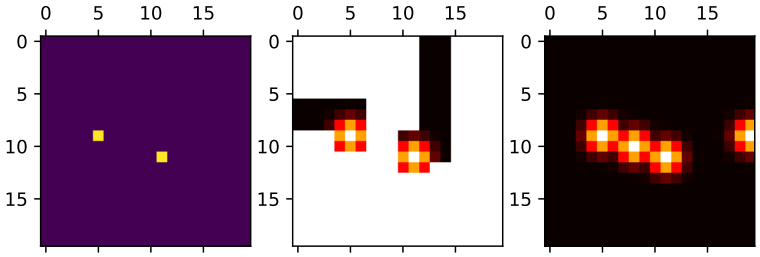
Given the position $\mathbf{x}_n$ and the action $\mathbf{a}_n$ of $n \in \mathcal{N}$, we define the function $\chi_n(\mathbf{X}, \mathbf{A})$ as

$$\chi_n(\mathbf{X}, \mathbf{A}) = \max_{m \in (\mathcal{N} \setminus n)} \delta\big(\mathbf{x}_n + \mathbf{a}_n(t)\nu(\mathbf{x}_n, \mathbf{a}_n) - \mathbf{x}_m - \mathbf{a}_m(t)\nu(\mathbf{x}_m, \mathbf{a}_m)\big). \tag{4.10}$$

where $\delta(\mathbf{m})$ takes value 1 if $\mathbf{m} = (0,0)$, and zero otherwise. Practically, $\chi_n(\mathbf{X}, \mathbf{A})$ indicates if two drones collide, returning 1 if one or more drones move to the same cell as drone u, and 0



(a) Episode beginning.



(b) Episode ending.

Fig. 4.2: Drone positions (left), known (center) and real map (right).

otherwise. Hence, given the state s and the joint action $\mathbf{a}$, we define the reward for $n \in \mathcal{N}$ as:

$$r_n(s, \mathbf{A}) = -\theta(1 - \nu(\mathbf{x}_n, \mathbf{a}_n) - \psi\chi_n(\mathbf{X}, \mathbf{A}) + (1 - \chi_n(\mathbf{X}, \mathbf{A})) \sum_{k \in \mathcal{K}} \delta(\mathbf{x}_n + \mathbf{a}_n - \mathbf{z}_k). \quad (4.11)$$

We observe that our model involves deterministic state transitions and system observations, which are affected only by the agents' decisions.

4.4.2 Distributed Deep Q-Learning

To deal with the described Partially Observable Markov Decision Process (POMDP) problem, we implement a Distributed Deep Q-Learning (DDQL) framework, defining a Q-value function $Q(\cdot)$ for each agent in the system. As explained in the Appendix, uch a function computes an estimate of the discounted return $G_n(t) = \sum_{\tau=0}^{\infty} \gamma^\tau r_n(t + \tau)$ achieved by following the learned policy π. In particular, the policy π determines the probability for each drone $n \in \mathcal{N}$ to take an action $\mathbf{a}_n \in \mathcal{A}$ given an observation $o_n \in \mathcal{O}$. Hence, we aim to determine the function $Q(\cdot)$ associated with the optimal policy π^*, maximizing the expected G_n. For the sake of readability, in the following, we omit the n subscript to indicate the agent whenever possible.

In our system, we approximate $Q(\cdot)$ by a NN that takes as input the last observation $o(t)$

and returns the quality of the possible drone actions, i.e., $Q(o(t), a)$, $\forall a \in \{(0,0),\ (0,1),\ (1,0),\ (0,-1),\ (-1,0)\}$. We maintain a single Deep Q-Network (DQN) during the training phase, updating the learning architecture with the experience of all the system agents. Therefore, we have a centralized learning framework that returns a single policy shared among multiple learning units. As it would be impossible for a single UAV to experience even just a non-negligible fraction of possible states, using centralized training is critical in the system's generalization ability. In other scenarios, following centralized training may not be convenient, especially when the agents have to learn different tasks.

To train our architecture, we follow the approach from [118] and leverage a *replay memory* to store the agent experience $e(t) = (o(t), a(t), r(t), o(t+1))$. At each training slot, the agent picks a batch of B_{size} elements from the replay memory. This process allows us to separate the algorithm training from the experience acquisition, avoiding overfitting specific action sequences. In particular, the experience replay is extremely valuable since it allows the system to improve the variety of the training samples by getting experience from the states seen by different agents.

As described in the Appendix, we exploit the *Double Q-Learning* technique to speed up the algorithm's convergence [130]. Hence, we maintain a *target network* to evaluate agents' actions and an *primary network* to select the policy, computing the bootstrap Q-value as

$$Q_{\text{new}}(o(t), a(t)) = r(t) + \lambda Q^{\text{primary}}(o(t+1), \arg\max_a Q^{\text{target}}(o(t+1), a)). \tag{4.12}$$

We use the value $Q_{\text{new}}(o(t), a(t))$ to perform backpropagation on the primary network and improve the agent policy. We iteratively replace the target network's weights with those of the update network and determine the algorithm learning rate using the Rectified Adam (RAdam) optimizer [131].

We implement the well-known ε-*greedy* and *softmax* policies to allow the agents to explore the action space. The exploration rate ε follows 2 different approaches, namely, ε-greedy and softmax. In the former, random action is chosen with probability ε, while the greedy action, i.e., the action with the highest Q-value, with probability 1-ε. The value of ε decreases to 0 at the end of the training since no more exploration is needed. In the latter, we compute the action probability $\pi(o, a)$ for the observation o as the output of a softmax density function taking the Q-values as input. In this case, the temperature T decreases during the training, reducing the randomness during the selection of the actions:

$$\pi(o, a) = \frac{e^{\frac{Q(o,a)}{T}}}{\sum_{a \in \mathcal{A}} e^{\frac{Q(o,a)}{T}}}\ , \tag{4.13}$$

where $A = |\mathcal{A}|$ is the cardinality of the UAV action space.

In our model, the observation for agent $n \in \mathcal{N}$ is represented by four $F \times F$ matrices. The first is a $F \times F$ matrix with all the values set to zero but the cell corresponding to the agent position, which are equal to 1. Instead, the other matrices are $\mathbf{X}_n$, $\hat{\mathbf{\Phi}}_n$, and $\hat{\mathbf{\Omega}}_n$, which have been defined in the previous section. To simplify the state space, we consider matrices $\hat{\mathbf{\Phi}}_n$ and $\hat{\mathbf{\Omega}}_n$ jointly, by

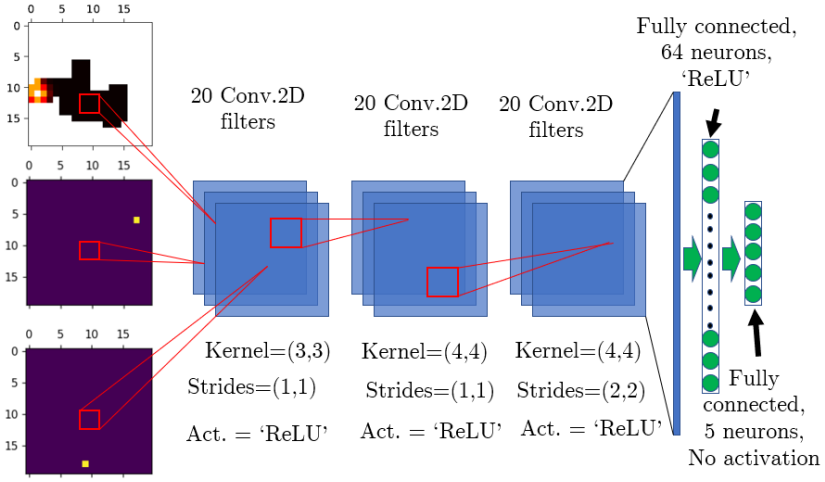


Fig. 4.3: Learning architecture.

defining the matrix $\hat{\mathbf{\Phi}}_n - \rho\hat{\mathbf{\Omega}}_n$, where ρ is a scalar parameter used to facilitate learning. Hence, our system approximates the function $Q(\cdot)$ by a Convolutional Neural Network (CNN), which receives an image with three channels as input and returns a vector as output. The learning architecture is described in Fig. 4.3, and includes three convolutional layers followed by two fully-connected layers. The dimension of the last layer is identical to the number of actions so that the CNN output can converge to the Q-values $Q(o, a)$, $\forall\ a \in \mathcal{A}$.

4.4.3 Computational Complexity

We now discuss the computational complexity of performing one inference procedure with our DQN architecture. We first analyze the complexity of fully-connected layers, denoting by N_k the number of neurons in the general k-th layer. To go from layer i to layer $i+1$, we need to compute the value of N_{i+1} nodes, each of which takes N_i multiplications followed by N_i additions and one non linear function. Therefore, the total number of operations is $N_{i+1}(2N_i + 1)$.

Then, we compute the complexity of one convolutional layer, as done in [132] when neither batch normalization nor pooling layers are present. We denote with (I_w, I_h, I_d) the shape of the input block. At layer i, we then have K_i filters with kernels dimension (W_i, H_i), stride S_i (we use the same value along the two axes), and padding P_i. The shape of the resulting output block will be $(\frac{I_w+2P_i-W_i}{S_i} + 1, \frac{I_h+2P_i-H_i}{S_i} + 1, K_i)$. The computation of each block's neuron involves $W_i \times H_i \times I_d$ multiplications followed by the same number of additions (sum all elements plus the bias) and one non-linearity. The total number of calculations is then $(\frac{I_w+2P_i-W_i}{S_i} + 1) \times (\frac{I_h+2P_i-H_i}{S_i} + 1) \times K_i \times (2W_iH_iI_d + 1)$.

Considering our specific architecture, the number of computations (multiplications, additions and non-linear operations) are 440 000, 3 704 980 and 628 180 for the three convolutional layers. Instead, the following fully-connected layers require 125 504 and 645 computations. Therefore, the total number of operations for one decision is 4 899 309. This computational complexity

allows UAVs to take decisions in real-time, as even embedded processors can deal with much more complex operations in less than 100 ms [133]. In our scenario, the speed of the UAVs and the vision algorithms required to identify targets are the main limiting factors for the swarm.

4.5 Simulation Settings and Results

In this section, we describe our simulation settings and evaluate the proposed learning approach in various scenarios with specific characteristics.

4.5.1 System Settings

We derive all the results through a Monte Carlo approach, where multiple independent simulations are carried out to obtain reliable statistical data. The training phase includes N_{train} episodes for each scenario (sparse or cluster), where each episode lasts T_{train} slots. In particular, training episodes are far longer than test episodes (which include T_{test} slots each) since the agents need more time for collecting experience. Before training, we initialize the replay memory by executing $N_{\text{memory}} = 1000$ episodes of T_{memory} slots each, allowing the agent to start the learning procedure immediately. If the episodes are too long, memory replay will mainly receive samples in which large portions of the map are explored, and the agents will not learn how to move at the beginning of the episodes. On the other hand, short episodes have the opposite problem, as the UAVs never learn to behave in the final parts of the episodes. A prioritized memory replay may solve this problem but increase the complexity of our framework.

We then opted for adapting the episode length in the training phase, making even training episodes have $T_{\text{train}}^{\text{even}} = 50$ slots each, and the odd episodes have $T_{\text{train}}^{\text{odd}} = 150$ slots. This alternating procedure prevents the replay memory from being too skewed towards situations where the map is fully explored or unexplored. Moreover, we apply TL to allow the agents trained in the sparse environment to quickly adapt the network to the cluster scenarios (or vice-versa). In such a case, we carry out N_{transf} training episodes. In all the cases, we test the performance of the proposed strategy in a total of $T_{\text{test}} = 500$ episodes. The complete simulation settings are reported in Tab. 4.2.

Tab. 4.2: Simulation settings.

Parameter	Value	Description	Parameter	Value	Description
M	{20, 24, 30, 40, 50}	Cell per map edge	F	20	Observation window size
N	$\{2, 3\}$	UAV number	K	4	Target number
σ^2	1	Target variance	d_{sparse}	8	Minimum target distance
η	{0,0.1}	Map fraction occupied by obstacles	ζ	3	UAV field of View
θ	1	Penalty for forbidden action	ψ	0.8	Penalty for collisions
ρ	0.2	Obstacle tuning parameter	γ	0.9	Discount factor
N_{train}	$\{250, 750, 1000, 3000\}$	Training episodes	N_{memory}	1000	Testing episodes
T_{train}	$\{50, 150\}$	Steps per training episode	T_{test}	40	Steps per test episode
N_{transf}	$\{125, 250, 375, 750\}$	TL episodes	N_{test}	100 (LA), 500 (DDQL)	Testing episode
P_{tx}	20 dBm	Communication power	N_0	-76 dBm	Noise floor

To assess the performance of our DDQL scheme, we compare it with a heuristic strategy based on Markov predictive control, by which drones can explore the map and reach the targets. Such a benchmark strategy is named *LA* and, at each slot, evaluates all possible combinations

of future actions and rewards, as its name suggests. In order to define it, we first define the LA reward $r_n^\ell(\mathbf{X}, \mathbf{a})$ as:

$$r_n^\ell(\mathbf{X}, \mathbf{a}) = \begin{cases} \frac{\hat{\phi}(\mathbf{x}_n + \mathbf{a}_n)}{\xi(\mathbf{x}_n + \mathbf{a}_n)} & \text{if } \nu(\mathbf{x}_n, \mathbf{a}_n) = 1; \\ -\infty & \text{otherwise,} \end{cases} \tag{4.14}$$

where $\xi(\mathbf{m})$ is the number of UAVs located in $\mathbf{m}$. The LA strategy never chooses actions going outside the map or on obstacles and makes each drone n maximize its expected cumulative reward over the following T_ℓ slots, assuming that none of the others is moving. In practice, the LA strategy makes each drone select the action a^* that maximizes

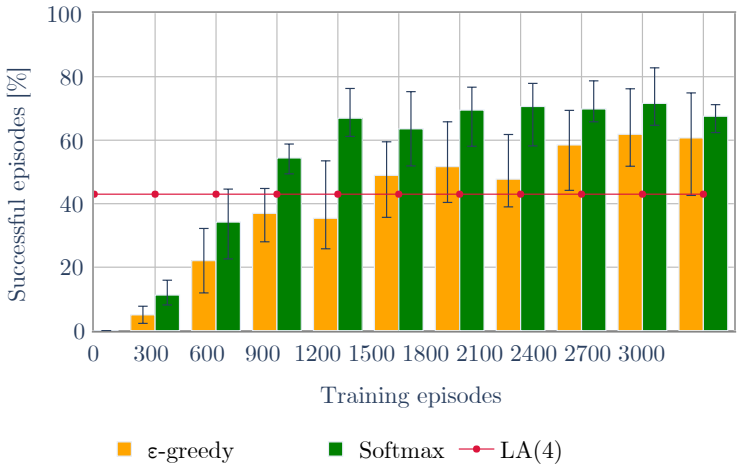
$$\max_{\mathbf{a} \in \Delta^{T_\ell}} \sum_{i=0}^{T_\ell - 1} r_n^\ell \left(\mathbf{X}, \sum_{j=0}^{i-1} \mathbf{a}^j \right), \tag{4.15}$$

where Δ^{T_ℓ} is the set of ordered sequences of action vectors $\mathbf{a}^0$, $\mathbf{a}^1$, ..., $\mathbf{a}^{T_\ell - 1}$, so that $\mathbf{a}_n^0 = a^*$ and $\mathbf{a}_m^i = (0, 0)$, $\forall\ i \in \{0, ..., T_\ell - 1\}, m \neq n$. In other words, Δ^{T_ℓ} is the set of possible move sequences of n assuming that the other UAVs are static. We observe that if several action sequences have the same expected reward, the LA strategy chooses one of them randomly.

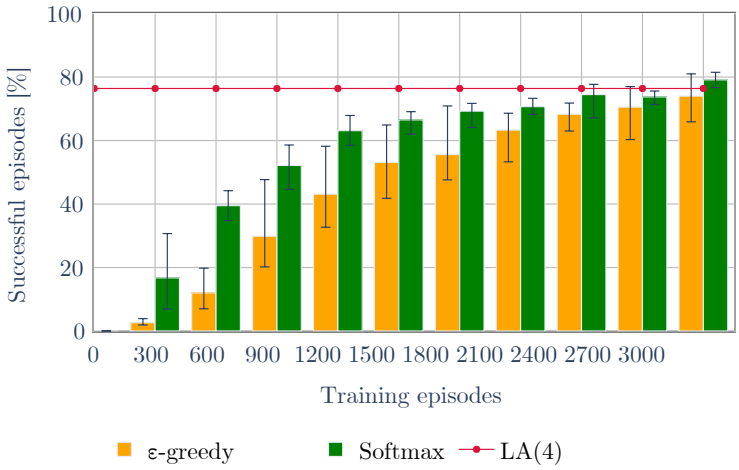
We note that each drone $n \in \mathcal{N}$ assumes that all the unexplored values $\phi(\mathbf{m})$ are equal to 1. Hence, the LA forces n to explore the map until it finds a target. Then, the considered UAV remains over the new target while the other drones remove the target information from their local value maps. The performance of LA mainly depends on the n_ℓ parameter: as it increases, drones can make more foresighted decisions but at a higher computational cost. In addition, the number of targets on the map also plays a role in determining the computational performance. If more targets are present, we have to check whether other agents are on a target more often to remove it from the map of available targets. As the LA strategy is computationally expensive, we set $N_{\text{test}} = 100$ when evaluating its performance.

4.5.2 Performance Analysis

We first consider a scenario with $N = 2$ UAVs and $k = 4$ targets in a 20×20 map. We perform multiple training phases of different duration, up to $N_{\text{train}} = 3000$ episodes, for a total of $3 \cdot 10^5$ training samples, ensuring that all our algorithms achieve convergence. In the following, we abbreviate the LA approach as LA(4), as we set $n_\ell = 4$. This algorithm requires a lot of computational resources: each LA decision takes approximately 15 times longer than using the DDQL strategy. We do not consider $n_\ell > 4$ since the computational cost of such a technique becomes excessive with limited performance gains. In some brief tests (which had to be on maps of a limited size due to the computational complexity of LAwith a longer horizon), we noticed that LA(8) and even LA(12) lead to similar results for LA(4). Without coordination among the UAVs, there is a limit on the swarm performance even with an infinite horizon. The most critical factor in determining the speed at which the UAVs find the target becomes the coordination of the swarm once the horizon reaches 3 or 4 slots.



(a) Cluster scenario.

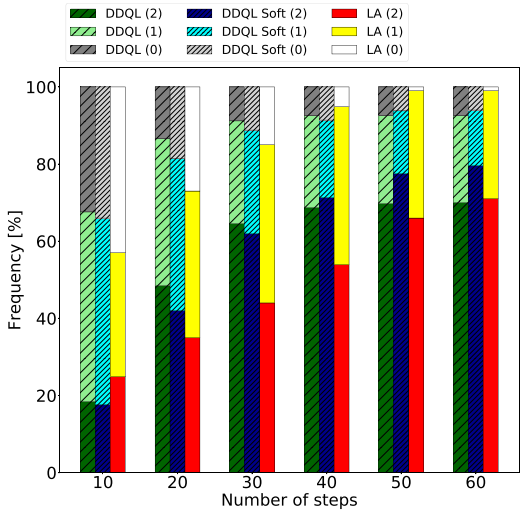


(b) Sparse scenario.

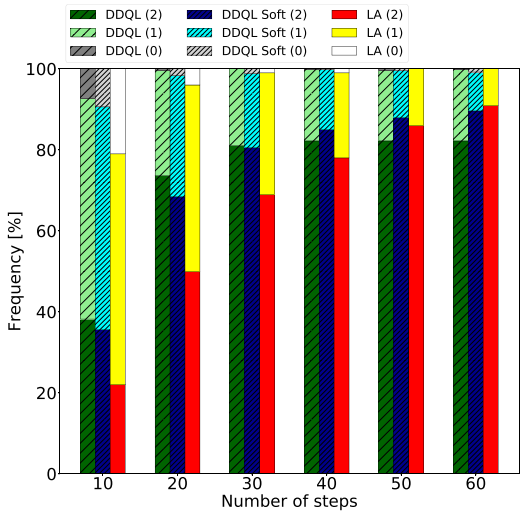
Fig. 4.4: Success probability during the training with 2 UAVs.

Fig. 4.4a shows the success probability in the cluster scenario as a function of the training set size and the considered exploration profile and approach. In particular, DDQL combined with the softmax approach catches up with LA(4) in less than 900 training episodes, converging to a success probability between 0.65 and 0.7. The ε-greedy approach has lower final performance and requires more time to converge with respect to the softmax profile. The error bars show the best and worst results over 5 test phases, proving that the performance improves as the UAVs gains more experience. The performance boost over the LA approach is due to the DDQL scheme's ability to exploit the correlation among the target positions, quickly finding the other targets after spotting the first one. In the sparse scenario, the final performance of DDQL is

similar to that of LA(4), as shown in Fig. 4.4b. We observe DDQL and LA(4) have more success than before, as finding the scattered targets is simpler than identifying clusters in the limited duration of an episode.



(a) Cluster scenario.



(b) Sparse scenario.

Fig. 4.5: Probability distribution of the number of drones reaching the targets.

In the following results, we better analyze the strategy learned by the proposed DDQL framework. First, Fig. 4.5 reports the probability of one or both drones reaching the target as a function of the number of slots per episode. Each group of bars refers to the performance achieved by DDQL (with and without softmax) and by LA in the cluster and sparse scenarios, with a total of $K = 4$ targets and $N = 2$ UAVs. Therefore, the figure shows the trade-off

between the time needed by UAVs to accomplish their task and the success rate. In the cluster scenario (Fig. 4.5a), DDQL is much faster than LA. At the same time, the DDQL performance peaks out, and, after 40 slots, the success probability does not change significantly. In many cases, when a drone reaches the target, but the other one is far from any feature of the map, the second drone stops its motion. This phenomenon occurs because the agent Q-values for that scenario are not precise, and all actions have a similar value. This situation rarely happens before the first UAV reaches a target as the system change state more frequently, making both the drones move.

Conversely, the LA approach does not suffer from the described problem, and the benchmark success rate keeps increasing with time. In the sparse scenario (Fig. 4.5b), LAeven ends up reaching more targets than DDQL after 50 slots. To overcome the issue, we decided to maintain a low softmax temperature even during the test phase. The bar chart shows that the softmax system is slightly slower than the greedy DDQL at the beginning but leads to better final performance. This randomization allows the agent to get out of loops, as sometimes a random sub-optimal action will change the observed state. Instead, the greedy system keeps performing the same action without getting new information about the environment. Essentially, LA does the same, randomizing its action when it cannot establish the best one.

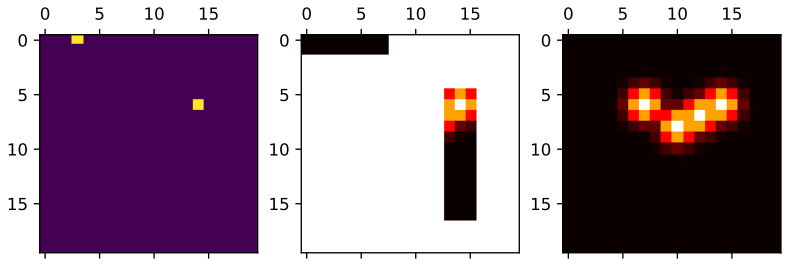


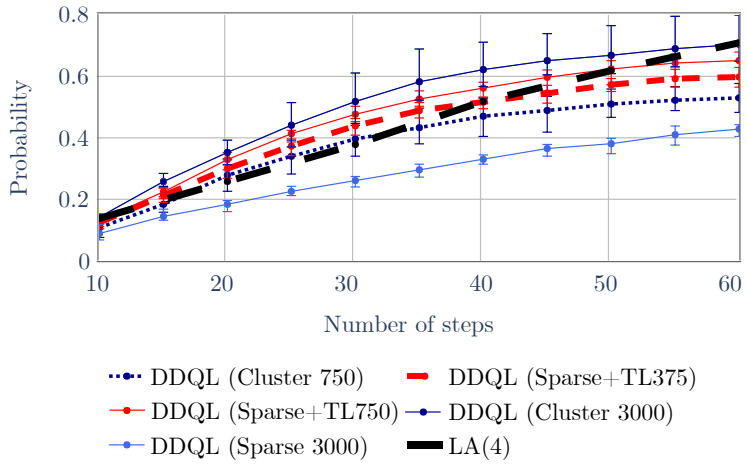
Fig. 4.6: Episode where an UAV is not able to reach the cluster.

Fig. 4.6 shows one of the situations described before: one UAV has reached its goal, while the other is far from any identified targets. Hence, the Q-values of the second UAV will be very similar to each other, making it stay motionless or follow small loops, as its state never changes. The fact that most of the map is still unexplored increases the probability of the UAV getting stuck, as it will have limited information. In the following, all the results are referred to the softmax system with a temperature $\tau = 0.1$ unless otherwise stated.

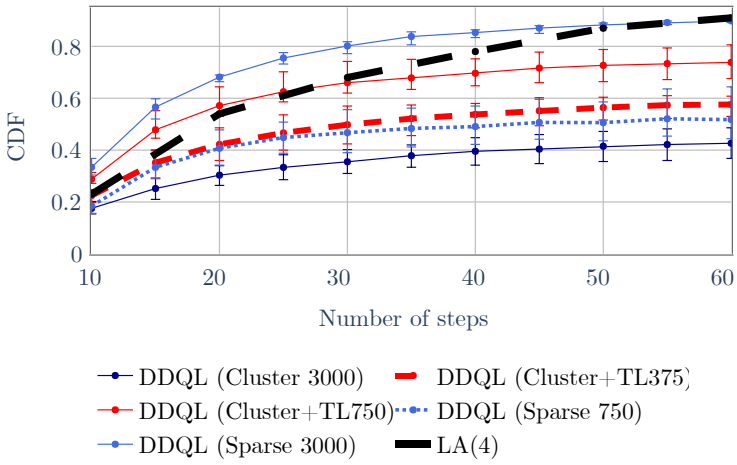
4.5.3 Transfer Learning

In the following, we investigate the adaptability of the proposed scheme and the potential of the TL paradigm. When using TL, we carry out an additional training phase in a different scenario than the one seen initially. To this end, we consider a first scenario, i.e., cluster (or sparse), and compare the results achieved using strategies learned in the other domain, i.e., sparse (or cluster). More specifically, we consider the following cases:

- Cluster N_{train}: training on N_{train} episodes in the cluster scenario;
- Sparse N_{train} : training on N_{train} episodes in the sparse scenario;
- Cluster+TL N_{transf}: pre-training on $N_{\text{train}} = 3000$ episodes in the cluster scenario, followed by an additional training of N_{transf} episodes in the target scenario.
- Sparse+TL N_{transf}: pre-training on $N_{\text{train}} = 3000$ episodes in the sparse scenario, followed by an additional training of N_{transf} episodes in the target scenario.



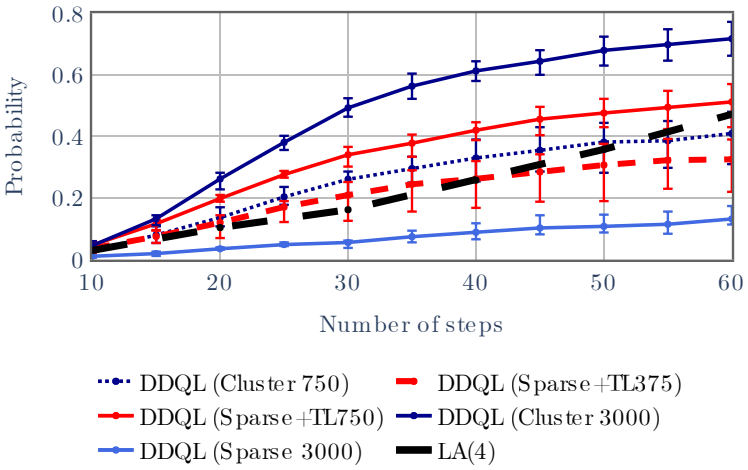
(a) Cluster scenario.



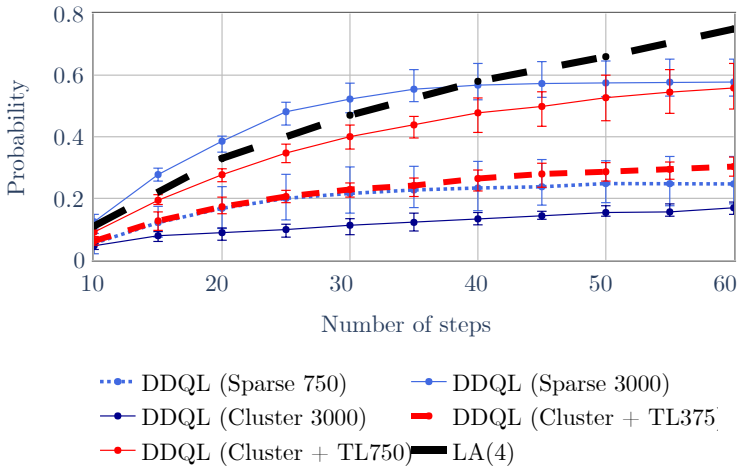
(b) Sparse scenario.

Fig. 4.7: CDF of the episode duration with 2 UAVs.

In Fig. 4.7a, we analyze the Cumulative Distribution Function (CDF) of the *episode duration*, defined as the time until all the drones reach targets or the episode limit (here fixed to 60



(a) Cluster scenario.



(b) Sparse scenario.

Fig. 4.8: CDF of the episode duration with 2 UAVs.

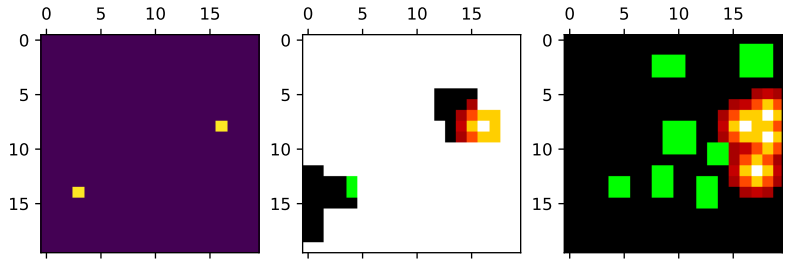
slots) is reached. We report the results for both the DDQL systems and LA with $n_\ell = 4$ as a benchmark. Each point represents the probability that all drones have accomplished their task by a pre-determined number of slots. As expected, the Cluster 3000 strategy achieves the highest success probability with a limited number of slots. The LA approach equals its performance only when the episode duration reaches the limit of 60 slots, i.e., in less than 30% of the cases. On the other hand, 750 training episodes in the cluster scenario are not sufficient to outperform LA(4) but enough to outperform the model trained with sparse targets. In particular, short retraining of such a model in the correct scenario allows the algorithm to get a significant performance boost. The resulting system outperforms the benchmark and gets

very close to the performance of the Cluster 3000 model, which is fully trained in the correct scenario and with a higher number of episodes.

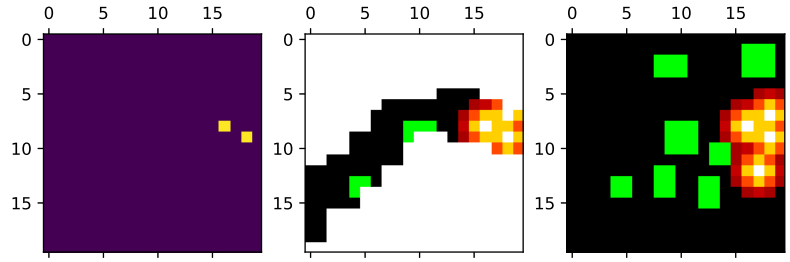
In Fig. 4.7b, we repeat the experiment by swapping the role of the sparse and cluster scenarios, and changing the number of episodes during the training phase, as reflected in the legend of Fig. 4.7b. As in the previous case, LA(4) meets the performance of DDQL only for episodes of 60 slots, i.e., in less than 15% of the cases. Besides, TL is very effective, as retraining of 750 episodes significantly boosts the baseline performance compared to starting a new model from scratch. In general, the number of slots necessary to reach the targets is comparatively lower than in the previous case. We observe this phenomenon since, as already discussed, it is easier for UAVs to find targets in the sparse scenario.

In Fig. 4.8, we show the results for a scenario with 3 UAVs. In both cases, TL is effective but leads to lower performance in the sparse scenario than in the cluster one. Besides, the risk of getting stuck is higher, and the algorithm needs more training to perform effectively on all maps.

4.5.4 Scenario Extensions



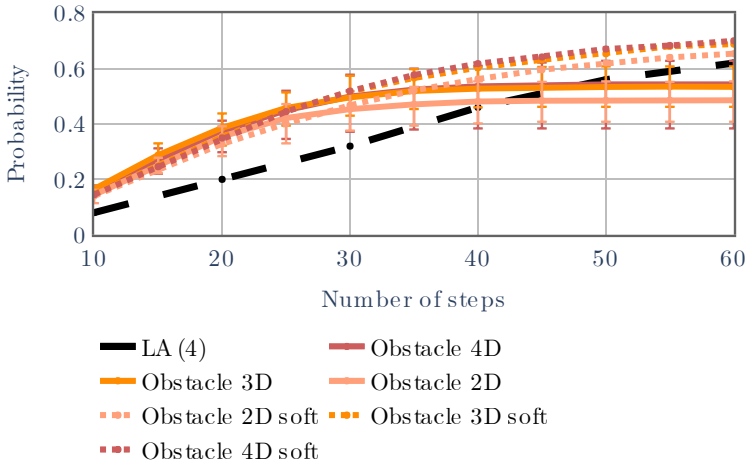
(a) Episode beginning.



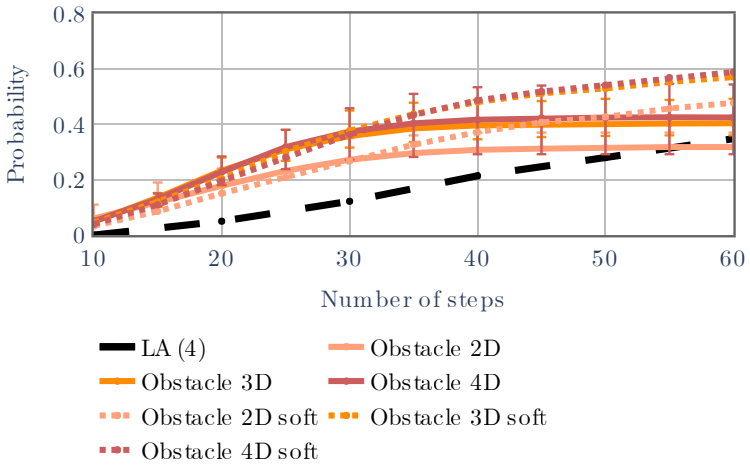
(b) Episode ending.

Fig. 4.9: Drone positions (left), known (center), and real map (right), with obstacles.

In what follows, we consider a modified version of the cluster scenario, with some obstacles added to the map. We empirically set the percentage of the map occupied by obstacles to 10%,



(a) 2 UAVs.



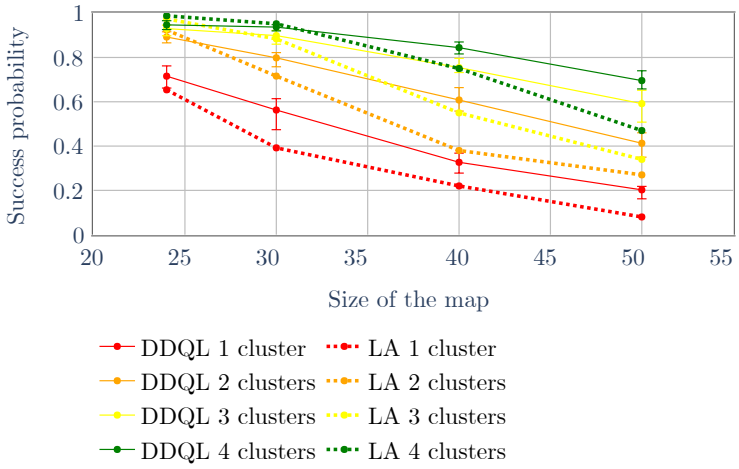
(b) 3 UAVs.

Fig. 4.10: CDF of the episode duration in the obstacle scenario.

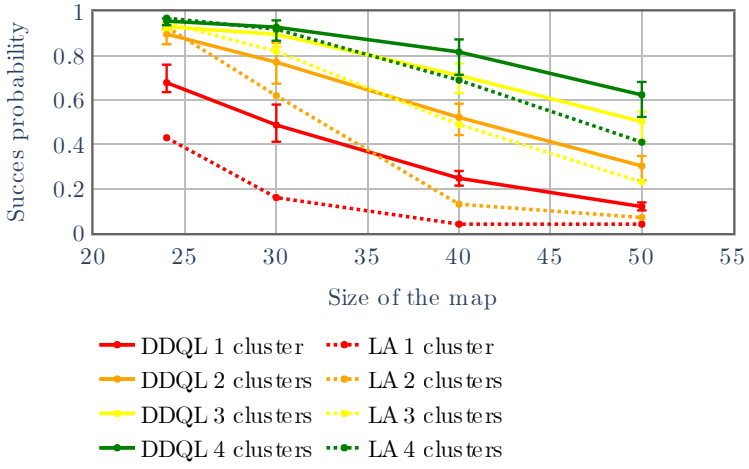
searching for a balance between increased system complexity and the realism of the scenario. In Fig. 4.9, we report an example of the system state at the beginning and the end of an episode, with the obstacles depicted in green. In particular, we can appreciate that the low obstacle density allows the UAVs always find a path to the target.

In Fig. 4.10a, we analyze the performance of LA and DDQL in the case of $K = 4$ targets. We train the DDQL solution for scenarios with $N \in \{2, 3, 4\}$ UAVs, and we label the resulting strategies as 2D, 3D, and 4D, respectively. Then, we test the system performance in the scenario with $N \in \{2, 3\}$ UAVs, with and without using the softmax algorithm in the testing phase. The models trained with more UAVs can outperform those with fewer UAVs. Furthermore,

the softmax policy during the testing phase increases the performance, especially when the episodes are longer, as it helps the UAVs to avoid stuck situations. In the scenario with 3 UAVs (Fig. 4.10b), the performance is generally lower, meaning that the swarm needs more training. At the same time, we observe that DDQL outperforms LA in both cases, reaching targets significantly faster in the scenario with 3 drones.



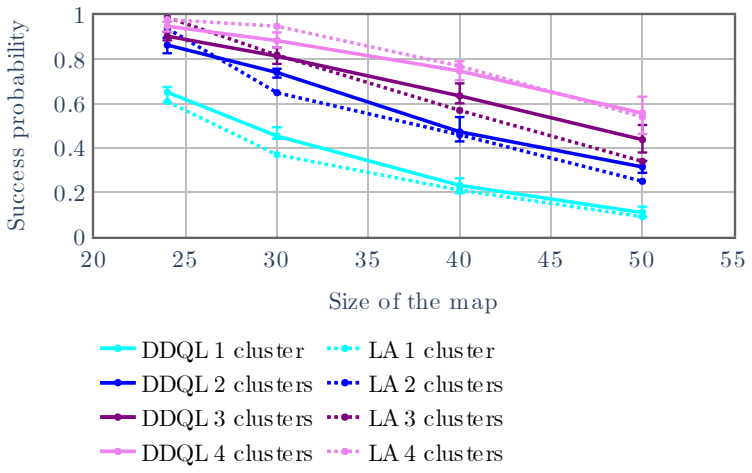
(a) 2 UAVs.



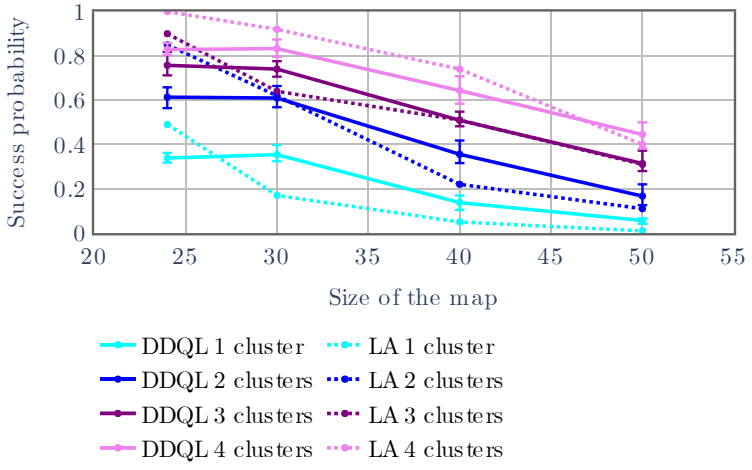
(b) 3 UAVs.

Fig. 4.11: Success probability as a function of the map size and the number of clusters.

Now, we analyze the performance of LA and the proposed DDQL strategy in wider maps, i.e., settings $M > F$. We train our algorithm on a map with $M = 24$, maintaining $F = 20$. Instead, we consider a higher value for M and vary the cluster number during the testing episodes. In order to allow the agents to find the targets under such conditions, we extend the episode



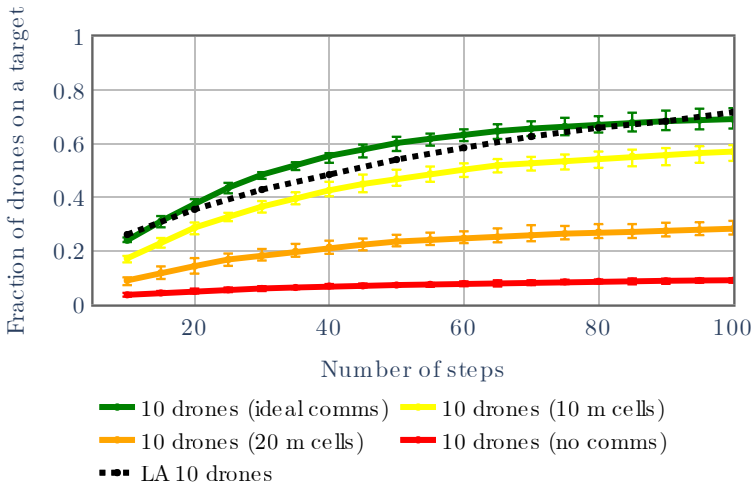
(a) 2 UAVs.



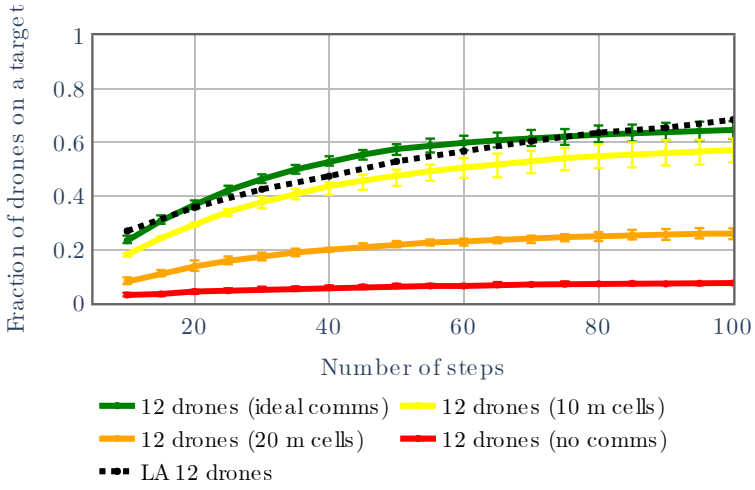
(b) 3 UAVs.

Fig. 4.12: Success probability in the obstacle scenario.

duration, considering 100 slots per episode. At the same time, we make the scenarios with more clusters maintain a similar proportion of surface occupied by targets despite the bigger map size. In Figs. 4.11a and 4.11b, we evaluate the system performance as a function of the map size and the number of clusters. In both cases, DDQL shows good adaptability, getting better performance than LA in all cases, with higher gains in bigger maps. In Figs. 4.12a and 4.12b, we consider the same scenarios with the addition of obstacles. In this case, DDQL will need some retraining to reach LA's performance on smaller maps, while the performance is similar when the map is wider. However, we recall that DDQL also has a significant advantage in terms of computational cost, so it is preferable if performance is similar.



(a) Performance of a swarm of 10 UAVs.



(b) Performance of a swarm of 12 UAVs.

Fig. 4.13: Effect of imperfect communications in a large map.

As the next step, we analyze the system performance in the case of realistic communications. Our architecture relies on an immediate sharing of local information, and a limited communication range can impair its performance significantly. In Fig. 4.13a shows, we consider 10 drones moving on a map covered with obstacles. We can appreciate that the DDQL strategy enables swarm coordination effectively with no retraining, outperforming the LA approach. The performance loss is limited even when the cell side is 10 m, corresponding to a maximum range of about 11 cells and 50% packet loss at the area boundary. These results confirm the intuitive idea that information from neighbors inside the visible area is the most critical to reaching the targets. Conversely, the success probability drops significantly if we double the cell side

and introduce errors even for packets between immediate neighbors. The performance degrades more critically when there is no information exchange between the UAVs.

Our system is a cooperative algorithm, and the information from other agents is necessary to optimize the map exploration. In particular, we have trained DDQL under ideal conditions, and the communication impairments have been considered only in the test phase. Hence, the UAVs are confused by the lack of information, and partial retraining might yield better results as the agents can learn to deal with the more limited feedback. On the other hand, the algorithm scales well to large swarms, slightly outperforming LAeven with no retraining in the new scenario. The same pattern holds for the case with 12 drones, whose results are reported in Fig. 4.13b.

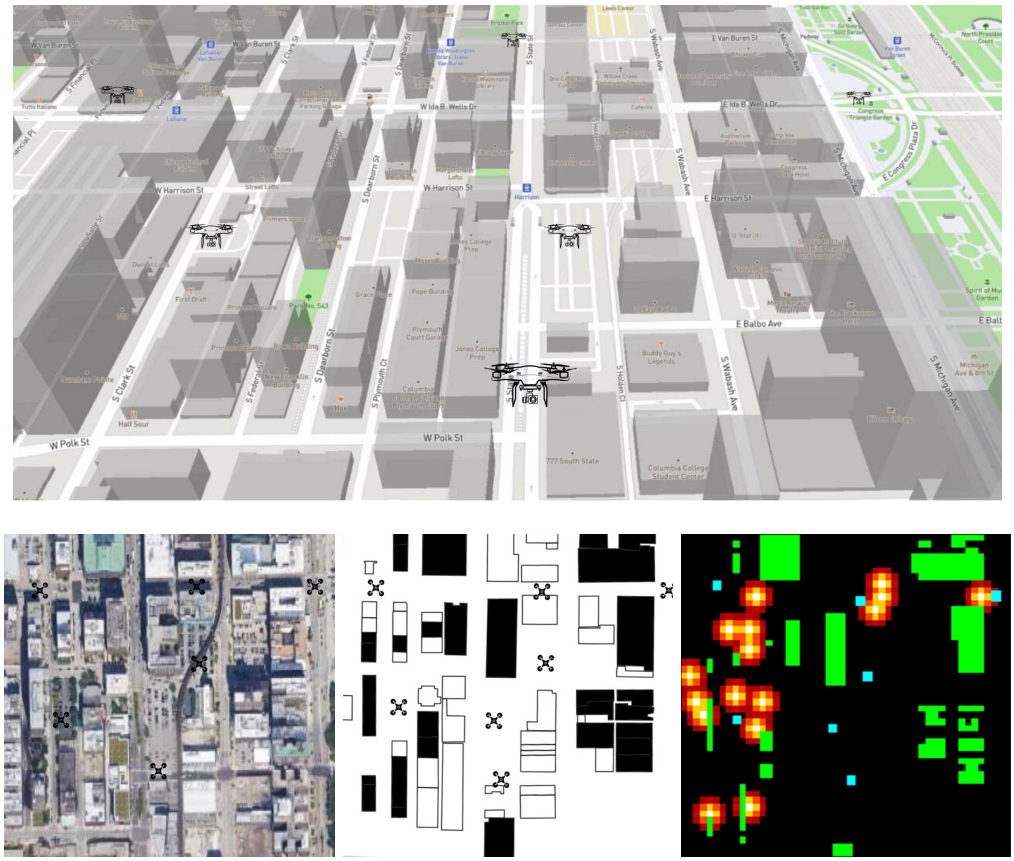


Fig. 4.14: Considered area in the downtown Chicago Loop neighborhood.

We perform a final test in a more complex scenario, increasing the map size and the UAV number and considering an obstacle distribution derived from an actual city map. We define the obstacle pattern according to a 500 m by 500 m area just east of LaSalle Street Station in downtown Chicago, in the central Loop neighborhood. As shown in Fig. 4.14, we obtain the height profiles of the buildings in the area and generate obstacles from the structures of over 10 stories. This height is approximately 40 m and coincides with the minimum legal hovering

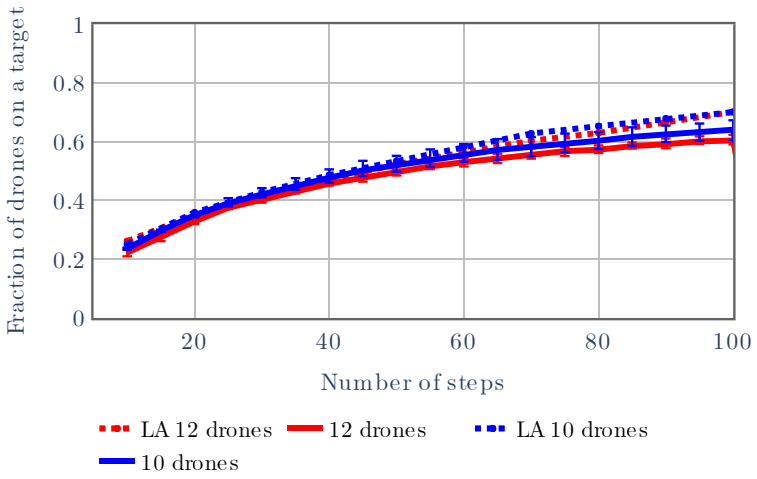


Fig. 4.15: Performances on the real map of Chicago.

altitude for UAVs). The area was then divided into 2500 square cells with 10 m sides, converting the height profiles to an obstacle map in the grid. In particular, the obstacles occupy about 11% of the map, a percentage similar to those used in the training phase. At the same time, the obstacles in the new scenario follow a different pattern than the training, concentrating along South Wabash Avenue and South Dearborn Street. This aspect is an additional hurdle for DDQL, which is not designed to deal with such a situation. As Fig. 4.15 shows, the DDQL system can find targets approximately as fast as LA. The only cases in which LA outperform the proposed system is when considering the higher percentiles of the success probability. We expect that fast retraining should adapt DDQL to the new scenario, exploiting the regularities in city blocks to avoid obstacles and find targets even more quickly.

The overall results showed that the proposed DDQL system is scalable to different maps, swarm sizes, and limited communications without additional training. In addition, it can deal with obstacles and very different target distributions with a limited amount of retraining, adapting to new conditions rapidly and efficiently. In particular, in most scenarios, DDQL allows UAVs to reach targets faster than the benchmark LA solution. There is still some improvement margin, particularly in scenarios where all the UAVs but one have already reached a target. This case represents most of the residual failures of the algorithm, and its resolution is a significant objective for the future.

4.6 Conclusions and Future Work

In this chapter, we investigated the problem of monitoring and surveilling a target area with a swarm of UAVs. We modeled the environment with a 2D grid and cast the problem into the theoretical framework of ND-POMDP. We conducted simulations in various settings, introducing obstacles that obstruct drone movements and maps of different sizes, considering a realistic

scenario based on the Chicago urban area. We showed that the proposed learning framework leads to efficient strategies and outperforms a computationally intensive LA approach in almost all the considered settings.

Future research directions include the introduction of dynamic targets, which make it possible to model more types of UAV applications. Another possibility consists in defining different roles for the network agents and training specific units to assign such roles. This framework will allow us to examine new aspects of MARL, increasing the complexity of the strategy thus learned.

5

Joint Communications and Control in Multi-Agent Networks

5.1 Introduction

Over the last decade, the use of Internet of Things (IoT) technology for industrial and manufacturing applications has exploded [134]. The digitization and automation of industrial processes are fundamental for the Industry 4.0 paradigm [135], while wireless communications have fostered the implementation of mobile robots in such scenarios. Notably, we can approach the optimization of robot systems from two opposite directions. The first is the field of networked control systems [136], which incorporates the Quality of Service (QoS) of the communication system into the controller design process, using metrics such as the latency distribution or the packet error rate. Instead, more recent works have focused on the communication system, measuring the freshness of the information available to the controller and optimizing scheduling and resource allocation to maximize it [137].

Another significant development in the field has been the explosion of Deep Reinforcement Learning (DRL) techniques, which began in 2015 with DeepMind's seminal paper on Deep Q-Networks (DQNs) [118]. The union of deep learning with reinforcement techniques has triggered a revolution in robotics, as DQNs can provide far higher control performance than human-designed techniques in various complex situations. Naturally, the DQN is optimized for its training scenario and will effectively act as a networked control system, countering the impairments caused by the wireless connection and maximizing performance accordingly. However, the inherent complexity of these learning agents complicates the design of the communication system. Since learning agents operate as black boxes, we cannot know which piece of informa-

The work presented in this chapter was submitted to *IEEE Transactions on Mobile Computing*.

tion will be crucial and which will be almost irrelevant [138], reducing the effectiveness of Age of Information (AoI)-based approaches.

The scientific community is increasingly interested in semantic communications, which do not solely consider the QoS of data delivery but also their content and meaning. In his famous introduction to the expanded version of Shannon's seminal paper [139], Warren Weaver posited an *effectiveness problem* that goes beyond the simple transmission of messages, arguing that the ultimate goal of communication is to affect the behavior of intelligent agents. Even though effectiveness has been discussed for decades [140], its complexity has limited the possibility of practical solutions until recently, when the massive improvement of machine learning techniques has allowed researchers to tackle it. However, this research field is still in its infancy, and the joint optimization of control and communication is an open problem for non-trivial scenarios [141].

In this chapter, we design a general model for Cyber-Physical Systems (CPSs) that integrates communication and control, allowing for their joint optimization using Multi-Agent Reinforcement Learning (MARL) [142]. More specifically, we consider a dynamic environment in which multiple agents operate and interact with each other. Some agents (robots) can perform actions that affect the physical environment, while others (sensors) can communicate their observations to the robots through resource-constrained communication channels. This scheme maps directly to the effectiveness problem, as the ultimate goal is to optimize the control policy of the robots. At the same time, sensor communications are necessary for the robots to compute an accurate picture of the system state. Our model follows Markov Decision Process (MDP) theory, and the combination of communication and control aspects, as well as the interaction between distributed agents, led us to name it Cyber-Physical POMDP (CP-POMDP). Hence, our proposal expands the Networked Distributed Partially Observable Markov Decision Process (ND-POMDP) framework, which includes agent communications but does not optimize them directly.

The proposed system captures the direct and indirect aspects of communication, as sharing information can affect the physical environment and the observations available to each agent. More importantly, our model defines control and communication decisions expressly and enables the joint training of agents that perform one or both actions, i.e., interact with the physical environment and communicate local information to other nodes. To the best of our knowledge, this is the first model for MARL that considers the dual nature of the problem, optimizing communications and control together. Our aim is to effectively bridge the gap between the communication and the control side, joining the two in a single comprehensive MARL model, both expressive and lightweight.

The benefits of solving the effectiveness problem are more significant if communications are constrained, i.e., if the amount of data that needs to be transmitted is non-negligible with respect to the channel capacity. Therefore, as test cases, we consider two underwater scenarios with acoustic communications [143] to showcase our CP-POMDP system. In both scenarios, an Autonomous Underwater Vehicle (AUV) needs to achieve a specific mission and reach a surface vessel, supported by a group of static buoys that can expand the AUV field of view by communicating their observations. As the acoustic medium can only support very low bitrates,

the communications from the buoys are limited and possibly affected by packet collisions. The mission in the first scenario involves avoiding obstacles, which could be underwater installations, natural rock formations, or sea mines. In the second case, we consider a data muling scenario [144] in which the AUV needs to recover data from floating sensors that might move with the current.

To evaluate our joint optimization approach, we implement a selection of heuristic schemes for the communication system (such as scheduling the closest buoy at any time). Our framework improves the system performance both when a centralized agent coordinates all the buoys and when the buoys act independently. When using the centralized approach, the buoys acquire a complete perception of the map and provide the most valuable information to the AUV accordingly. Instead, the distributed scheme makes it necessary for each buoy to learn how to coordinate and avoid packet losses. Naturally, the proposed model is not limited to these simple scenarios, and we provide some examples of larger and more complex CP-POMDPs.

We organize the rest of this chapter as follows: Sec. 5.2 presents the state-of-the-art on AoI optimization and networked control systems; Sec. 5.3 defines the proposed CP-POMDP, while Sec. 5.4 reports the underwater scenarios where the model is tested; Sec. 5.5 describes the learning approaches used to compute the control and communication policies; Sec. 5.6 gives the simulations results; finally, Sec. 5.7 concludes the chapter and lists some possible avenues for future work.

5.2 Related Work

Over the past few years, the scientific community has put significant efforts into optimizing communication in resource-constrained scenarios such as IoT networks. Recently, it has become clear that packet latency is not the most appropriate metric for this task, which has led to the rise of AoI [145]. Unlike simple latency, AoI evaluates the update generation process as the network [146] and can better represent remote monitoring applications. Since its first introduction, there has been a plethora of works computing AoI in different scenarios and optimizing communication systems to minimize the average or the tail of AoI distribution [96].

However, AoI does not necessarily convey enough information to optimize intelligent networked control systems: the underlying process that the transmitter is monitoring might remain stable over long periods, making frequent updates wasteful, or sharply change, making the new information relevant even if an update has just been transmitted. In Markovian scenarios where the process takes discrete values, we can solve this issue by using the concept of *age of incorrect information* [147], which extends the notion of age to consider only updates that carry new information. In particular, as long as the process remains in the same state, new updates are not valuable. This concept can also be extended to continuous-valued and non-Markovian processes, leading to the more general concept of Value of Information (VoI) [138]. This metric is essentially an error function that depends on the difference between the estimated state at the receiver and the actual state of the process, which can integrate complex processes and predictive estimators [1].

Age and packet loss are not the only issues in IoT networks: often, the most important constraint on networked control systems is the bandwidth. Communication capacity is critical if sensors transmit video or other feature-rich data, used in new vision-based control applications, requiring significant bit rates. In this case, the communication system needs to choose which information to transmit and at which compression level [148] in order to maximize the system performance while avoiding channel congestion.

However, even VoI might not be the optimal choice for networked control systems, as errors in the state estimation might or might not translate to lower control performance. If the controller is highly sensitive to some error types while relatively insensitive to others, defining the error function becomes complicated, and it is better to use control performance as a proxy. This phenomenon leads to the Urgency of Information (UoI) metric [149], which directly considers how much an update would affect a (known) controller. However, UoI assumes the controller to be fixed, while the communication system is optimized to accommodate it.

The work on networked control systems and MARL goes in the opposite direction, taking a fixed communication strategy and optimizing the controller (or controllers) to deal with it. The most common method to deal with network impairments is to model a limited number of QoS metrics and design a controller that can perform well within a given operating region. To address the latency limits, network designers assume that packet delay follows a given distribution and implement the controller to handle it [150]. It is also possible to take the difference between the delays in the two directions into account, as in [151]. Another fundamental parameter for designing networked control systems is packet loss, as the erasure of an update can significantly affect the controller's decisions [152]. We can deal with such errors by using reliable communication protocols, retransmitting lost packets, or using Forward Error Correction (FEC), trading additional latency for communication reliability [153]. Naturally, the design of the controller is affected both by the features of the connection itself and the protocol.

Finally, controllers need to deal with compression, often expressed as quantization [154] or partial information, which can impair both inputs and outputs of the controller [155]. Learning systems have recently been used to deal with random delays and errors [156], as their generalization capabilities allow them to better deal with unforeseen situations and compensate for network impairments, particularly in non-linear systems [157]. Learning systems are also a viable option in case of compression of feedback or control signals [158]. However, to the best of our knowledge, no unified framework for learning in networked control systems has been proposed.

The problem of communication and control has also been approached in MARL, either for optimizing communication or dealing with imperfect and delayed inputs. One of the first works to do so, by Foerster *et al.* [117], used simple problems that required coordination between agents, showing emergent communication capabilities. Other works use explicit or implicit communication costs so that agents can choose whether to coordinate their actions or take decisions independently [159]. For a more thorough review of the cooperative MARL literature, we refer the reader to [160].

Two recent works are the most similar to our own: in the first [161], multiple agents learn

the parameters of a simple scheduling algorithm while performing a reinforcement task in real-time. In that case, however, the usage of a simple scheduling algorithm limits the solution, constraining its maximum performance by the communication policy. Instead, our work uses the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) framework to model communication and control entirely within the MARL problem, making it possible to fully explore the system environment and identify new and more efficient solutions. The second one [141] considers a scenario similar to our own, with one sensor and one robot. The task involves learning the channel statistics and designing a shared codebook instead of learning which information to transmit. In some ways, this work is complementary to ours as it focuses on encoding the available data instead of estimating the most valuable piece of information in a distributed setting.

5.3 Joint Communication and Control

We can define the problem addressed in this chapter as a special case of the more general class of Dec-POMDPs. Hence, while Dec-POMDPs considers communication implicitly, our model gives an explicit structure to the communication process among the system agents. We name our model CP-POMDP, as agents can both act in a physical environment and exchange information with other agents, coordinating to achieve a common goal.

We consider two sets of agents, named *sensors* ($\mathcal{G}$) and *robots* ($\mathcal{L}$). The sensors can perceive the environment and transmit their observations, while the robots have a limited view but can act on the environment and change its physical state. Some agents might be able to do both: in such cases, $\mathcal{G}$ and $\mathcal{L}$ are partially overlapping. The environment has a set $\mathcal{S}$ of possible states, and each agent $i \in \mathcal{G} \cup \mathcal{L}$ (either sensor or robot) receives observations from a set $\mathcal{O}_i$, depending on the current state. We denote the set of joint observations as $\mathcal{O} = \prod_{i \in \mathcal{G} \cup \mathcal{L}} \mathcal{O}_i$, and the probability of having a certain observation in a certain state as $P_o : \mathcal{S} \to \mathcal{O}$.

We consider that, in a given state s, each agent can communicate, move, or take both actions. Hence, the state space consists in the union of two subsets: $\mathcal{S}_c$ and $\mathcal{S}_m$, which include *communication* and *movement* states, respectively. If the environment is in a communication state, i.e., $s \in \mathcal{S}_c$, each sensor $g \in \mathcal{G}$ can transmit its information to other nodes, taking actions in the space $\mathcal{A}_\mathcal{G}(s)$. For instance, $\mathcal{A}_\mathcal{G}(s)$ can be binary, with one action for transmission and one for silence, or have multiple values for different possible messages. On the other hand, if the environment is in a movement state, i.e., $s \in \mathcal{S}_m$, each robot $\ell \in \mathcal{L}$ can make an action in the physical environment, with action space $\mathcal{A}_\mathcal{L}(s)$. Note that $\mathcal{S}_c \cup \mathcal{S}_m = \mathcal{S}$, which implies that at least one type of agent can make an action in each state; in the case $\mathcal{S}_c \cap \mathcal{S}_m \neq \emptyset$, there might be some states that allow both types of agent to act.

If the system is in state s, the joint agent actions are described by a vector $\mathbf{a} \in \mathcal{A}(s) = \prod_{g \in \mathcal{G}} \mathcal{A}_\mathcal{G}(s) \times \prod_{\ell \in \mathcal{L}} \mathcal{A}_\mathcal{L}(s)$, which is the concatenation of the single actions a_i, $\forall\, i \in \mathcal{G} \cup \mathcal{L}$. In

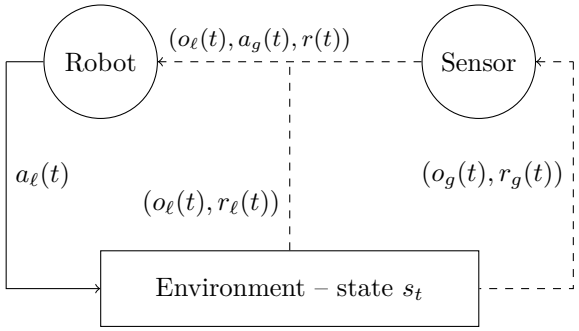


Fig. 5.1: CP-POMDP model with two agents.

particular, each agent can take action or not depending on the current state space:

$$\begin{aligned} \mathcal{A}_\mathcal{G}(s) &= \{a_\varnothing\}, \text{ if } s \notin \mathcal{S}_c; \\ \mathcal{A}_\mathcal{L}(s) &= \{a_\varnothing\}, \text{ if } s \notin \mathcal{S}_m; \end{aligned} \tag{5.1}$$

where $a_\varnothing$ is a null action with no effects. As in a classical MDP, the system evolution only depends on the current state and the robots' actions, and we denote the state transition probability function as $P_s : \mathcal{S} \times \mathcal{A}_\mathcal{L} \to \mathcal{S}$.

In our model, there are two different reward functions: r_g (for the sensors) and r_ℓ (for the robots), which both have $\mathcal{S} \times \mathcal{A}$ as a domain and return values in $\mathbb{R}$. The robots' reward is independent from the sensors, which implies that, given two different action vectors $\mathbf{a}, \mathbf{b} \in \mathcal{A}$, $\mathbf{a} \neq \mathbf{b}$, the reward does not change if the robots' actions are identical in the two cases. Instead, the sensor reward is indirect since it depends on both their actions and those of the robots. As we are considering a cooperative scenario, in which the goal of the sensors is to help the robot in its task, the two components of the reward will need to be designed with two objectives in mind: the first is the success of the communication, i.e., avoiding collisions or packet losses as much as possible, while the second is the *relevance* of the information that is transmitted, i.e., how much the new information can help the robots improve their performance.

We can see a simple schematic of the model in Fig. 5.1, with one sensor and one robot: while the actions of the robot affect the environment, the sensor only affects the future observations of the robot, contributing to the task indirectly by giving the robot the most useful information.

The full CP-POMDP model is then defined by the tuple $(\mathcal{G}, \mathcal{L}, \mathcal{S}_c, \mathcal{S}_m, \mathcal{A}_\mathcal{G}, \mathcal{A}_\mathcal{L}, P_s, P_o, r_g, r_\ell)$. In general, the nature of the model reflects the indirect effect of the sensors, whose actions do not affect the physical state of the environment but only the information available to other nodes. They can then change the transition probabilities from one state to another associated with the same physical features but yielding richer observations to all or a subset of robots.

By appropriately defining the state space and transition probabilities, it is possible to represent several different communication scenarios:

- *Scheduled transmission*: there is no interference between transmissions, but the set $\mathcal{A}_\mathcal{G}$ might be too small to encode all the information each sensor has about the state of the

system. The sensors then have to encode part of the information, deciding which parts of their observation are more important to transmit.

- *Slotted ALOHA*: the sensors need to coordinate to avoid interference. Each sensor can decide whether to transmit in a given state or which resource block to use, and collisions significantly reduce the reception probability. Therefore, each sensor needs to estimate the information utility and decide to transmit, risking a packet collision, or remain silent.

- *Spatial reuse*: the sensors have a limited communication range, reducing the interference with the transmissions in other regions of the scenario. However, the hidden terminal problem may still cause packet collision, and multi-hop transmission is necessary to disseminate sensors' observations.

In the same way, we can define specific scenarios depending on the number of robots and the effect of their actions:

- *Distributed awareness*: there is a single robot (i.e., $L = |\mathcal{L}| = 1$, where $|\cdot|$ is the cardinality operator), while all other nodes are sensors that aim to augment the robot's awareness and improve its performance in the task.

- *Coordination*: all nodes are both robots and sensors, and they must share their observations to achieve an agreement before acting.

- *Consensus*: this is a special case of coordination where the agents take action only if a consensus threshold is reached (either with unanimity or some form of qualified majority). A well-known example from the literature is the block-pushing task for a series of puck robots, who are too weak to push the block individually but can act in concert to accomplish the task [162].

5.4 Underwater System Model

In this section, we present the general system model we use as a test case for the CP-POMDP learning framework proposed in this paper. We consider an underwater scenario modeled by square grid of size $M \times M$; each cell of the map is identified by a coordinate vector $\mathbf{x} \in \mathcal{M} = \{1, ..., M\} \times \{1, ..., M\}$. We assume that an AUV ℓ moves within the environment to fulfill a mission and reach a surface vessel; hence, we have $L = |\mathcal{L}| = 1$ as in the distributed awareness scenario. The AUV has two possible tasks: the first involves the avoidance of marine debris, while the second is a data muling mission. To achieve its goal, the AUV is supported by a set of buoys $\mathcal{G}$, which can perceive their surrounding environment and communicate their observations to the AUV through an acoustic link. We resume the model notations in Tab. 5.1 as a reference to the reader.

Tab. 5.1: Model parameters.

Parameter	Description	Parameter	Description
t	Timeslot index	T	Timeslots per episode
$\mathcal{L}$	AUV set	$\mathcal{G}$	Sensor set
L	AUV number	G	Sensor number
$\mathcal{M}$	Coordinate set	M	Cell per map edge
$\mathcal{M}^*$	Valid coordinates	η	Collision penalty
$\mathcal{Z}$	Map status	$\mathcal{O}$	Joint observation space
$P_o(\cdot)$	Observation function	$P_s(\cdot)$	State transition function
$\mathcal{S}_m$	Mobility state space	$\mathcal{S}_c$	Communication state space
$\mathcal{O}_\ell$	Observation space of ℓ	$\mathcal{A}_\mathcal{L}$	AUV action space
$\mathcal{O}_g$	Observation space of g	$\mathcal{A}_\mathcal{G}$	Sensor action space
$r_\ell(\cdot)$	Reward function of ℓ	$r_g(\cdot)$	Reward function of g
ρ	Final reward	σ	Intermediate reward
$\mathbf{x}_\ell$	AUV position	$\mathbf{X}_\ell$	AUV position matrix
$\mathbf{x}_v$	Vessel position	$\mathbf{X}_v$	Vessel position matrix
$\mathbf{V}_\ell$	AUV vision matrix	$\mathbf{V}_g$	Sensor vision matrix
$\mathbf{Z}$	Map status matrix	$\hat{\mathbf{Z}}$	Estimated map status matrix
T_{period}	Transmission period	Δ	AoI matrix
$\mathcal{H}$	Set of transmission areas	H	Number of transmission areas
Z_{debris}	Number of debris rows	Z_{node}	Number of underwater nodes
$\Phi(\cdot)$	Debris avoidance reward function	$\Psi(\cdot)$	Data muling reward function

5.4.1 AUV Behavior

We assume that the time is discretized into slots $t \in \mathbb{Z}^+$, and we denote by $\mathbf{x}_\ell(t)$ the AUV position at slot t. Following a common Reinforcement Learning (RL) approach, we divide the time into episodes, each of which lasts $T \in \mathbb{N}$ slots. The system is always in a movement state (i.e., $\mathcal{S}_m = \mathcal{S}$), which implies that the AUV performs a new action $\mathbf{a}_\ell(t)$ at each slot t. The AUV needs to reach a target cell $\mathbf{x}_v \in \mathcal{M}$, which represents the surface vessel's location, after avoiding the debris or visiting the underwater nodes and recovering their data, depending on its mission profile. We assume that the vessel's position $\mathbf{x}_v$ is randomly generated at the beginning of each episode, but remains static throughout the episode. The AUV and starting vessel's starting positions are given by

$$\mathbf{x}_\ell(0) = (\mathcal{U}([1, N]), 1)\,, \tag{5.2}$$

$$\mathbf{x}_v = (\mathcal{U}([1, N]), N)\,, \tag{5.3}$$

where $\mathcal{U}([1, N])$ is a discrete uniform distribution in the interval $[1, N]$. Hence, the AUV starts moving from the lower edge of the map, while the vessel is located on the opposite side.

At each slot t, the AUV receives the observation $o_\ell(t) \in \mathcal{O}_\ell$, which includes its position, as well as the surface vessel's. We encode such information in the $M \times M$ binary matrices $\mathbf{X}_\ell(t)$ and $\mathbf{X}_v$, whose elements are defined as:

$$\mathbf{X}_\ell(t, \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} = \mathbf{x}_\ell(t); \\ 0, & \text{otherwise;} \end{cases} \tag{5.4}$$

$$\mathbf{X}_v(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} = \mathbf{x}_v; \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

In particular, $\mathbf{X}_\ell(t, \mathbf{x})$ and $\mathbf{X}_v(\mathbf{x})$ are the matrices' elements corresponding to the coordinate vector $\mathbf{x}$.

Another matrix, denoted by $\mathbf{Z}(t) \in \{0,1\}^{M \times M}$, encodes the map status at time t, with a value of 0 if the corresponding cell is empty, and 1 if it contains some features. In the first mission profile, cells with a value of 1 correspond to debris, and the AUV must avoid them, while in the second, they correspond to unvisited data nodes, which must be visited before reaching the vessel. The AUV knows its position in the grid at any time, but it can only observe a limited field of view, i.e., the 8 cells directly adjacent to its position. We describe the AUV vision with the matrix $\mathbf{V}_\ell(t)$, whose elements are defined as:

$$\mathbf{V}_\ell(t, \mathbf{x}) = \begin{cases} \mathbf{Z}(t, \mathbf{x}), & \text{if } ||\mathbf{x} - \mathbf{x}_v||_2 < 2; \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

In order to account for the AUV's memory of past observations, the observation $o_\ell(t)$ includes two $M \times M$ matrices, $\hat{\mathbf{Z}}(t) \in \{0,1\}^{M \times M}$ and $\mathbf{\Delta}(t) \in \mathbb{N}^{M \times M}$, as well as the AUV and vessel's positions. The first matrix includes the latest known value for each cell, while the second matrix represents the AoI of those values. In practice, these matrices depend not only on the AUV's actions, but also on the buoys' transmissions. We will define the way they are updated in the following.

After receiving the observation $o_\ell(t) = (\mathbf{X}_\ell(t),\ \mathbf{X}_v(t),\ \hat{\mathbf{Z}}(t),\ \mathbf{\Delta}(t),\ \mathbf{V}_\ell(t)] \in \mathcal{O}_\ell$, the AUV has to decide which actions to take. The AUV action space is given by 4 possible movements: $\mathcal{A}_\mathcal{L} = \{(1,0),\ (0,1),\ (-1,0),\ (-1,0)\}$. Hence, if the AUV performs action $\mathbf{a}_\ell(t) \in \mathcal{A}_\mathcal{L}$ at slot t, its position at the next slot is given by

$$\mathbf{x}_\ell(t+1) = \begin{cases} \mathbf{x}_\ell(t) + \mathbf{a}_\ell(t) & \text{if } \mathbf{x}_\ell(t) + \mathbf{a}_\ell(t) \in \mathcal{M}^*; \\ \mathbf{x}_\ell(t) & \text{otherwise;} \end{cases} \tag{5.7}$$

where $\mathcal{M}^* \subseteq \mathcal{M}$ is the subset of valid positions. This condition implies that the AUV can never exit the map's boundaries or directly over an obstacle. At the end of slot t, the AUV receives a reward $r_\ell(t)$ linked to the achievement of its mission. We will define the reward for the AUV's tasks (i.e., debris avoidance and data muling) in the following.

5.4.2 Buoy Behavior

The goal of the buoys is to support the AUV by selecting and transmitting the pieces of information that are most relevant for its mission. Therefore, the buoys' performance does not directly depend on their actions, but rather on how they affect the AUV's movements. Since the acoustic channel has a constrained bandwidth, we assume that the buoys are not allowed to operate at each slot but only periodically. Practically, two consecutive buoy actions are separated by $T_{\text{period}} \in \mathbb{N}$ slots and the system is in a communication state $s \in \mathcal{S}_c$ only when $\text{mod}(t, T_{\text{period}}) = 0$, where $\text{mod}(\cdot)$ is the integer modulo function.

We assume that each buoy g can sense a specific portion $\mathcal{V}_g \subseteq \mathcal{M}$ of the map, representing its field of view. We encode $\mathcal{V}_g$ in the $M \times M$ binary matrix defined as

$$\mathbf{V}_g(t, \mathbf{x}) = \begin{cases} \mathbf{Z}(t, \mathbf{x}), & \text{if } \mathbf{x} \in \mathcal{V}_g; \\ 0, & \text{otherwise.} \end{cases} \tag{5.8}$$

When the system is in a communication state, i.e., $s(t) \in \mathcal{S}_c$, each buoy $g \in \mathcal{G}$ receives an observation $\mathbf{o}_g(t) = [\mathbf{X}_\ell(t),\, \mathbf{X}_v,\, \hat{\mathbf{Z}}(t),\, \mathbf{\Delta}(t),\, \mathbf{V}_\ell(t),\, \mathbf{V}_g(t)] \in \mathcal{O}_g$. We highlight that $\mathbf{o}_g(t)$ is a super-set of the AUV's observations: each buoy knows everything about the AUV, as well as the status of the map within its own field of vision. In particular, each buoy detects all the packets exchanged on the channel and, therefore, can compute $\hat{\mathbf{Z}}(t)$ using the AUV's requests for information.

We assume the acoustic system to have an extremely low bitrate, limiting the amount of information that each buoy g can transmit to a transmission area $\mathcal{H}_i$. In general, each buoy can choose to transmit one of several sub-areas within its field of vision, and we denote the set of all possible transmission areas as $\mathcal{H}$. We assume that the acoustic channel is error-free, but collisions are destructive, i.e., if multiple buoys choose to transmit, all the packets are lost because of interference. Therefore, after the new observations have been received, the buoys need to coordinate and decide which pieces of information $\mathcal{H}_i \in \mathcal{H}$ to communicate to the AUV. We now encode the information contained in $\mathcal{H}_i$ with the $M \times M$ matrix $\mathbf{H}_i$ defined as

$$\mathbf{H}_i(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \mathcal{H}_i; \\ 0, & \text{otherwise.} \end{cases} \tag{5.9}$$

Hence, we can design how the AUV's estimate of the map and the transmissions' age of information are updated. If the packet on area $\mathcal{H}_i$ is successfully delivered, the matrices $\hat{\mathbf{Z}}(t)$ and $\mathbf{\Delta}(t)$ are updated as:

$$\hat{\mathbf{Z}}(t) = (\mathbf{J} - \mathbf{H}_i) \circ \hat{\mathbf{Z}}(t-1) + \mathbf{H}_i \circ \mathbf{Z}(t), \tag{5.10}$$

$$\mathbf{\Delta}(t) = (\mathbf{J} - \mathbf{H}_i) \circ \left(\mathbf{\Delta}(t-1) + \frac{1}{T} \right), \tag{5.11}$$

where $\mathbf{J}$ is an $M \times M$ all-ones matrix and $\circ$ represents the Hadamard product.

We can obtain multiple working configurations by varying the number of buoys, their fields of view, the transmission areas' sizes, and the medium access technique used to coordinate. In this work, we focus on two different communication scenarios, namely, *centralized* and *distributed control*.

- In the centralized control scenario, a single buoy $g \in \mathcal{G}$ sees the full map, i.e., $\mathcal{V}_g = \mathcal{M}$. These settings represent the case in which the buoys can freely communicate over the air and collectively decide which of them will send an update to the AUV through the underwater acoustic channel. However, packets still have a limited size, so the transmitting

buoy needs to decide which transmission area $\mathcal{H}_i \in \mathcal{H}$ to send. Therefore, $\mathcal{A}_\mathcal{G} = \mathcal{H}$ and the number of possible buoy actions is $H = |\mathcal{H}|$. This configuration is easy to manage, as communication decisions are centralized and packet collisions are impossible, i.e., the communication is always successful.

- In the distributed control scenario, there are multiple buoys, each of which is assigned to a different transmission area $\mathcal{H}_g$, so that the number $G = |\mathcal{G}|$ of buoys is equal to $H = |\mathcal{H}|$. In particular, each buoy can perceive only the sub-area which it is associated with, so that we have $\mathbf{V}_g(t) = \mathbf{H}_g \circ \mathbf{Z}(t),\ \forall\ g \in \mathcal{G}$. If the system is in a communication state, each buoy can independently decide whether to transmit information on its map portion or remain silent; therefore, the buoy action space is $\mathcal{A}_\mathcal{G} = \{0, 1\}$. Since the buoys act simultaneously, it is challenging to find a communication policy that allows the AUV to receive enough information while minimizing the risk of packet collisions. In particular, each buoy is not aware of the information sensed by the others, which implies that the same observation $o_g(t)$ may be associated with different system states and, consequently, different optimal actions.

In the centralized control scenario, the reward $r_g(t)$ for each buoy $g \in \mathcal{G}$ depends solely on the AUV's behavior:

$$r_g(t) = \sum_{\tau=t}^{t+T_{\text{period}}} r_\ell(\tau), \tag{5.12}$$

where T_{period} is the number of slots within two subsequent buoy actions. Instead, in the distributed control scenario, each buoys $g \in \mathcal{G}$ receives a penalty whenever its actions cause a packet collision:

$$r_g(t) = \begin{cases} -\eta, & \text{if } a_g(t) \neq 0 \wedge \sum_{j\in\mathcal{G}} a_j(t) \geq 2; \\ \sum_{\tau=t}^{t+T_{\text{period}}} r_\ell(\tau), & \text{otherwise,} \end{cases} \tag{5.13}$$

where $\eta \in \mathbb{R}^+$ is a constant penalty term.

5.4.3 Debris Avoidance

In the debris avoidance scenario, there are several obstacles in the marine environment, such as rocks, sea mines, or oil installations, and the AUV must reach the vessel as fast as possible without crashing into them. In general, the debris locations change in time (accounting for currents and drift), and the set $\mathcal{Z}(t)$ in this scenario represents the map state at slot t, including only the cells of $\mathcal{M}$ that are occupied by obstacles and debris. The elements of the corresponding matrix $\mathbf{Z}(t)$ are equal to 1 if the corresponding cell is occupied by debris and to 0 otherwise:

$$\mathbf{Z}(t, \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \mathcal{Z}(t); \\ 0, & \text{otherwise.} \end{cases} \tag{5.14}$$

We assume that the AUV cannot enter the area occupied by the debris. To represent this, we define the set of valid positions at slot t, $\mathcal{M}^*(t)$, as follows:

$$\mathcal{M}^*(t) = \mathcal{M} \setminus \mathcal{Z}(t). \tag{5.15}$$

The outcomes of the AUV's actions are then determined by (5.7), using $\mathcal{M}^*(t)$ as the valid position subset.

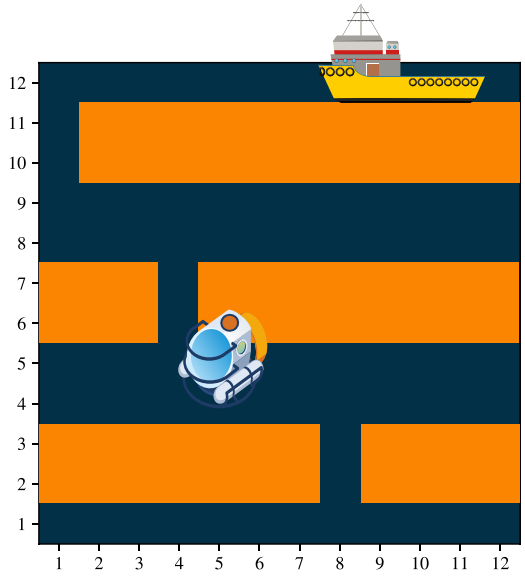


Fig. 5.2: Debris avoidance scenario.

We consider a scenario in which the obstacles are arranged in $Z_{\text{debris}} = 3$ horizontal lines on the map, as shown in Fig. 5.2. Each line of debris has a single opening, whose location can change over time. We denote the coordinates of the free passages on the x-axis at slot t by $x_0(t)$, $x_1(t)$, and $x_2(t)$, respectively. Hence, at the beginning of each slot t, the set $\mathcal{Z}(t)$ contains the elements (x, y)

$$\mathcal{Z}(t) = \bigcup_{z=0}^{Z_{\text{debris}}-1} (\{0, \dots, M-1\} \setminus \{x_z(t)\}) \times \mathcal{Y}_z, \tag{5.16}$$

where $\mathcal{Y}_z$ is the set containing the coordinates of the z-th debris line on the y-axis, and we have:

$$\mathcal{Y}_0 = \{2, 3\}, \mathcal{Y}_1 = \left\{\frac{M}{2}, \frac{M}{2} + 1\right\}, \mathcal{Y}_2 = \{M-2, M-1\}. \tag{5.17}$$

At the beginning of each episode, the free passages' coordinates are generated from a discrete uniform distribution in $\{0, \dots, M-1\}$. The locations $x_0(t)$, $x_1(t)$, and $x_2(t)$ are then updated following the same periodicity as the buoys' communication. Hence, at the beginning of any slot t such that $\text{mod}(t, T_{\text{period}}) = 0$, the new values of $x_0(t)$, $x_1(t)$, and $x_2(t)$ are picked from

the distribution

$$P(x_z(t)=x)=\frac{(M-|x-x_z(t-1)|)^2}{\sum_{q=1}^{M}(M-|q-x_z(t-1)|)^2},\quad z\in\{0,1,2\}. \tag{5.18}$$

In this scenario, the reward $r_\ell(t)$ needs to be tuned to encourage the AUV to avoid the debris and finally reach the vessel. In particular, the AUV receives a reward $\rho\in\mathbb{R}^+$ when it reaches the surface vessel, i.e., if $\mathbf{x}_\ell(t+1)=\mathbf{x}_v$. We also assign a smaller, positive intermediate reward $\sigma<\rho$ if the AUV makes progress its task, i.e., in one of these cases:

1. If $\mathbf{a}_\ell(t)=(1,0)\vee\mathbf{a}_\ell(t)=(-1,0), |\mathbf{x}_\ell(t)^{(1)}+\mathbf{a}_\ell(t)^{(1)}-x_z(t)|<|\mathbf{x}_\ell(t)^{(1)}-x_z(t)|$, and $\mathbf{x}_\ell(t)^{(2)}+1\in\mathcal{Y}_z$, i.e., if the AUV has reached a debris line, and moves horizontally towards a free passage;

2. If $\mathbf{a}_\ell(t)=(1,0)\vee\mathbf{a}_\ell(t)=(-1,0), |\mathbf{x}_\ell(t)^{(1)}+\mathbf{a}_\ell(t)^{(1)}-\mathbf{x}_v^{(1)}|<|\mathbf{x}_\ell(t)^{(1)}-\mathbf{x}_v^{(1)}|$, and $\mathbf{x}_\ell(t)^{(2)}=N$, i.e., if the AUV is on the top row of the map, and moves horizontally towards the vessel;

3. If $\mathbf{a}_\ell(t)=(0,1)$, and $\mathbf{Z}(t,\mathbf{x}_\ell(t)+\mathbf{a}_\ell(t))=0$, i.e., if the AUV moves upward in the map, without entering in the area occupied by the debris.

We can then summarize these conditions in the Boolean function $\Phi(\mathbf{x}_\ell(t),\mathbf{x}_v,\mathbf{a}_\ell(t),\mathcal{Z}(t))$, which returns a true value if one of the above conditions is verified. Hence, the AUV reward $r_\ell(t)$ at slot t only depends on its position $\mathbf{x}_\ell(t)$, the surface vessel's position $\mathbf{x}_v$, the robot action $\mathbf{a}_\ell(t)$, and the debris positions $\mathcal{Z}(t)$:

$$r_\ell(t)=\begin{cases}\rho, & \text{if } \mathbf{x}_\ell(t)+\mathbf{a}_\ell(t)=\mathbf{x}_v(t);\\ \sigma, & \text{if } \Phi(\mathbf{x}_\ell(t),\mathbf{x}_v(t),\mathbf{a}_\ell(t),\mathcal{Z}(t));\\ 0, & \text{otherwise.}\end{cases} \tag{5.19}$$

Once the robot reaches the vessel, the episode is over; the long-term reward function is an incentive for the AUV to accomplish its task as fast as possible. At the same time, we exploit the intermediate reward σ to simplify the learning problem and reduce the reward's delay, which can often be problematic for the convergence of RL algorithms.

5.4.4 Data Muling

In the data muling scenario, the is a set of that the AUV has to visit $Z_{\text{node}}=2$ underwater floating nodes to recover their data through high-throughput short-range optical communications [163] before reaching the vessel to upload the data. Each node occupies a single cell and can change its location in time: we denote by $\mathcal{Z}(t)$ the set of cells occupied by the unvisited nodes at slot t. The elements of the corresponding matrix $\mathbf{Z}(t)$ are equal to 1 if the corresponding cell

is occupied by an unvisited node and to 0 otherwise:

$$\mathbf{Z}(t, \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \mathcal{Z}(t); \\ 0, & \text{otherwise.} \end{cases} \tag{5.20}$$

We observe that the AUV is free to move within the environment, and the only illegal actions are those that would bring it outside of the map borders, so that the set $\mathcal{M}^* = \mathcal{M}$ in (5.7). We now denote the coordinates of the z-th node at slot t by $\mathbf{x}_z(t) \in \mathcal{Z}(t)$. The total number of nodes is maximum (i.e., $|\mathcal{Z}(t)| = Z_{\text{node}}$) at the beginning of each episode, and decreases as the AUV visits each node. To visit a node, the AUV needs to move to its position. Hence, if $\mathbf{x}_\ell(t) = \mathbf{x}_z(t)$ at any slot t, the z-th node is removed from $\mathcal{Z}(t+1)$.

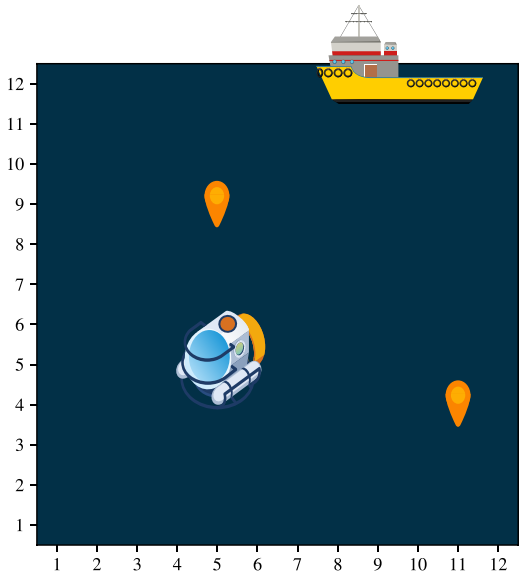


Fig. 5.3: Data muling scenario.

At the beginning of each episode, the nodes' coordinates are generated from a discrete uniform distribution in $\mathcal{M}$. Then, $\mathbf{x}_z(t) \in \mathcal{Z}(t)$ change values following the same periodicity as the buoys' communications. Whenever $\text{mod}(t, T_{\text{period}}) = 0$, the node's coordinates are updated independently and according with the same distribution:

$$P(x_z(t+1)^{(i)} = x) = \frac{(M - |x - x_z(t)^{(i)}|)^2}{\sum_{q \in \{0,\dots,M-1\}} (M - |q - x|)^2} \quad \forall i \in \{1, 2\}, z \in \{1, 2\}. \tag{5.21}$$

A picture of the considered scenario, representing the floating nodes, the AUV, and the surface vessel, is given in Fig. 5.3.

In this scenario, the reward $r_\ell(t)$ is designed to make the AUV visit all nodes, then reach the vessel in the shortest possible time. The AUV receives a reward $\rho \in \mathbb{R}^+$ if it has already visited all the nodes and reaches the surface vessel, i.e., if $|\mathcal{Z}(t)| = 0 \wedge \mathbf{x}_\ell(t+1) = \mathbf{x}_v$. Additionally, we

assign a positive reward $\sigma < \rho$ if the AUV advances in performing its task, i.e., in one of the following cases:

1. If $|\mathcal{Z}(t)| = 2$, $|\mathbf{x}_\ell(t)+\mathbf{a}_\ell(t)-\mathbf{x}_z(t)| < |\mathbf{x}_\ell(t)-\mathbf{x}_z(t)|$, and $|\mathbf{x}_\ell(t)-\mathbf{x}_{3-z}(t)|+|\mathbf{x}_{3-z}(t)-\mathbf{x}_v| < |\mathbf{x}_\ell(t) - \mathbf{x}_{3-z}(t)| + |\mathbf{x}_{3-z}(t) - \mathbf{x}_v|$, with $z \in \{1, 2\}$, i.e., if both nodes are still unvisited, the AUV is moving towards $\mathbf{x}_z(t)$, and it is more convenient to visit the node in $\mathbf{x}_z(t)$ to end the mission in the shortest time;

2. If $|\mathcal{Z}(t)| = 1$ and $|\mathbf{x}_\ell(t) + \mathbf{a}_\ell(t) - \mathbf{x}_1(t)| < |\mathbf{x}_\ell(t) - \mathbf{x}_1(t)|$, i.e., if only a single node is still unvisited and the AUV is moving towards its position;

3. If $|\mathcal{Z}(t)| = 0$ and $|\mathbf{x}_\ell(t) + \mathbf{a}_\ell(t) - \mathbf{x}_v| < |\mathbf{x}_\ell(t) - \mathbf{x}_v|$, i.e., if all the nodes have been visited and the AUV is moving towards the vessel.

We can then summarize these conditions in the Boolean function $\Psi(\mathbf{x}_\ell(t), \mathbf{x}_v, \mathbf{a}_\ell(t), \mathcal{Z}(t))$, which returns a true value if one of the above conditions is verified. Hence, the AUV reward $r_\ell(t)$ at slot t depends only on its position $\mathbf{x}_\ell(t)$, the surface vessel's position $\mathbf{x}_v$, the action $\mathbf{a}_\ell(t)$, and the set $\mathcal{Z}(t)$ containing the locations of the unvisited nodes :

$$r_\ell(t) = \begin{cases} \rho, & \text{if } |\mathcal{Z}(t)| = 0 \wedge \mathbf{x}_\ell(t) + a_\ell(t) = \mathbf{x}_v; \\ \sigma, & \text{if } \Psi(\mathbf{x}_\ell(t), \mathbf{x}_v, \mathbf{a}_\ell(t), \mathcal{Z}(t)); \\ 0, & \text{otherwise.} \end{cases} \tag{5.22}$$

As in the previous mission, we use the intermediate reward σ to facilitate the learning process of the AUV. Finally, we highlight that a more accurate estimate of $\mathbf{Z}(t)$ allows the AUV to complete its task in a shorter times in both the considered scenarios.

5.5 Learning Strategy

To determine the best policy for managing our system, we exploit a DRL architecture that combines RL with Neural Networks (NNs). The RL paradigm is a mathematical framework that enables the resolution of decision-making problems. Instead, NNs are a set of computational tools that make it possible to approximate complex functions from experience samples. We give more details about RL and its combination with NNs in Appendix B.

5.5.1 Neural Network Architecture

In this work, we need to learn multiple quality functions $Q(\cdot)$, one for each agent in $\mathcal{L} \cup \mathcal{G}$. As explained in the Appendix, the function $Q(\cdot)$ returns the best action $a^* \in \mathcal{A}$ for each state observation $o \in \mathcal{O}$, leading the agent to the optimal policy. In the centralized control scenario, we define two quality functions, named $Q_\ell(\cdot)$ and $Q_g(\cdot)$, which determine the movements of the AUV and the buoys' communication strategy, respectively. On the other hand, in the

Tab. 5.2: Agent architecture.

Agent	Layer							
	Input (convolutional)		Hidden (convolutional)		Hidden (convolutional)		Output (linear)	
AUV	Input channel	5	Input channel	64	Input channel	64	Input neurons	64
	Output channel	64	Output channel	64	Output channel	64		
	Kernel	(3,3)	Kernel	(3,3)	Kernel	(3,3)	Output neurons	4
	Stride	1	Stride	(4,4)	Stride	(4,4)		
Buoy	Input channel	6	Input channel	64	Input channel	64	Input neurons	64
	Output channel	64	Output channel	64	Output channel	64		
	Kernel	(3,3)	Kernel	(3,3)	Kernel	(3,3)	Output neurons	$\lvert\mathcal{A}_\mathcal{G}\rvert$
	Stride	1	Stride	(4,4)	Stride	(4,4)		

distributed control scenario, we have a total of $1 + |\mathcal{G}|$ quality functions, i.e., one managing the AUV and an additional function for each of the buoys.

In the case of the AUV, the function $Q(\cdot)$ is approximated by a Convolutional Neural Network (CNN) with 4 output neurons, i.e., one for each possible AUV action. The CNN input is a convolutional layer that handles the AUV observation as an $M \times M$ image with 5 channels. The input channels are given by the matrices $\mathbf{X}_\ell(t)$, $\mathbf{X}_v$, $\hat{\mathbf{Z}}(t)$, $\mathbf{\Delta}(t)$, and $\mathbf{V}_\ell(t)$ contained in the AUV's observation. Therefore, at each slot t, the neural network takes $\mathbf{o}_\ell(t)$ as its input information and returns the expected value of each action $a \in \mathcal{A}_\mathcal{L}$ as output.

The function $Q(\cdot)$ of each buoy $g \in \mathcal{G}$ is instead approximated by a CNN using a convolutional layer with 6 channels as input, and a linear layer with $|\mathcal{A}_\mathcal{G}|$ neurons as output. In this case each buoy observation is treated as an $M \times M$ image with 5 channels, given by the matrices $\mathbf{X}_\ell(t)$, $\mathbf{X}_v$, $\hat{\mathbf{Z}}(t)$, $\mathbf{\Delta}(t)$, $\mathbf{V}_\ell(t)$, and $\mathbf{V}_g(t)$ containing the buoy's observation. At each slot t, the described system takes $\mathbf{o}_g(t)$ as input information and returns the quality of each action $a \in \mathcal{A}_\mathcal{G}$ as output.

The network architectures for the AUV and buoys have four convolutional layers followed by two fully connected linear layers. We consider the Rectifier Linear Unit (ReLU) activation function for the hidden layers because of its efficiency in representing non-linear dynamics, while we implement a linear function in the output layer. The settings of the learning architectures are summarized in Tab. 5.2.

5.5.2 Training Framework

The training of the learning agents can follow two different approaches, namely Networked Control (NC) and Joint Communication and Control (JCC). In the NC scenario, the buoys follow a pre-determined communication strategy. In other words, the AUV considers the buoys as part of the learning environment and adapts its policy to the buoys' behavior. Hence, the learning environment is more stable, but the final performance is constrained by the buoys' initial policy. Hence, in the NC scenario, only the AUV's NN is trained for a total number of N_{train} episodes, where each episode lasts t slots.

In the JCC scenario, both the AUV and the buoys are controlled by learning agents, and the system training follows an iterative approach. This is a common practice in MARL, as multiple

agents exploring the environment and improving their policies at the same time can prevent convergence: by only training one kind of agent at a time, we ensure that the environment that agent sees is Markovian. The training consists in N_{round} rounds, each of which includes two phases: the first lasts N^{ℓ}_{train} episodes and is dedicated to the AUV training, while the second lasts N^{g}_{train} episodes, and is dedicated to the buoy training. Once this iterative training phase is over, we train the AUV architecture again for N^{ℓ}_{train} episodes. Hence, the total number of AUV training episodes is $N^{\ell}_{\text{train}} \times (N_{\text{round}} + 1)$, while the buoys' is $N^{g}_{\text{train}} \times N_{\text{round}}$.

During the first phase of the first round, the buoys are associated with a pre-determined strategy, as occurs in the NC scenario. Hence, the AUV is trained for N^{ℓ}_{train} episodes, learning how to perform its task optimally while adapting to the buoy policy. After this, the weights of the AUV's NN are fixed, while those of the buoys' NNs are initialized from scratch. Then, we train the buoys for N^{g}_{train} episodes, learning how to adapt the communication process to the AUV's actions. In the subsequent round, the weights of the buoys' NNs are fixed while the AUV's NN are reinitialized and trained once again. The same pattern is repeated for each round, allowing the learning agents to adapt to each other progressively . The described framework makes it possible to fully explore the learning environment, while ensuring sufficient robustness in the training phase, and is commonly used in MARL problems.

5.6 Simulation Settings and Results

In this section, we implement our CP-POMDP model in two underwater scenarios described in Sec. 5.4. First, we describe the benchmark strategies against which our learning framework is tested and present the simulation settings. Finally, we analyze the system performance in the two test scenarios, highlighting the benefits and drawbacks of the proposed JCC approach.

5.6.1 Benchmark Strategies

As benchmarks for our model, we consider three learning systems trained according to the NC approach presented in Sec. 5.5.2, implementing different algorithms to manage the communication process of the buoys. All these strategies are based on the centralized control scenario described in Sec. 5.4, in which we have only one buoy viewing the entire map so that there is no risk of packet collisions. In practice, at each communication slot, the benchmarks manage the buoy's transmissions as follows:

- Random Communication (RC): the buoy transmits a random sub-area $\mathcal{H}_i \in \mathcal{H}$;
- Closest Communication (CC): the buoy transmits the sub-area $\mathcal{H}_i$ closest to the AUV;
- Oracle Communication (OC): the buoy transmits the entire map at each slot, ensuring that $\hat{\mathbf{Z}}(t) = \mathbf{Z}(t)$, $\forall\ t$.

We highlight that the OC strategy represents an upper bound for the performance our system, as it models a case in which there is no communication bottleneck and the AUV is always aware of the real map state.

Tab. 5.3: Simulation settings.

Parameter	Value	Description	Parameter	Value	Description
L	1	Number of AUVs	G	$\{1, 9\}$	Number of buoys
$M \times M$	12×12	Map size	T	100	Steps per episode
T_{period}	5	Inter-transmission period	H	9	Number of transmission areas
Z_{debris}	3	Number of debris rows	Z_{node}	2	Number of underwater nodes
ρ	1.0	Final reward	σ	0.22	Intermediate reward
λ	0.95	Discount factor	ζ	10^{-5}	Maximum learning rate
N_{round}	2	Training rounds	N_{test}	10^4	Testing episodes
N^{ℓ}_{train}	10^5	Training episodes for the AUV	N^{g}_{train}	$5 \cdot 10^5$	Training episodes for the buoys

The above strategies are tested against two configurations of our JCC model. The first, named Centralized JCC (CJCC), is based on the centralized control settings given in Sec. 5.4.2. The other, named Distributed JCC (DJCC), makes use of multiple buoys, as in the distributed control scenario given in Sec. 5.4.2. We observe that the DJCC approach is more challenging, since buoys communicate independently and packet collisions are possible.

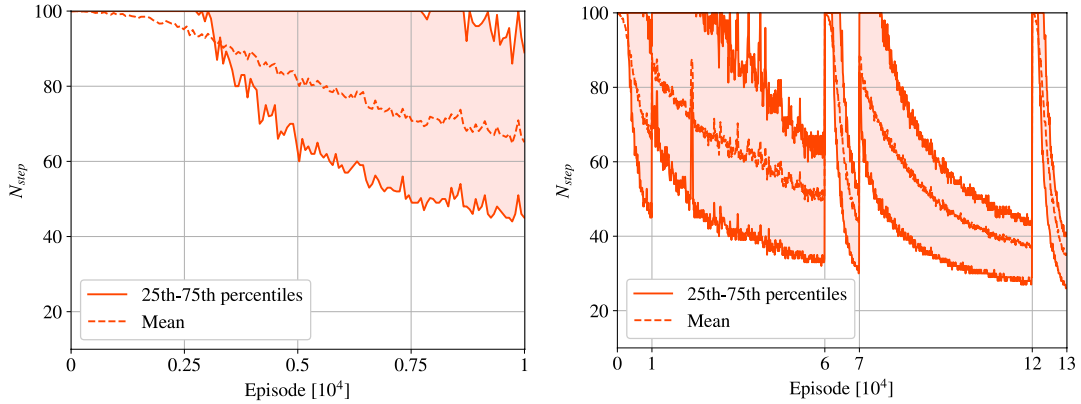
5.6.2 System Settings

We consider the reference model described in Sec. 5.4, with a map of size $M = 12$, divided into $H = 9$ different non-overlapping portions. Hence, in the centralized control scenario (used by the benchmarks and the CJCC system), there is a single buoy with 9 possible actions. Instead, DJCC considers 9 different buoys, each of which can perceive and transmit a specific portion of the map. We set the period between two consecutive communication states to $T_{\text{period}} = 5$ slots and the maximum episode duration to $T = 100$ slots. If the AUV spends more than 100 slots in the map without finishing the task, it fails to achieve its goal and the episode ends. If the AUV reaches the surface vessel with the data, it receives a reward $\rho = 10$, while the collision penalty parameter is set to $\eta = 1$. The intermediate reward σ instead varies over time and increases as the AUV approaches the end of its mission.

When training the benchmark strategies, we consider a unique training phase of $N_{\text{train}} = 100000$ episodes. On the other hand, the training of the JCC systems includes $N_{\text{round}} = 2$ rounds, where the AUV training phases last $N^{g}_{\text{train}} = 100000$ episodes each, and those of the buoys $N^{g}_{\text{train}} = 500000$ episodes each. We observe that N^{g}_{train} is higher than N^{ℓ}_{train}, since the buoys take actions less frequently than the AUV and, consequently, need more episodes to reach convergence. During a single training round, the agents use an *ε-greedy policy* to explore the observation-action space. The policy selects the best possible action with probability $1 - \varepsilon$ and a random suboptimal action with probability ε, with the value of ε slowly decreasing from 0.9 to 0.1. Finally, we set the discount factor to $\lambda = 0.95$ and we exploit the Adaptive moment estimation (Adam) algorithm to optimize the NN architectures [164], considering $\zeta = 0.00001$ as the maximum learning rate. All the simulation parameters are reported in Tab.5.3.

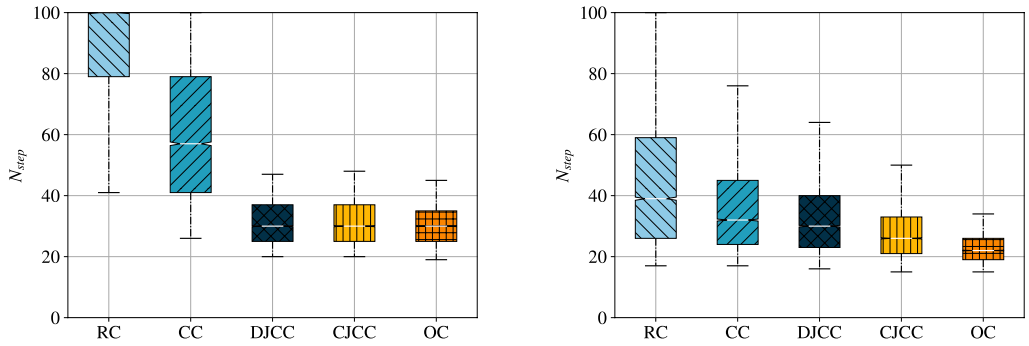
5.6.3 Performance Analysis

As the main performance metric of our experiments, we consider the number N_{step} of steps that the AUV takes to perform its task. Particularly, if the AUV is not able to achieve its goal before the episode ends, we have that $N_{\text{step}} = T$, which is the maximum episode duration.



(a) CC strategy. **(b)** CJCC strategy.

Fig. 5.4: Training phase (debris avoidance).

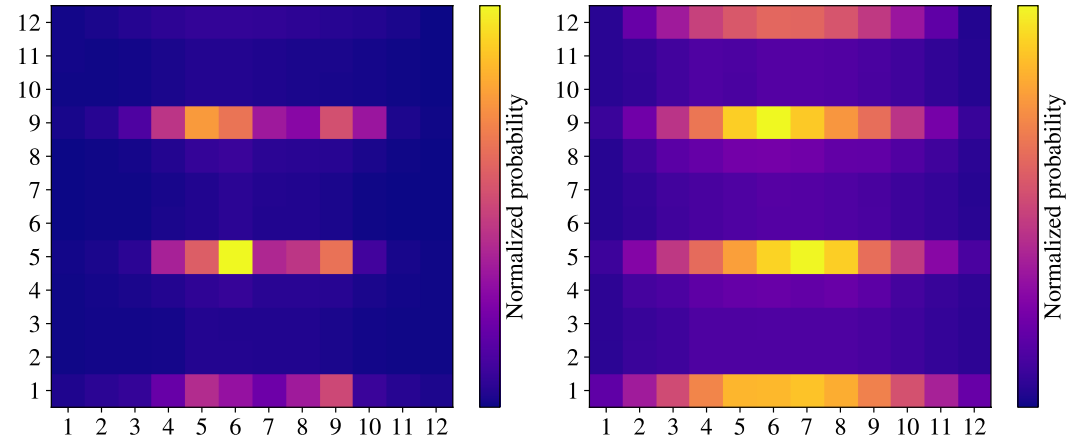


(a) Debris avoidance. **(b)** Data muling.

Fig. 5.5: Performance of the different strategies.

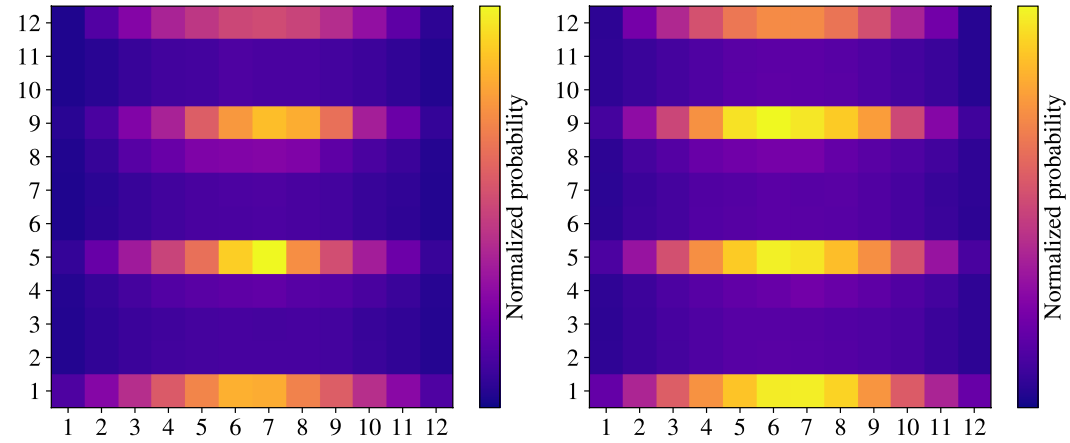
Fig. 5.4 shows the distribution of N_{step} during the training of the CC (above) and CJCC (below) strategies in the ostacle avoidance scenario. We can observe that the mean value and quartiles of N_{step} decrease almost monotonically for CC. Indeed, at the beginning of the training, the AUV chooses actions randomly (as a consequence of the ε-greedy policy) and improves its policy as the number of episodes increases. On the other hand, in case of CJCC, the distribution of N_{step} exhibits a regular pattern, where the average of N_{step} repeatedly drops from 100 to about half of that value. This is because the NNs are periodically reset after each round of the iterative training, making the AUV and the buoys forget the policy learned in the past.

At the same time, as more training rounds are completed, the average values of N_{step} become significantly lower, since the agents have progressively learned to adapt to each other.



(a) CC strategy.

(b) DJCC strategy.



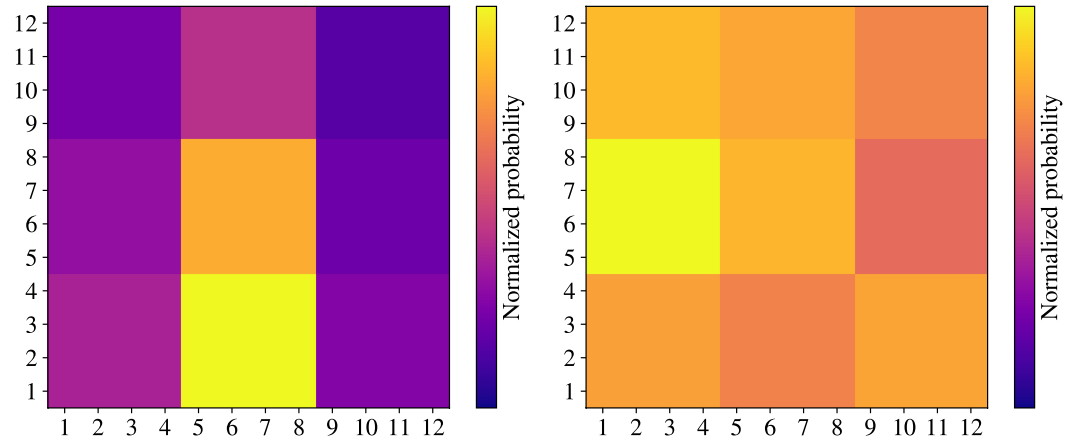
(c) CJCC strategy.

(d) OC strategy.

Fig. 5.6: AUV location probability (debris avoidance).

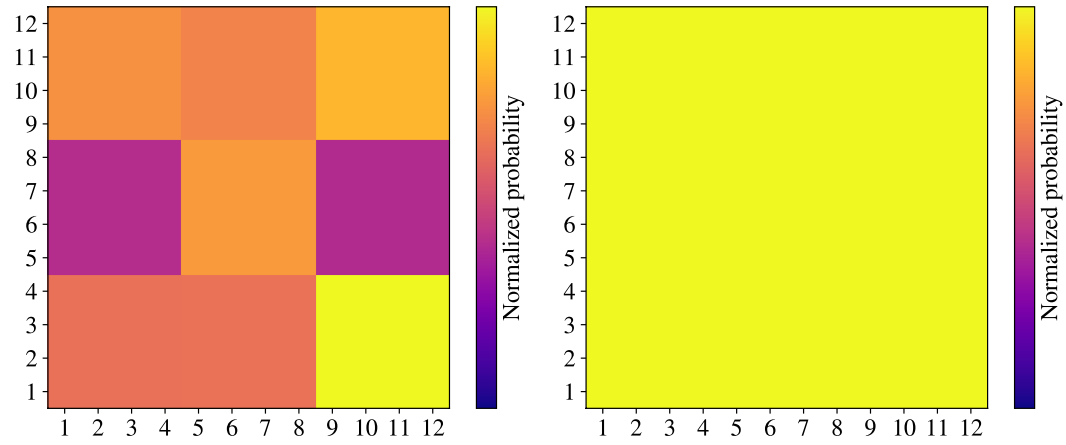
To better evaluate the system performance, we carry out additional $N_{\text{test}} = 10000$ episodes for each strategy. The distribution of N_{step} during the test phase for the two considered AUV tasks is presented as a boxplot in Fig. 5.5. The whiskers represent the 5-th and 95-th percentiles of the distribution, the edges the 25-th and the 75-th percentiles, and the white line in the middle of each box is the distribution median.

As expected, OC outperforms all other strategies, with the lowest values of N_{step}. In the obstacle avoidance scenario, the AUV accomplishes its goal in less than 30 steps in half of the testing episodes, while the distribution median decreases to 22 steps in the data muling case. This is because the OC scheme allows the AUV to perfectly know the environment, making the problem trivial to solve. On the opposite extreme, the RC strategy leads to the worst results,



(a) CC strategy.

(b) DJCC strategy.



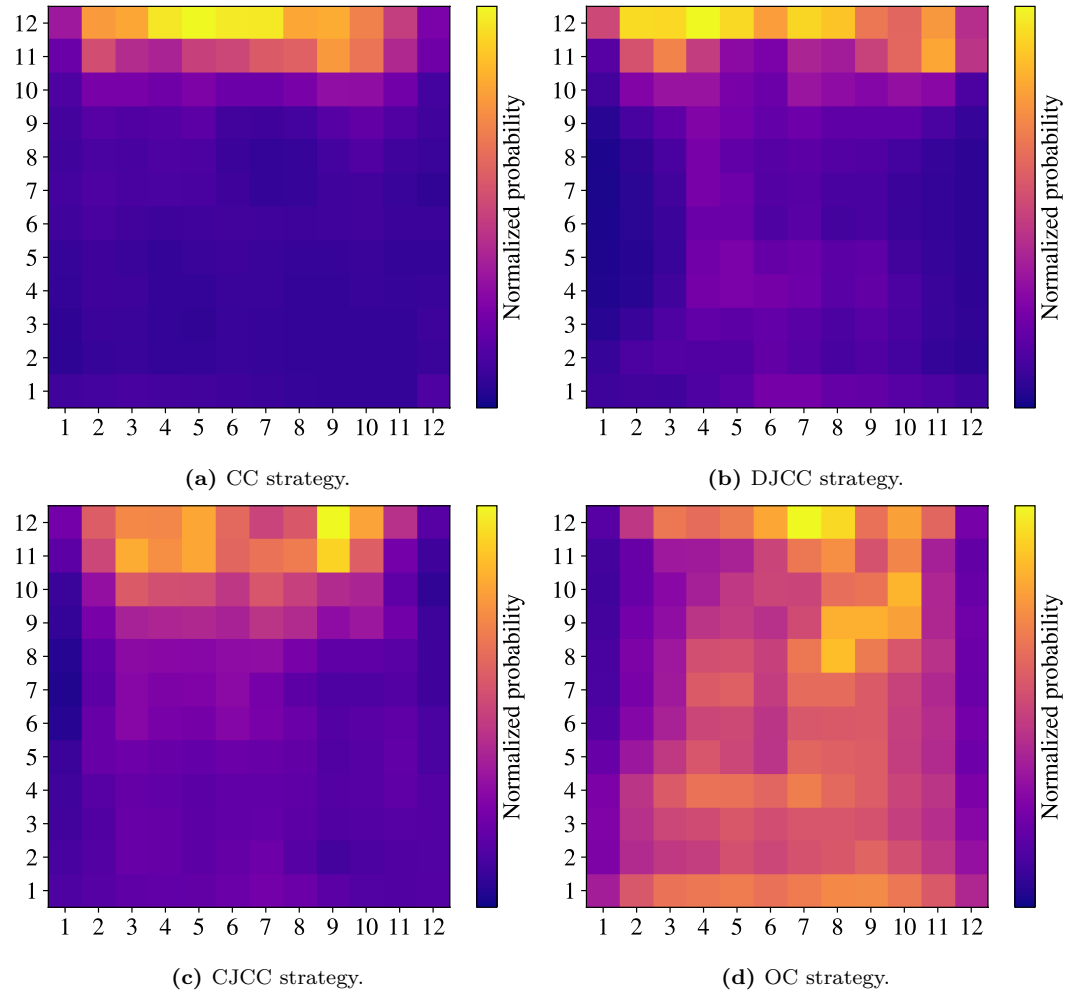
(c) CJCC strategy.

(d) OC strategy.

Fig. 5.7: Sensor transmission probability (debris avoidance).

as the buoy transmissions do not take into consideration the relevance of the information to the AUV when choosing what to transmit. This leads to a median scenario duration of 39 steps in the data muling scenario, while the AUV fails more than 50% of the test episodes in the obstacle avoidance scenario, not reaching the vessel even after 100 steps. Finally, the CC scheme has an intermediate performance, taking 32 steps to finish the median episode in the data muling scenario and 57 in the obstacle avoidance scenario.

However, the benchmark strategies are significantly outperformed by the JCC schemes, which can almost get the same performance as the ideal communication setup. In particular, both DJCC and CJCC can achieve close to optimum performance in the obstacle avoidance mission, coordinating communications so as to always deliver the most relevant information to the AUV. On the other hand, the data muling scenario is more difficult, and even more so for a distributed setup: as buoys often only know the position of one of the two nodes, the risk of collisions



(a) CC strategy.

(b) DJCC strategy.

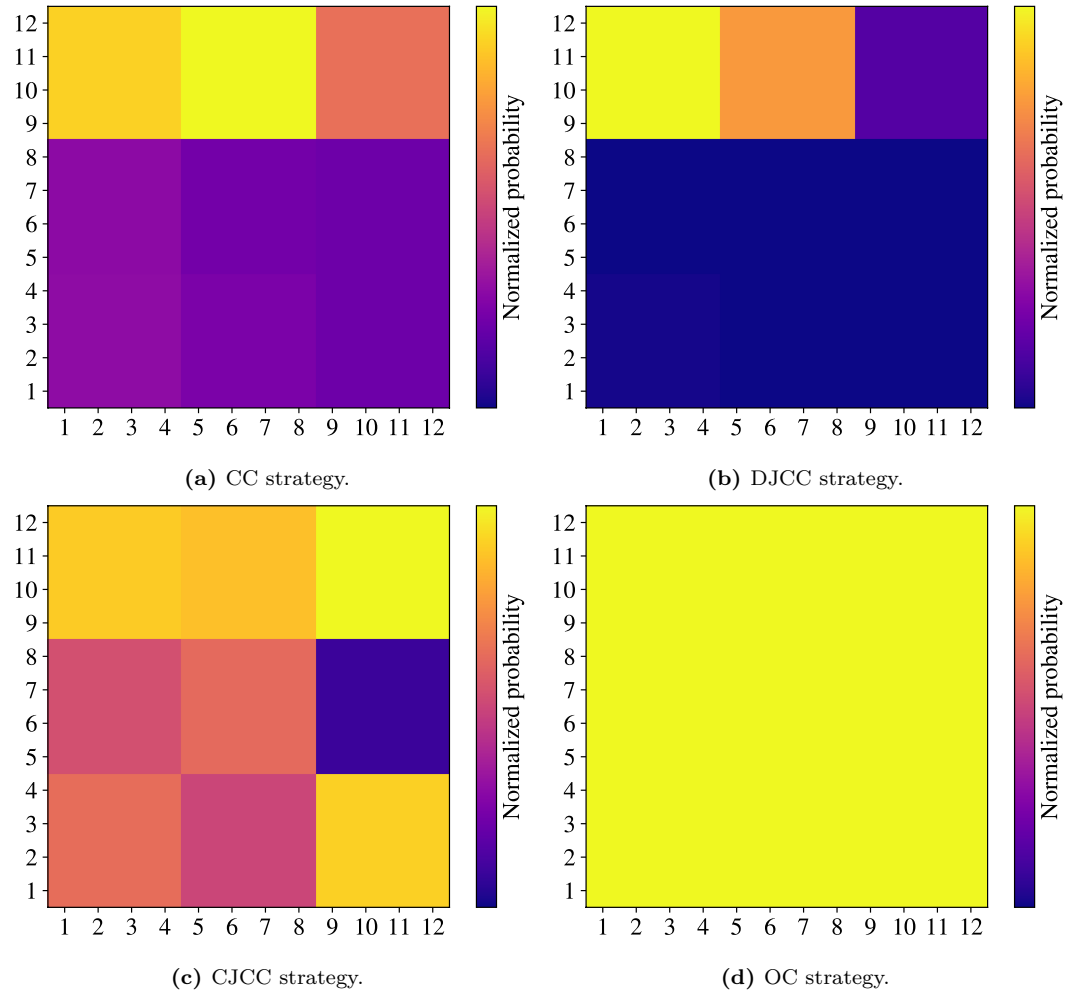
(c) CJCC strategy.

(d) OC strategy.

Fig. 5.8: AUV location probability.

is high, and the buoys will tend to behave more conservatively. In the latter scenario, the median value of N_{step} is 26 and 30 for CJCC and DJCC, respectively, which is still significantly lower than the benchmarks. This is even more noticeable when looking at the worst-case performance, represented by the upper whiskers of the boxplot: the difference between the schemes is extremely stark in this case, showing the benefits of joint control and communications and of having centralized knowledge of the state of the environment.

To better illustrate the agents' behavior, Fig. 5.6 shows the distribution of the AUV position in the obstacle avoidance scenario with different control strategies. The distribution is presented as a colormap: the lighter the cell color, the higher the probability of the AUV to be in the corresponding position during the testing phase. We can notice that the AUV tends to keep close to the center of the map when using the CC scheme, never straying too far to the left or right: on the other hand, the JCC strategies have a distribution more similar to the OC case.



(a) CC strategy.

(b) DJCC strategy.

(c) CJCC strategy.

(d) OC strategy.

Fig. 5.9: Sensor transmission probability.

This is due to the influence of transmissions from the buoys: if the AUV can know where the opening is, it can move directly toward it, instead of staying close to the center of the map and then gradually exploring. While this is a relatively simple example, and constructing a better transmission heuristic might be possible, the joint design can optimize both strategies jointly without any additional design requirements.

We can see the same pattern in Fig. 5.7, which shows the transmission patterns for the buoys: while the buoys closer to the center of the map transmit much more often when using the CC scheme, buoys that actually have relevant information are much more active in the two JCC schemes, helping the AUV reach the target faster.

Finally, Figs. 5.8–5.9 show the distribution of the AUV positions and the buoys' transmissions in the data muling scenario. As for the previous case, we can appreciate that the AUV takes more complex trajectories in the case of the JCC approaches (Figs. 5.8b–5.8c) than when using

the CC strategy (Figs.5.8a). In this case, there is still a gap between the JCC strategies and the OC scheme, as coordination is much harder, and we can see that the trajectories for CJCC are much more evenly distributed than for DJCC. This is explained by the transmission distribution in Figs. 5.9b–5.9c: while centralized communications can always deliver relevant information, the buoys in the DJCC scenario cannot coordinate effectively when the AUV is in the lower portion of the map, i.e., at the beginning of the task, when neither node has been visited. In order to avoid collisions, which are heavily penalized by the reward function, the buoys tend to only transmit when the AUV is already close to the target, helping it reach the second node when necessary. This also explains the performance gap between CJCC and DJCC, as both the transmission and location distribution maps for DJCC are in between CC and CJCC.

5.7 Conclusions and future work

In this chapter, we developed a new framework, named CP-POMDP, to model the communication and physical actions of multiple agents operating in the same learning environment. The proposed model is extremely flexible and can represent a plethora of practical applications with different complexity levels. We tested the described framework in a reference scenario, in which an AUV needs to complete a mission in an underwater environment, and a set of buoys can transmit information about the map. Our simulations show that the joint training of the buoys and the AUV, based on our CP-POMDP model, significantly outperforms classical networked control strategies.

Future work on the subject might involve more complex scenarios with multiple robots and sensors, in which information is distributed across the agents, none of which has a full view of the environment. This step leads to novel machine learning problems, where agents have to estimate not just the state but also the knowledge of the others, and to game theoretical issues since the agent behavior follows game theory principles. In particular, if agents have conflicting objectives, it may be beneficial to transmit false or misleading information, introducing questions of trust and reciprocity to the problem.

Part III

Intelligent Resource Management in Dynamic Networks

Introduction

When considering novel telecommunication scenarios with mutable requirements and resource availability, it is necessary to carefully allocate the resources among the many applications running over the system. In this context, a promising solution is to exploit the Network Slicing (NS) paradigm, which enables the virtualization of physical resources and the definition of multiple logical networks, named slices, each designed to support a specific service class. Traditional approaches based on static protocols are inadequate to manage NS scenarios, while advanced learning strategies have exhibited high-quality results in this field. On the other hand, learning algorithms in the network edge involve a high cost in terms of bandwidth and computational resources, which may lead to overestimating the actual system performance.

In the following chapters, we complete the vision defined in this thesis, attacking the problem of allocating network resources among different service classes. To this end, we exploit Deep Reinforcement Learning (DRL) and design a distributed architecture where multiple learning agents cooperate to orchestrate the different network components. We can naturally integrate this system with the communication and control strategies previously designed, offering a comprehensive architecture for managing dynamic networks. By means of extensive simulations, we prove that our approach yields better performance than static resource allocation and an efficient empirical strategy. At the same time, the proposed system shows high adaptability to different working conditions without additional training, ensuring high performance even in the case of mutable network topology.

As the last step of this research, we analyze the cost of deploying learning algorithms in telecommunication networks and define an optimization problem in which training a DRL agent makes it possible to improve network management but also reduces the current resources. The designed model is highly flexible and can suit any resource allocation scenario involving a learning agent, including applications outside the telecommunication field. We test our approach in a reference channel access scenario, where an intelligent unit distributes communication bandwidth among multiple applications with different urgency levels. Our stimulative results prove that cost of learning is critical for managing network resources, while a cost-free learning model leads to degraded performance.

6

Distributed Reinforcement Learning for Slicing Orchestration

6.1 Introduction

The 5th Generation (5G) paradigm aims at supporting different applications with very specific requirements over the same infrastructure. In this perspective, the 3rd Generation Partnership Project (3GPP) consortium has identified three main service classes, namely enhanced Mobile BroadBand (eMBB), Ultra Reliable Low Latency Communication (URLLC) and massive Machine Type Communication (mMTC) [166]. The eMBB class provides very high throughput in both downlink and uplink, while URLLC appeals to applications with strict latency and reliability constraints. Instead, mMTC is intended to support thousands of monitoring or sensing devices that communicate with minimal energetic and computational costs [167].

Traditional telecommunication networks are often based on a rigid architecture and are not apt to support such services [168]. To overcome this problem, the research community has introduced the concepts of Software Defined Networking (SDN) and Network Function Virtualization (NFV), which can make networks more flexible and adaptable to different requirements [169]. The NFV principle makes it possible to execute network functions over general-purpose devices or virtual machines in the Cloud. Instead, SDN separates the control plane from the forwarding plane, enabling dynamic and differentiated routing of data flows.

In particular, the SDN and NFV concepts are key enablers of the Network Slicing (NS) paradigm, which makes it possible to define multiple virtual networks over the same physical infrastructure [170, 171]. Under this vision, a slice consists of a virtual overlay network designed to support communication services with similar characteristics [172]. Hence, a slice supporting

The work presented in this chapter was published in *IEEE Transactions on Networking* [165].

eMBB applications (e.g., video streaming) should provide a very high bit rate, while a slice supporting URLLC applications (e.g., telesurgery) should guarantee extremely high reliability and low latency.

If defined over the same infrastructure, different slices contend for the same resources, which can be both physical (e.g., optical links) and virtual (e.g., virtual baseband processing units) [173]. In general, the *slice broker* (i.e., the body in charge of initializing and orchestrating slices) acquire such resources from the *infrastructure providers* (i.e., the owners of the physical elements of the network) [174]. Then, the broker assigns slices to the *slice tenants* (e.g., virtual network operators), which offer slice services to the end-users. The amount of slice resources are specified in the so-called Service Level Agreement (SLA) between the slice tenant and broker [175]. Therefore, a fundamental challenge in NS systems is how to distribute resources among the different slices in an efficient way, ensuring that all the SLAs are satisfied [176].

In this chapter, we consider a scenario where two slice classes (i.e., eMBB and URLLC), with dynamic and complementary requirements in terms of throughput, computational power, memory capacity, and delay, are instantiated over the same network infrastructure. In our vision, each slice is composed of multiple information flows (with static routes) that contend for the bandwidth of the network links and the computational and memory resources of the network nodes. Hence, the target problem is to dynamically distribute network resources among flows, considering the characteristic of the slices they belong. The *naive* approach is to allocate communication and computational resources across the slices in a static fashion. However, this method cannot exploit the statistical multiplexing of the information flows and, consequently, may lead to over-provisioning costs and low utilization of the available resources. On the other hand, conventional allocation schemes are generally unsuitable for scenarios in which the type, quantity, and location of resources can be very disparate, and the utility functions are non-convex. These characteristics make the resource allocation in NS a very challenging problem, generally incompatible with a centralized approach.

To address the target scenario, we propose a Machine Learning (ML) approach and attack the problem by exploiting the Deep Reinforcement Learning (DRL) paradigm, which combines Reinforcement Learning (RL) algorithms and Neural Networks (NNs) to find strategies for the management of complex environments [118]. In particular, we first introduce a general network model that represents multiple communication slices running over the same infrastructure in different configurations. Hence, we design a learning system where multiple agents collaborate to allocate network resources among the different slices running over the same infrastructure. The continuous interaction between such learning units makes it possible to increase the system's efficiency in different scenarios independently of the network topology. Besides, we show how Transfer Learning (TL) can improve the system performance by specializing the agent strategy in a scalable manner. The benefits of our approach are assessed in multiple scenarios and compared with a meta-heuristic technique and an efficient empirical algorithm to analyze the advantages and limits of our solution. To our knowledge, our proposal is one of the first attempts to orchestrate network core resources through a fully distributed strategy, where each agent takes decisions with a partial vision of the overall network status. Besides, the proposed

approach exploits TL to adapt policies learned under specific conditions to new network scenarios without the need to re-train everything from scratch. Hence, if the network topology or the slice characteristics change, the single agents can adapt to the new scenario through a limited number of training episodes.

The remainder of the work is organized as follows: Sec. 6.2 discusses the most relevant works in the considered field; Sec. 6.3 describes the system model used for our analysis; Sec. 6.4 describes our learning architecture; Sec. 6.5 presents the resource allocation strategies used as a benchmark; Sec. 6.6 describes the simulation settings and the obtained results; Finally, Sec. 6.7 concludes the chapter with a recap of the lessons learned and some ideas for future work.

6.2 Related Work

In the last years, the emergence of the NS paradigm has made it necessary to investigate new strategies to orchestrate telecommunication systems. The authors of [177] analyze a 5G scenario with end-to-end slices contending for virtual resources offered by data centers and propose a fully distributed algorithm to maximize the slice utility functions, which are assumed to be strictly concave. In [178], Leconte *et al.* design a NS model where multiple traffic flows share network bandwidth and cloud processing units; hence, they implement the Alternating Direction Method of Multipliers [179] to determine the best resource allocation scheme. In [180], a similar approach is applied to a system where multiple network operators share both licensed and unlicensed spectrum. Besides, the authors of [181] focus on the problem of offloading user tasks to edge computing facilities, designing a novel algorithm to optimize server utilization, even when the task execution time is unknown. Finally, Fossati *et al.* propose a framework to adapt multi-resource allocation according to different fairness goals [182]. In doing so, the authors consider also the critical scenario where resources are insufficient to satisfy all the slice demands.

Because of the high complexity of NS scenarios, many studies have also proposed ML-based solutions. For instance, the authors of [183] exploit NNs to predict the traffic evolution in a mobile core network, thus optimizing the routing and the wavelength assignment according to the SDN principles. Another example can be found in [184], where generative adversarial NNs are used to improve the estimation of SLA satisfaction. Instead, [185] proposes a system based on convolutional NNs to associate users with network slices according to the required Quality of Service (QoS). Finally, in [186], Saputra *et al.* implement a distributed architecture predicting the amount of data that has to be cached in the network edge to address the user demands.

Among all the ML techniques used for slice orchestration, DRL is appreciated for its ability to learn complex strategies by trial and error without the need for labeled data. In this regard, the authors of [187] propose a centralized DRL scheme to split the wireless bandwidth among different slices in a single-cell system, targeting the maximization of spectral efficiency and SLA compliance. A similar framework is presented in [188], where a DRL agent optimizes the throughput of users associated with different slices, each with its own QoS requirements. The proposed algorithm enables achieving optimal resource allocation, provided that the number of

resource blocks does not vary over time. We find another example in [189], where the authors consider a more complex system with two optimization layers, each characterized by a different set of actions. At the higher layer, a DRL agent tunes the users' data rates, considering the slice traffic requirements over a long time horizon; instead, at the lower layer, a second agent allocates the cell bandwidth among the users over a short time interval.

A further step towards the development of a comprehensive DRL framework for NS is given in [190], where the authors consider an environment with multiple slices contending for the computational and bandwidth resources provided by a set of nodes. In this case, a centralized DRL agent has a global view of the overall network status and, at each slot, varies the number of resources reserved by a node to a slice. The agent receives a reward proportional to the efficiency of the resource allocation and the SLA compliance. Finally, the work presented in [191] focuses on the cooperation between multiple base stations that belong to the same network. The authors consider a multi-agent scenario where each base station is associated with a DRL agent that determines its bandwidth allocation. In particular, the agents can estimate the overall network status by exchanging embedding vectors, which are computed via a Graph NN system.

Despite the growing interest in DRL for slice orchestration, many questions remain open to investigation. The current literature exploits DRL to optimize only a specific portion of the network (e.g., the wireless access), neglecting the interactions among different learning units. Most of the above approaches assume that the DRL agent has a full vision of the target system. Instead, in more complex (and realistic) scenarios, the network status can be observed partially or with some delay. Finally, to the best of our knowledge, there are no scientific works investigating the adaptability of DRL solutions to different network topologies, which is a key aspect of slice orchestration.

The high heterogeneity of future telecommunication systems requires more flexible strategies, enabling the coexistence of multiple services and adapting to new resource demands in a dynamic fashion. A promising approach is to exploit hierarchical reinforcement learning, which has not yet been fully investigated in this context. Moreover, the TL paradigm can be used to improve the training of the learning agents [192], thus increasing the system's adaptability to multiple scenarios. Our work develops along these directions, with the final aim of designing a fully scalable DRL system that can be separated into smaller units, capable of acting autonomously and cooperating to orchestrate network resources under multiple working conditions.

6.3 System Model

In this section, we model an NS environment in which multiple information flows contend for the same physical and virtual network resources. We adopt a fluid traffic model, where the traffic through a link is viewed as a data stream with a given flow rate. In particular, we assume that network slices have a hierarchical organization so that more flows at different levels can be compounded into an aggregate slice possibly managed by a new tenant. The resulting framework is thus very flexible and can model the interactions among many of the actors involved in a NS scenario. For the reader's convenience, we report the main parameters of our model in Tab. 6.1.

Tab. 6.1: Model parameters.

Parameter	Description	Parameter	Description
t	Discrete time	τ	Slot duration
$\mathcal{L}$	Link	$\mathcal{N}$	Node set
Φ	Information flow set	Σ	Slice class set
Φ_σ	Flows of class σ	Ω	System utility
Φ_l	Flows crossing link l	Φ_n	Flow crossing node n
Ω^ρ	System utility for resource ρ	Ω_σ	System utility for class σ
$\epsilon^i_\phi, \epsilon^e_\phi$	Endpoints of flow ϕ	$\mathbf{r}_\phi$	Demand vector of flow ϕ
ρ_ϕ	Resource required by flow ϕ	$\hat{\rho}_\phi$	Resource assigned to flow ϕ
$F_\sigma(\cdot)$	Flow performance function of class σ	$f_\sigma(\cdot)$	Resource performance function of class σ
c	Computational power	η	Throughput
m	Memory	δ	Delay
$b_{l,\phi}$	Bit rate assigned by link l to flow ϕ	σ_ϕ	Slice class of flow ϕ
$c_{n,\phi}$	Computation assigned by node n to flow ϕ	$m_{n,\phi}$	Memory assigned by node n to flow ϕ
B_l	Rate capacity of link l	C^c_n	Computational capacity of node n
C^m_n	Memory capacity of node n	$b^i_{l,\phi}$	Input rate of flow ϕ in link l
$b^o_{l,\phi}$	Output rate of flow ϕ in link l	d_n	Routing delay due to node n
$D_{l,\phi}$	Data of flow ϕ queued in link l	$d^q_{l,\phi}$	Queuing delay of flow ϕ in link l
$d^\tau_{l,\phi}$	Transmission delay of flow ϕ in link l	d^p_l	Propagation delay due to link l
$\mathcal{S}$	Agent state space	$\mathcal{A}$	Agent action space
$\pi(\cdot)$	Policy function	$V(\cdot)$	State-value function
$L_{\text{critic}}(\cdot)$	Critic loss function	$L_{\text{actor}}(\cdot)$	Actor loss function
$A(\cdot)$	Advantage function	$H(\cdot)$	Entropy function
κ	Entropy tuning parameter	λ	Discount factor
s_ϕ	State of flow ϕ	Γ^ρ_σ	Controller managing resource ρ of class σ
$s^\rho_{n,\phi}$	State of resource ρ of flow ϕ in node n	$s_{l,\phi}$	State of flow ϕ in link l
b^σ_l	Aggregate rate demand of class σ in link l	ρ^σ_n	Aggregate demand of class σ in node n
$b^*_{l,\phi}$	Rate demanded by flow ϕ in link l	$\rho^*_{n,\phi}$	Demand of flow ϕ in node n
$r_{l,\phi}$	Reward of flow ϕ in link l	$r^\rho_{n,\phi}$	Reward of flow ϕ in node n for resource ρ

6.3.1 Slice Model

In our system, we define a *network slice* as an aggregation of information flows with similar behaviors and requirements. We denote by Σ the set including all the slice classes, and by Φ the one including all the information flows. Given $\sigma \in \Sigma$, we indicate by Φ_σ the set of all information flows of class σ. Each information flow $\phi \in \Phi$ is characterized by a tuple of parameters, namely:

- the *flow endpoints* $\mathcal{E}_\phi = (\epsilon^i_\phi, \epsilon^e_\phi)$, which correspond to the network nodes where the users' data enter/exit the slice and usually correspond to base stations, edge routers of autonomous systems, or servers;
- the *resource demand vector* $\mathbf{r}_\phi = [\eta_\phi, c_\phi, m_\phi, \delta_\phi]$, whose elements are the requirements in terms of throughput (η), computational power (c), memory (m), and maximum delay (δ) of the flow;
- the *performance function* $F_\phi(\cdot)$, which describes the performance of the information flow for a given level of fulfillment of its SLA.

We assume that $\mathcal{E}_\phi$, δ_ϕ and $F_\phi(\cdot)$ do not change for the whole duration of the flow ϕ, while η_ϕ, c_ϕ and m_ϕ may change in time, depending on the dynamic of the data source. Note that the throughput η_ϕ is measured in bits per second [b/s], the memory m_ϕ in bits [b], and the delay δ_ϕ in seconds [s]. Finally, we assume the computational requirements are somehow related to

the amount of data generated by the flow source. We hence define the computational power c_ϕ as the speed at which data are processed in the computing facilities; particularly, we express c_ϕ in bits per second [bps].

In our model, we discretize the time in slots $t \in \mathbb{Z}^+$ of τ seconds, assuming that the information flow parameters change only slot by slot. We write $\mathbf{r}_\phi = [\eta_\phi(t), c_\phi(t), m_\phi(t), \delta_\phi(t)]$ and $\hat{\mathbf{r}}_\phi(t) = [\hat{\eta}_\phi(t), \hat{c}_\phi(t), \hat{m}_\phi(t), \hat{\delta}_\phi(t)]$ to indicate the resource vector demanded by and allocated to ϕ during slot t, respectively. Note that $\mathbf{r}_\phi(t)$ is determined by the slice class σ that ϕ belongs to, while $\hat{\mathbf{r}}_\phi(t)$ is determined by the resource allocation strategy.

As mentioned, we consider two different slice classes: eMBB (e) and URLLC (u). In particular, given Φ_e (i.e., the set of the eMBB flows) and Φ_u (i.e., the set of the URLLC flows), we have $\Phi = \Phi_e \cup \Phi_u$ and $\Phi_e \cap \Phi_u = \emptyset$. All the information flows of class σ shares the same performance function $F_\sigma(\cdot)$, i.e., $\forall \phi \in \Phi_\sigma$, $F_\phi(\cdot) = F_\sigma(\cdot)$. In general, $F_\sigma(\cdot)$ depends on both $\mathbf{r}$ and $\hat{\mathbf{r}}$, and returns a value in $[0, 1]$, where 1 means that the SLA has been completely fulfilled. We assume that $F_\sigma(\cdot)$ is a combination of four identical functions $f_\sigma(x_\rho) \in [0, 1]$, with $\rho \in \{\delta, \eta, c, m\}$, where $x_\rho \in [0, 1]$ indicates the level at which the demand for resource ρ is satisfied. The function $f_\sigma(x_\rho)$, hence, can take different forms, depending on the characteristics of the considered slice, but in general it is non-decreasing and such that $f_\sigma(x_\rho) = 1 \; \forall \; x_\rho \geq 1$. To maximize the performance, the allocated resource should equal or exceed the flow request.

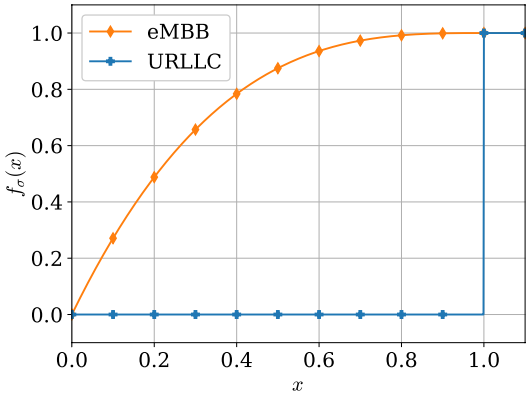


Fig. 6.1: Resource performance function.

Although our framework can accommodate virtually any utility function, we focus on two cases that we deem suitable to represent eMBB and URLLC classes. For what concerns the eMBB slice, we assume

$$F_e(\mathbf{r}, \hat{\mathbf{r}}) = \alpha_\delta f_e\left(\frac{\delta}{\hat{\delta}}\right) + \sum_{\rho \in \{\eta, c, m\}} \alpha_\rho f_e\left(\frac{\hat{\rho}}{\rho}\right) \tag{6.1}$$

where $\alpha_\eta, \alpha_c, \alpha_m$, and α_δ are non negative and add up to 1, while $f_e(x_\rho)$ is defined as

$$f_e(x) = \begin{cases} \beta_1 x + \beta_2 x^2 + \beta_3 x^3, & x \in [0,1); \\ 1, & x \geq 1. \end{cases} \tag{6.2}$$

Particularly, β_1, β_2 and β_3 are scalar parameters ensuring that $f_e(\cdot)$ is concave and monotonic increasing for $x \in [0,1]$. The smooth and concave shape of $f_e(\cdot)$ shown in Fig. 6.1 reflects the flexibility of the eMBB services with respect to the SLA requirements. Hence, we assume that the quality of experience of the eMBB users degrades rather graciously when the SLA is violated, as in the case of video-streaming applications [193].

Conversely, URLLC flows have strict requirements that, if infringed, cause the sudden degradation of the related services. This case includes applications like, for example, teleguided robotic surgery, which does not tolerate any increase in communication delay. The performance function for this class of services is the product of step functions $f_u(\cdot)$ (shown in Fig. 6.1):

$$F_u(\mathbf{r}, \hat{\mathbf{r}}) = f_u\left(\frac{\delta}{\hat{\delta}}\right) \times \prod_{\rho \in \{\eta, c, m\}} f_u\left(\frac{\hat{\rho}}{\rho}\right), \tag{6.3}$$

where

$$f_u(x) = \begin{cases} 0, & x \in [0,1); \\ 1, & x \geq 1. \end{cases} \tag{6.4}$$

In this case, $F_u(\cdot)$ is either maximum when all the SLA requests are satisfied or zero when even one resource requirement is not met.

We remark that our system can be easily extended by defining other slices with different performance functions. In this way, the slice broker can change the composition of the slice set, addressing the requirements of new tenants in different scenarios.

Given the function $F_\sigma(\cdot)$ for each class $\sigma \in \Sigma$, the system utility is obtained a weighted sum of the slices performance:

$$\Omega = \frac{|\Phi_e|}{|\Phi|}\Omega_e + \frac{|\Phi_u|}{|\Phi|}\Omega_u, \tag{6.5}$$

where

$$\Omega_\sigma = \frac{1}{|\Phi_\sigma|} \sum_{\phi \in \Phi_\sigma} F_\sigma(\mathbf{r}_\phi, \hat{\mathbf{r}}_\phi), \ \sigma \in \Sigma, \tag{6.6}$$

and $|\mathcal{X}|$ represents the cardinality of $\mathcal{X}$. We observe that $F_\sigma(\cdot)$ always takes values in $[0,1]$, so that we also have $\Omega \in [0,1]$.

For completeness, we also define the system utility for a specific type of resource ρ as

$$\Omega^\rho = \frac{|\Phi_e|}{|\Phi|}\Omega_e^\rho + \frac{|\Phi_u|}{|\Phi|}\Omega_u^\rho, \tag{6.7}$$

where

$$\Omega_{\sigma}^{\rho} = \begin{cases} \frac{1}{|\Phi_{\sigma}|} \sum_{\phi \in \Phi_{\sigma}} f_{\sigma}\left(\rho/\hat{\rho}\right), & \text{if } \rho = \delta; \\ \frac{1}{|\Phi_{\sigma}|} \sum_{\phi \in \Phi_{\sigma}} f_{\sigma}\left(\hat{\rho}/\rho\right), & \text{otherwhise;} \end{cases} \tag{6.8}$$

and $\sigma \in \Sigma$.

6.3.2 Network Model

Our model is based on two different network elements, namely *node* and *link*, as detailed below.

- We distinguish two types of nodes: *access nodes* are located at the network edge and connect the users with the rest of the network; *core nodes* are located in the core of the network and forward the aggregate data flows exchanged by the access nodes. We assume that each node n is equipped with a certain amount of computational C_n^c and memory C_n^m resources, which may differ between access and core nodes.
- We call *link* any connection l between two different nodes of the network (fronthaul or backhaul). This element is provided with a bit rate B_l to support the communications between the connected nodes.

From now on, we denote by $\mathcal{N}$ and $\mathcal{L}$ the set of network nodes and links, respectively.

In our model, each information flow $\phi \in \Phi$ is initialized between two access nodes (ϵ_{ϕ}^{i} and ϵ_{ϕ}^{e}) and passes through a certain number of core nodes and links. We observe that, in general, slices can be activated and deactivated on-demand, thus varying the number of flows in the network. Nonetheless, once established, the path of a flow ϕ is maintained for the whole duration of the connection unless some links become unavailable or the communication endpoints change. In this case, our model will react as the flow was interrupted, while a new one starts operating along the new path. We assume that such events are rare and do not impact the dynamics of the proposed scheme.

We call Φ_l and Φ_n the set of information flows that cross link $l \in \mathcal{L}$ and node $n \in \mathcal{N}$, respectively. Given a link $l \in \mathcal{L}$, each flow $\phi \in \Phi_l$ gets assigned a portion $b_{l,\phi}$ of the link bit rate B_l. Similarly, each node $n \in \mathcal{N}$ assigns to ϕ an amount $c_{n,\phi}$ and $m_{n,\phi}$ of its computational and storage resources. Consequently, any resource allocation pattern must comply with the following feasibility conditions:

$$\sum_{\phi \in \Phi_l} b_{l,\phi} \leq B_l, \qquad \forall\ l \in \mathcal{L}; \tag{6.9}$$

$$\sum_{\phi \in \Phi_n} \rho_{n,\phi} \leq C_n^{\rho}, \qquad \forall\ n \in \mathcal{N}, \rho \in \{c, m\}. \tag{6.10}$$

Given a certain allocation of network resources, we want to compute $\hat{\mathbf{r}}_{\phi}$, $\forall\ \phi \in \Phi$. We denote by $\mathcal{N}_{\phi}$ and $\mathcal{L}_{\phi}$ the ordered set of network nodes and links crossed by ϕ. In particular, the first and the last element of $\mathcal{N}_{\phi}$ constitute the flow endpoints. We assume that the computational and memory requests of a flow $\phi \in \Phi$ can be addressed by aggregating the resources assigned

to ϕ by each node along its path:

$$\hat{\rho}_\phi = \sum_{n \in \mathcal{N}_\phi} \rho_{n,\phi}, \quad \forall\ \rho \in \{c, m\}. \tag{6.11}$$

To determine the throughput $\hat{\eta}_\phi$, instead, we need to consider the output flow rate $b^o_{l,\phi}(t)$ at time t from each link $l \in \mathcal{L}$. Indeed, the throughput corresponds to the output rate from the last link λ along the path:

$$\hat{\eta}_\phi(t) = b^o_{\lambda,\phi}(t). \tag{6.12}$$

In turn, $b^o_{l,\phi}(t)$ depends on both the bit rate $b_{l,\phi}(t)$ assigned to ϕ by l, the input flow rate $b^i_{l,\phi}(t)$ from the upstream link, and the amount of data of ϕ queued at node n at the end of the previous slot, which is denoted by $D_{l,\phi}(t-1)$. The input flow rate $b^i_{l,\phi}(t)$ is given by

$$b^i_{l,\phi}(t) = \begin{cases} b^o_{\ell,\phi}(t), & \text{if } \ell \text{ is the upstream link of } l \text{ in } \mathcal{L}_\phi; \\ \eta_\phi(t), & \text{if } l \text{ is the first link in } \mathcal{L}_\phi. \end{cases} \tag{6.13}$$

The output flow rate $b^o_{l,\phi}(t)$ is then given by the minimum between the allocated rate $b_{l,\phi}(t)$ and the sum of the incoming and queued traffic, i.e.,

$$b^o_{l,\phi}(t) = \min\left\{ b_{l,\phi}(t), \frac{D_{l,\phi}(t-1)}{\tau} + b^i_{l,\phi}(t) \right\}. \tag{6.14}$$

The variable $D_{l,\phi}(t)$ is set to 0 for any time t before the initialization of the flow, and then it is updated as

$$D_{l,\phi}(t) = \max\left\{0, D_{l,\phi}(t-1) + \tau(b^i_{l,\phi}(t) - b_{l,\phi}(t))\right\}. \tag{6.15}$$

The above equation reflects that the queue length varies proportionally to the difference between incoming flow rate $b^i_{l,\phi}$ and assigned link rate $b_{l,\phi}$. In particular, $D_{l,\phi}(t)$ grows when this difference is positive and decreases when it is negative (until the queue is empty).

For what concerns the delay experienced by ϕ, we have

$$\hat{\delta}_\phi(t) = \sum_{n \in \mathcal{N}_\phi} d_n + \sum_{l \in \mathcal{L}_\phi} d_{l,\phi}(t), \tag{6.16}$$

where d_n is a positive value representing the delay due to routing operations at node n, assumed constant over time. Instead, $d_{l,\phi}(t)$ is computed as

$$d_{l,\phi}(t) = \tau^q_{l,\phi}(t) + \tau^\tau_{l,\phi}(t) + \tau^p_l, \tag{6.17}$$

where $\tau^q_{l,\phi}$, $\tau^\tau_{l,\phi}$, and τ^p_l represent the queuing, transmission, and propagation delays of ϕ through link l, respectively. In particular, $\tau^q_{l,\phi}(t)$ is given by (see Sec.6.3.3)

$$\tau^q_{l,\phi}(t) = \frac{2D_{l,\phi}(t-1) - \tau(b^o_{l,\phi}(t) - b^i_{l,\phi}(t))}{2b_{l,\phi}(t)}, \tag{6.18}$$

while, $\tau^{\tau}_{l,\phi}(t)$ is

$$\tau^{t}_{l,\phi}(t) = \frac{1}{b^{o}_{l,\phi}(t)}. \tag{6.19}$$

Finally, τ^{p}_{l} is a positive and constant value that depends on the physical characteristic of the communication link. We highlight that, despite considering a discrete time frame, $\hat{\delta}_{\phi}$ can take any real (non-negative) value.

We aim to determine the resource allocation that maximizes the system utility given in (6.5). Mathematically, we want to determine $\hat{\mathbf{r}}_{\phi}$, $b_{l,\phi}$, $c_{n,\phi}$, $m_{n,\phi}$, $b^{i}_{l,\phi}$, $b^{o}_{l,\phi}$, $\tau^{q}_{l,\phi}$, and $\tau^{\tau}_{l,\phi}$, $\forall\ \phi \in \Phi$, $n \in \mathcal{N}$, $l \in \mathcal{L}$ that maximize Ω, under the constraints given in (6.9)-(6.19).

The many constraints and the non-convexity of Ω make the problem very complex to solve. Only a centralized controller would have a complete and up-to-date view of all system variables at each slot, making it possible to determine the optimal solution. Assuming such complete information is available, we can use conventional optimization tools or meta-heuristic techniques to identify the best resource allocation scheme. However, the first may converge on local maxima, while the latter may be unable to find a solution within a short time. These issues can be alleviated by distributed control algorithms that promptly take new actions, albeit with a partial view of the overall network. The DRL paradigm is suitable for this problem since it can provide high-performance solutions to carry out our complex control tasks when the environment is partially observable.

6.3.3 Queuing Time Approximation

In what follows, we derive equation (6.18), which determines the average queuing time $d^{q}_{l,\phi}(t)$ of ϕ in link l during slot t. Let τ^{*} be the minimum between the slot boundary τ and the time at which the queue empties, i.e.,

$$\tau^{*} = \min\left(\tau, \frac{D_{l,\phi}(t-1)}{b_{l,\phi}(t) - b^{i}_{l,\phi}(t)}\right). \tag{6.20}$$

Furthermore, let $\mathcal{D}_{l,\phi}(t,u)$ denote the residual backlog of flow ϕ in link l, u seconds after the beginning of slot t. Therefore, for any $u \in [0, \tau^{*}]$ we have $\mathcal{D}_{l,\phi}(t,u) = D_{l,\phi}(t) - u(b_{l,\phi}(t) - b^{i}_{l,\phi}(t))$, while $\mathcal{D}_{l,\phi}(t,u) = 0$ for $u \in (\tau^{*}, \tau]$. Now, the queuing delay experienced by the incoming flow at time $u \in [0, \tau]$ is zero if the queue is empty, and otherwise equal to $\delta(u,t) = \frac{\mathcal{D}_{l,\phi}(u,t)}{b_{l,\phi}(t)}$. The average delay over slot t is hence

$$d^{q}_{l,\phi}(t) = \frac{1}{\tau}\int_{0}^{T} \delta(u,t)\mathrm{d}u = \frac{1}{\tau}\int_{0}^{\tau^{*}} \frac{\mathcal{D}_{l,\phi}(u,t)}{b_{l,\phi}(t)}\mathrm{d}u \tag{6.21}$$

$$= \frac{2\tau^{*} D_{l,\phi}(t) - {\tau^{*}}^{2}(b_{l,\phi}(t) - b^{i}_{l,\phi}(t))}{2\tau b_{l,\phi}(t)}. \tag{6.22}$$

For $\tau^* = \tau$, we obtain

$$d^q_{l,\phi}(t) = \frac{2D_{l,\phi}(t) - \tau(b_{l,\phi}(t) - b^i_{l,\phi}(t))}{2b_{l,\phi}(t)}. \tag{6.23}$$

For $\tau^* < \tau$, instead, we have $\tau^*(b_{l,\phi}(t) - b^i_{l,\phi}(t)) = D_{l,\phi}(t)$, so that (6.22) yields

$$d^q_{l,\phi}(t) = \frac{\tau^* D_{l,\phi}(t)}{2\tau b_{l,\phi}(t)} < \frac{D_{l,\phi}(t)}{2b_{l,\phi}(t)}. \tag{6.24}$$

Recalling that $b^o_{l,\phi}(t)$ used in (6.18) is defined as the minimum between $b_{l,\phi}(t)$, and $\frac{2D_{l,\phi}(t-1)}{\tau} + b^i_{l,\phi}(t)$, we can see that (6.18) is indeed a compact expression for $d^q_{l,\phi}(t)$, provided that it is approximated by its upper bound when $\tau^* < \tau$.

6.4 Learning Strategy

In order to efficiently orchestrate communication resources in a NS scenario, we develop a distributed architecture based on multiple learning units, named *local controllers*, that collaborate to maximize the overall system utility given by (6.5). Our approach is based on RL, a powerful mathematical framework to address decision-making problems in complex environments.

6.4.1 Learning Architecture

In this work, we adopt an actor-critic approach [194], which learns the policy π by two different units. The first is named *actor* and approximates the policy function $\pi_\theta(\cdot)$, parameterized by θ; the latter is named *critic* and approximates the state-value function $V_\gamma(\cdot)$, parameterized by γ. We give the complete definitions of policy and state-value functions, as the other aspects of the RL theory, in Appendix B. Hence, during any slot t, the actor returns the action $a(t) \in \mathcal{A}$ that the policy π_θ takes in the current state $s(t) \in \mathcal{S}$, where $\mathcal{A}$ and $\mathcal{S}$ are the agent action and state space, respectively. Instead, the critic computes the expected discounted reward obtained following the policy π_θ from state $s(t) \in \mathcal{S}$.

To carry out the system training, we exploit the Advance Actor Critic (A2C) algorithm, which has shown to provide stable DRL solutions in very complex scenarios [195]. The critic is trained to minimize the function $L_{\text{critic}}(\cdot)$, defines as

$$L_{\text{critic}}(s(t), a(t)) = (A(s(t), a(t)))^2 = (r(t) + \lambda V_\gamma(s(t+1)) - V_\gamma(s(t)))^2, \tag{6.25}$$

where $r(t)$ is the reward gained by the agent at slot t. Particularly, the function $A : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is called *advantage* and represents the benefit obtained by choosing action $a(t)$ rather than the one recommended by the current policy π. Conversely, the actor aims to minimize the function

$$L_{\text{actor}}(s(t), a(t)) = -\nabla_\theta \log \pi_\theta(s(t), a(t)) A(s(t), a(t))^2 - \kappa H(\pi_\theta), \tag{6.26}$$

where ∇_θ is the gradient with respect to θ, $H(\pi_\theta)$ is the entropy of π_θ, and κ is a scalar value.

As suggested in [195], the actor loss function depends linearly on the policy entropy $H(\pi_\theta)$. We can promote the exploration of the action space by increasing κ since, in such a case, the actor obtains benefits from taking random actions. Instead, as $\kappa \to 0$, the actor will choose actions that are expected to bring the highest reward according to the current experience.

We highlight that A2C enables us to consider continuous action spaces: this is not possible with traditional RL algorithms (e.g., Q-Learning) that, instead, can only take actions from discrete sets. Besides, the A2C algorithm supports on-policy training and, consequently, allows the agents to continuously refine the target policy while using its outputs in a real scenario. Hence, the slice broker can progressively and dynamically adapt the policy to new scenarios without needing to train a new system from scratch every time the working conditions (e.g., target network topology or slice requirements) change.

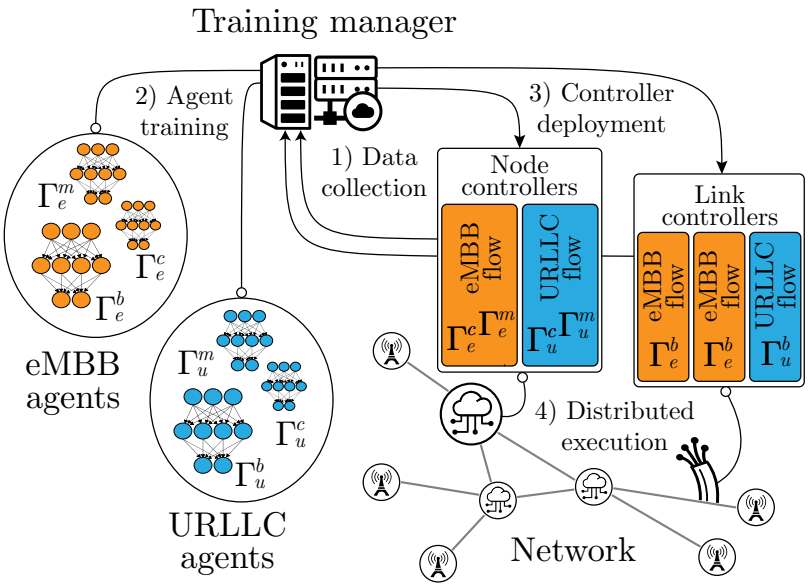


Fig. 6.2: Learning Architecture.

Our learning architecture (shown in Fig. 6.2) provides a different controller for each information flow and network element. From an operational point of view, the total number of local controllers depends on the network topology and the cardinality of Φ. Practically, the local controllers are all replicas of $3 \times |\Sigma|$ learning agents. During the training phase, we design a tuples of agents $(\Gamma^b_\sigma, \Gamma^c_\sigma, \Gamma^m_\sigma)$ for each different class $\sigma \in \Sigma$: the agent Γ^b_σ is trained to orchestrate the bit rate of each information flow $\phi \in \Phi_\sigma$ in each link $l \in \mathcal{L}$; instead, Γ^c_σ and Γ^m_σ are trained to orchestrate the computation and memory resources of each information flow $\phi \in \Phi_\sigma$ in each node $n \in \mathcal{N}$.

As shown in Fig. 6.2, the training phase is performed by a central entity, named *training manager*, which collects the system information and updates the learning agents accordingly. Then, the system deploys copies of Γ^b_σ, Γ^c_σ, and Γ^m_σ in each network element crossed by any flow $\phi \in \Phi_\sigma$. Alternatively, each network element may develop its own version of the learning

agents, using only local information and avoiding interactions with the central manager; the drawback is that the training phase lacks generality. In Sec. 6.6.3, we will show how to use the TL paradigm for combining the two approaches.

Tab. 6.2: Agent architectures.

	Γ_σ^b		Γ_σ^c		Γ_σ^m	
Parameter	Actor	Critic	Actor	Critic	Actor	Critic
Input size	11	11	7	7	7	7
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Hidden size	12	12	8	8	8	8
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Hidden size	6	6	4	4	4	4
Activation	Linear	Linear	Linear	Linear	Linear	Linear
Output size	1	1	1	1	1	1

According to the A2C algorithm, each agent is composed of two units, i.e., the actor and the critic, which are implemented by means of NNs. Particularly, we consider an architecture with two hidden layers and the Rectifier Linear Unit (ReLU) as activation function [196]. The output of the actor is the amount ρ^* of resources that the local controller demands, while that of the critic is the expected future reward. The size of the NN input varies according to the target resources as we explain in the next subsection. We resume the settings of the learning architectures in Tab. 6.2.

6.4.2 Observations and Actions

In our system, each local controller has full knowledge of the element where it operates, while it has a limited view of the network, which implies that the system state is only partially observable [197]. Let us consider a local controller managing the rate resources of a flow ϕ in a link l. At the beginning of each slot, the controller is provided with two vectors representing the status of the information flow and the network element it is associated with.

The first vector gives the state of ϕ at the beginning of slot t:

$$\mathbf{s}_\phi(t) = [\mathbf{r}_\phi(t), \hat{\mathbf{r}}_\phi(t-1)], \tag{6.27}$$

where $\mathbf{r}_\phi(t)$ and $\hat{\mathbf{r}}_\phi(t-1)$, defined in Sec. 6.3, are the resources requested by and granted to ϕ during slot t and $t-1$, respectively. We observe that $\mathbf{r}_\phi(t)$ and $\hat{\mathbf{r}}_\phi(t-1)$ can be computed only knowing the *aggregate* amount of network resources assigned to ϕ by the network elements across its routing path. Therefore, $\mathbf{s}_\phi(t)$ is shared among all the controllers assigned to ϕ at the beginning of each slot t. However, the size of $\mathbf{s}_\phi(t)$ is negligible with respect to the rate requirements of the slices and, therefore, can be transmitted within the user data plane of ϕ without degrading the performance of our system.

The second vector provides the state of the rate resources of ϕ in l at the beginning of slot t:

$$\begin{aligned}\mathbf{s}_{l,\phi}(t) = [&B_l, d_{l,\phi}(t-1), D_{l,\phi}(t-1), b^*_{l,\phi}(t-1),\\ &b_{l,\phi}(t-1), b^e_l(t-1), b^u_l(t-1)],\end{aligned} \tag{6.28}$$

where $b_l^\sigma(t-1)$ is the aggregate rate demanded at the link l by all flows of class σ during slot $t-1$, while the other parameters were defined in Sec. 6.3. We highlight that this information is locally available to the link controller and, consequently, $\mathbf{s}_{l,\phi}(t)$ does not need to be transmitted across the network. Hence, at the beginning of slot t, the controller takes $\mathbf{s}_\phi(t)$ and $\mathbf{s}_{l,\phi}(t)$ as input and returns $b^*_{l,\phi}(t)$, which is the bitrate requested by ϕ at link l during slot t.

When considering a controller associated to a node n and a flow ϕ we use the same approach and, depending on the resource $\rho \in \{c, m\}$ we want to allocate, we substitute (6.28) with $s^c_{n,\phi}(t)$ or $s^m_{n,\phi}(t)$, which are the states of the computation and memory resources assigned to ϕ by node n. In particular, $s^c_{n,\phi}(t)$ or $s^m_{n,\phi}(t)$ are defined as

$$\begin{aligned}\mathbf{s}^\rho_{n,\phi}(t) = [C^\rho_n, \rho^*_{n,\phi}(t-1), \rho_{n,\phi}(t-1),\\ \rho^e_n(t-1), \rho^u_n(t-1)],\end{aligned} \tag{6.29}$$

where $\rho^\sigma_n(t-1)$ is the aggregate amount of resource ρ demanded to node n by all the flows of class σ during slot $t-1$. As before, the node controller takes $\mathbf{s}_\phi(t)$ and $\mathbf{s}^\rho_{n,\phi}(t)$ as input and returns $\rho^*_{n,\phi}(t)$, with $\rho \in \{c, m\}$, which is the amount of computational (or memory) capacity demanded by ϕ to node n during slot t.

6.4.3 Reward Function

In accordance with the RL paradigm, we need to define a reward function $r(\cdot)$ that represents the benefit generated by each possible state-action pair of the policy. In particular, to maximize the overall utility, each local controller should demand enough resources to address the requirements of the target flow ϕ without subtracting too many resources from the other flows. In our system, given a controller associated with a link $l \in \mathcal{L}$ and a flow $\phi \in \Phi_l$, the reward at time t is given by

$$\begin{aligned}r_{l,\phi}(t) =& \gamma_0 \left(f_{\sigma_\phi}\left(\frac{\eta_\phi(t)}{\hat{\eta}_\phi(t)}\right) + f_{\sigma_\phi}\left(\frac{\hat{\delta}_\phi(t)}{\delta_\phi(t)}\right)\right) + \\ & \frac{\gamma_1}{|\Phi_l|}\sum_{\psi\in\Phi_l}\left(f_{\sigma_\psi}\left(\frac{\eta_\psi(t)}{\hat{\eta}_\psi(t)}\right) + f_{\sigma_\psi}\left(\frac{\hat{\delta}_\psi(t)}{\delta_\psi(t)}\right)\right),\end{aligned} \tag{6.30}$$

where σ_ϕ is the slice class that ϕ belongs to, while γ_0 and γ_1 are positive scalar values. In particular, γ_0 weights the throughput and delay performance of flow ϕ, while γ_1 measures the average throughput and delay performance of all the other flows crossing link l.

Similarly, a controller assigned to a node n and a flow ϕ is rewarded according to

$$r^\rho_{n,\phi}(t) = \gamma_0 f_{\sigma_\phi}\left(\frac{\rho_\phi(t)}{\hat{\rho}_\phi(t)}\right) + \frac{\gamma_1}{|\Phi_n|}\sum_{\psi\in\Phi_n} f_{\sigma_\psi}\left(\frac{\rho_\psi(t)}{\hat{\rho}_\psi(t)}\right), \tag{6.31}$$

where $\rho \in \{c, m\}$, while the scalar values γ_0 and γ_1 have the same role as before.

Therefore, the reward function consists of the weighted sum of two terms: the first reflects the performance of the flow targeted by the controller, while the second represents the aggregate

performance of all the flows crossing the considered network element. With such a mixed reward function, the controller will attempt to improve the quality of the targeted flow without unduly penalizing other flows.

6.5 Benchmark Strategies

In this section, we discuss some DRL approaches that can be considered to benchmark the proposed solution, starting from some of the works described in Sec. 6.2. Then, we define two baseline strategies: the first is an algorithm that tries to distribute communication and processing resources in each network location fairly; the latter is based on meta-heuristic optimization and performs a static allocation of network resources.

We highlight that the chosen benchmarks use information that is not supposed to be available to the DRL approach, thus gaining a competitive advantage for problem resolution. This choice permits a simple interpretation of the results, highlighting the pros and cons of our DRL system without being trivial.

6.5.1 Other Learning-Based Approaches

Out of the several ML algorithms described in Sec. 6.2, the approaches proposed in [187, 188, 189, 190, 191] are those more aligned with the problem addressed in this paper. Although fundamental differences in the system model prevent a direct comparison between such approaches and our solution, some high-level considerations are still possible.

We first observe that the authors of [187, 188, 189] focus on the basic scenario of coordinating the users associated with a single cell, focusing on one specific type of resource (namely, channel bandwidth). These solutions are not well suited to handle the allocation of multiple resources, which affect different performance indices in an intertwined manner. In our system, such methods will fail, not being designed to deal with the reward variability determined by the interaction among multiple agents deployed in different network locations.

The authors of [190] design a multi-node topology but assume that a single centralized agent can operate in the network, considering one node at a time. This approach does not suit the high dynamism of our scenario, which requires reallocating bandwidth and computational power in multiple network nodes and links simultaneously. In particular, if the user demands change too fast in time, the resources assigned to the different slices may be insufficient or unnecessary. On the other hand, our system can well suit the scenario presented in [190], allowing a faster allocation at the cost of a sub-optimal performance since it is not possible for the learning agents to observe the entire network status.

The framework proposed in [191] is the only one involving a multi-agent system to orchestrate the slice resources, in line with our proposal. However, in [191], the different agents can share their views on the network status in real-time, violating one of the hypotheses of our model. To make such a strategy consistent with our system model, we should consider the delay in the computation of the embedding vectors, which makes it impossible for the agents to estimate

the network status in real time. At the same time, implementing our approach in the model of [191], we expect to obtain a degraded performance since each agent or our system exploits only one part of the network information available.

We finally highlight that none of the above algorithms can adapt to dynamic network topologies, which is one of the most important aspects of our multi-agent framework. Implementing any of these approaches in our system model requires the complete redesign of the solutions, making fair analysis impossible. For this reason, in the rest of the paper, we will compare our DRL architecture with two simple algorithms, based on easy-to-interpret criteria, in what follows.

6.5.2 Empirical Strategy

Similarly to our approach, the empirical strategy implements a distributed resource allocation scheme. At the beginning of slot t, each flow ϕ crossing link l demands a bit rate sufficient to both satisfy the current throughput requirement and transmit any buffered data, i.e.,

$$b^*_{l,\phi}(t) = \eta_\phi(t) + \frac{D_{l,\phi}(t-1)}{\tau}. \tag{6.32}$$

For what concerns computation power and storage, each flow ϕ distributes the resource demands among all the nodes in its path proportionally to their capacity. More specifically, the amount of resources of type $\rho \in \{c, m\}$ required to node n is equal to

$$\rho^*_{n,\phi}(t) = \chi_{n,\phi}\rho_\phi(t), \tag{6.33}$$

with

$$\chi_{n,\phi} = \frac{C^\rho_n}{\sum_{k\in\mathcal{N}_\phi} C^\rho_k}. \tag{6.34}$$

We highlight that, to compute $\chi_{n,\phi}$, it is necessary to know the computation and storage capacities of each node $n \in \mathcal{N}_\phi$, information that is not assumed to be available to the controllers in our DRL strategy. Hence, the empirical strategy has an advantage with respect to our learning framework since it uses information that is not available in a fully distributed approach.

We observe that, either using the empirical or the DRL strategy, network elements may not be able to satisfy all the requests they receive. Hence, the aggregate amount of resources demanded by a link l (or a node n) may exceed its overall capacity. Hence, we need to map the demanded resources $b^*_{l,\phi}$, $c^*_{n,\phi}$, $m^*_{n,\phi}$ to the assigned resources $b_{l,\phi}$, $c_{n,\phi}$, $m_{n,\phi}$, $\forall\ l \in \mathcal{L}$, $n \in \mathcal{N}$, ensuring that the feasibility constraints (6.9) and (6.10) are always satisfied.

Let us consider a link $l \in \mathcal{L}$ during a slot t. If the total demand is lower than B_l, the feasibility constraints are already met: consequently, we can set $b_{l,\phi}(t) = b^*_{l,\phi}(t)$. Conversely, if the aggregate request exceeds the link capacity B_l, then each flow gets at most a fraction of the

link rate proportional to its request:

$$b_{l,\phi}(t) = b^*_{l,\phi}(t) \min \left\{ 1, \frac{B_l}{\sum_{\psi \in \Phi_l} b^*_{l,\psi}(t)} \right\}. \tag{6.35}$$

Using the same principle for the computational and memory resources, we can write

$$c_{n,\phi}(t) = c^*_{n,\phi}(t) \min \left\{ 1, \frac{C^c_n}{\sum_{\psi \in \Phi_n} c^*_{n,\psi}(t)} \right\}, \tag{6.36}$$

and

$$m_{n,\phi}(t) = m^*_{n,\phi}(t) \min \left\{ 1, \frac{C^m_n}{\sum_{\psi \in \Phi_n} m^*_{n,\psi}(t)} \right\}. \tag{6.37}$$

6.5.3 Static Strategy

Meta-heuristic techniques have been shown to determine the optimal solution to highly complex optimization problems with non-convex constraints [198]. In our system, a Genetic Algorithm (GA) may be used to approximate the best resource allocation pattern, thus outperforming both the DRL and empirical strategies. However, meta-heuristic algorithms are based on a randomized search of the target solution and require an extremely long calculation time, which makes it unfeasible to execute them at each time slot. At the same time, it is reasonable to exploit meta-heuristic techniques if the resource requirements do not vary in time.

In the static strategy we consider in this work, we divide the network resources among the different slices without varying them independently on the network conditions. Practically, each traffic flow ϕ is assumed to have fixed requirements, corresponding to the average amount of resources it demands, i.e., $\mathbf{r}^a_\phi$. Hence, a GA is used to determine the optimal resource allocation pattern under such conditions [199].

We observe that, using the static strategy, the values of $b_{l,\phi}, c_{n,\phi}, m_{n,\phi}\ \forall\ \phi \in \Phi$, $l \in \mathcal{L}$, $n \in \mathcal{N}$ are maintained fixed. In other words, the variability of information flows is not taken into account, and the performance of each flow may deteriorate as soon as its requirements exceed the average values. This issue is critical, especially for the URLLC slice, whose performance function suddenly drops if any resource requirements are not satisfied.

To better highlight the characteristics of the different strategies, in Fig. 6.3, we represent the share of resources assigned to the eMBB and URLLC services in a network link during a period of 50 slots. In particular, we consider a scenario where the link capacity is lower than the aggregate resource demand, which means that all the strategies fully use the available bitrate. As expected, GA statically distributes network resources: the bitrate assigned to the different slices does not vary in time. Instead, the empirical algorithm dynamically adapts to the slice requirements, and the bitrate distribution changes at each slot. Also, the DRL strategy follows a dynamic trend but assigns a higher amount of resources to the URLLC services than the benchmarks.

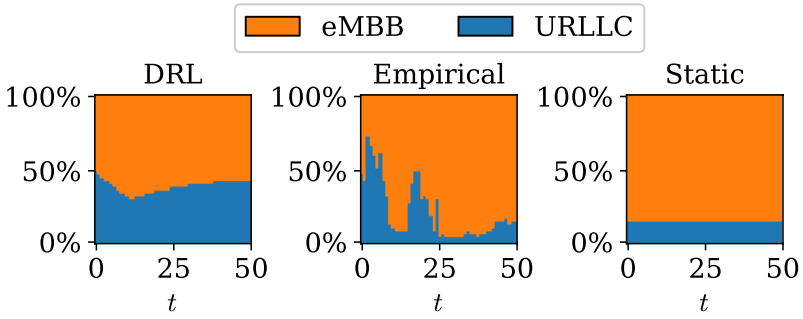


Fig. 6.3: Link resource allocation.

6.6 Simulation Setting and Results

Tab. 6.3: Simulation settings.

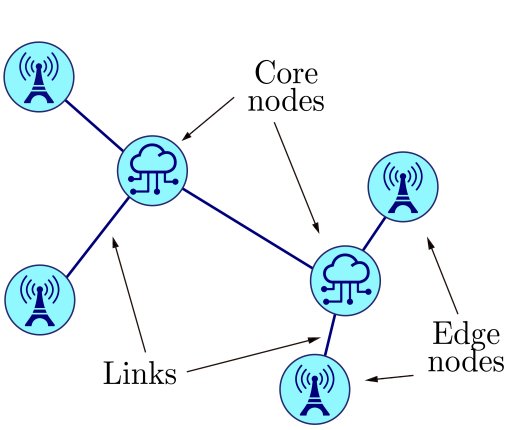
Parameter	Value	Description	Parameter	Value	Description
τ	0.1 s	Slot duration	T	50	Slots per episode
B_l	50 Gbps	Link rate capacity	C_n^c	$\{10, 20, 30, 60\}$ Gbps	Node computational capacity
C_n^m	$\{10, 20, 30, 60\}$ Gb	Node memory capacity	S_σ	10	Resource requirements of class σ
d_n	0.001 ms	Node routing delay	d_l^p	0.1 ms	Link propagation delay
N_{train}	$\{3, 5\} \cdot 10^4$	Training episodes	N_{test}	500	Testing episodes
N_{transfer}	$2 \cdot 10^4$	TL episodes	N_Φ	$\{2, ..., 6\}$	Number of flows
ζ_a	10^{-5}	Actor learning rate	ζ_c	10^{-5}	Critic learning rate
λ	0.9	Discount factor	κ	10^{-4}	Entropy tuning parameter
$\alpha_\eta, \alpha_c, \alpha_m, \alpha_\delta$	0.25, 0.25, 0.25, 0.25	eMBB performance weights	γ_0, γ_1	0.1, 1	Reward weights

In this section, we first describe the scenarios where our algorithms are tested, and the setting of our simulations, whose parameters are summarized in Tab. 6.3. Then, we investigate the performance of our DRL architecture against the benchmark strategies and under different conditions. Finally, we show how the TL paradigm can further improve the system's performance.

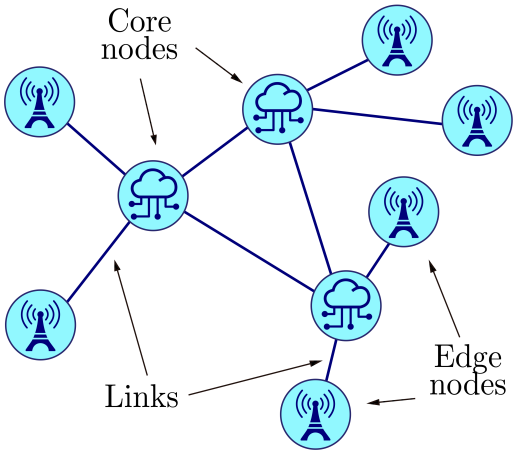
6.6.1 System Setting

We consider three different network scenarios, named *Dumbbell* (D), *Triangle* (T), and *Pyramid Network* (P), whose topologies are reported in Fig. 6.4. In all the cases, the number of information flows in the network is $N_\Phi \in \{2, ..., 6\}$. The capacities of each network element are fixed; specifically, we set $B_l = 50$ Gbps, $C_n^c = 60$ Gbps and $C_n^m = 60$ Gb for the core nodes, $C_n^c = 20$ Gbps and $C_n^m = 20$ Gb for the access nodes. Hence, our model assumes that most computing and storage resources are in the core network. Concerning the delay, we assume that $d_l^p = 0.1$ ms, $\forall\ l \in \mathcal{L}$, and $d_n = 0.001$ ms, $\forall\ n \in \mathcal{N}$. Although arbitrary, these values are well-aligned with the features of modern network elements.

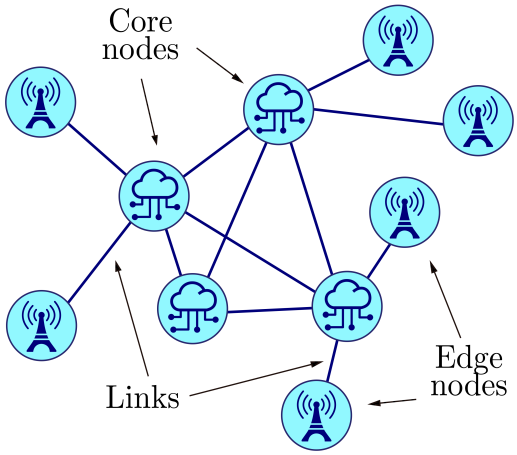
To model the information flow requirements, we consider a Markov model [208] M_σ with transition probability matrix $\mathbf{P}_\sigma$ for each class $\sigma \in \Sigma$. The model include $S_\sigma = 10$ states and is designed in such a way that transitions can only occur between adjacent states; mathematically, $\mathbf{P}_{i,j} = 0\ \forall\ (i, j) : |i-j| > 1$. Each state of M_σ represents a combination of resource requirements,



(a) Dumbbell Network (D).



(b) Triangle Network (T).



(c) Pyramid Network (P).

Fig. 6.4: Network topologies.

Tab. 6.4: Traffic flow requirements.

Service class	Parameter	Range of values	Unit	Sources
eMBB	η	$0.30 - 42.5$	Gbps	[200, 201]
	c	$50 - 100$	Gbps	[202, 203]
	m	$50 - 100$	Gb	[202, 203]
	δ	20	ms	[204, 205, 206, 207]
URLLC	η	$2.08 - 10$	Gbps	[200, 201]
	c	$50 - 100$	Gbps	[202, 203]
	m	$50 - 100$	Gb	[202, 203]
	δ	1	ms	[204, 205, 206, 207]

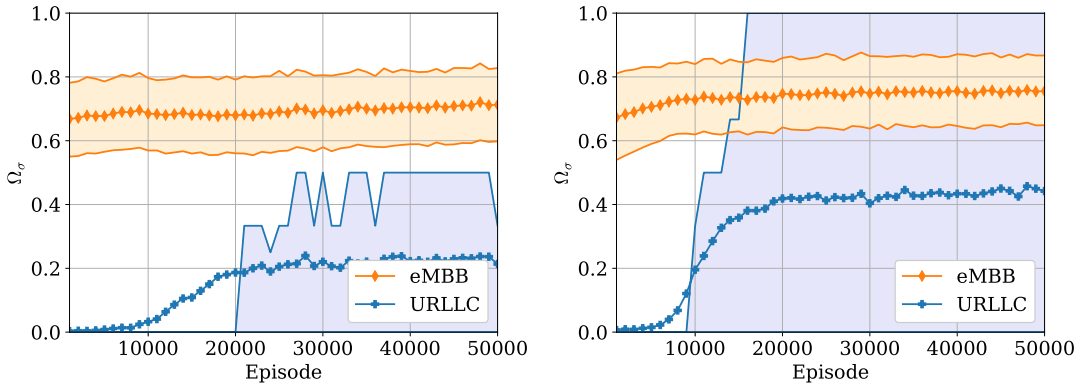
i.e., a different realization of the vector $\mathbf{r}$. Hence, each information flow $\phi \in \Phi_\sigma$ is associated to an independent copy of M_σ that changes state at each slot t, thus altering the vector $\mathbf{r}_\phi$. The minimum and maximum values of the resource requirements are given in Tab. 6.4.

We note that, in our simulations, the number of flows and their requirements change randomly so that the aggregate resource requests can exceed the network capacity. In such conditions, some flows will unavoidably experience low performance, while the allocation strategy should decide which flow to penalize to maximize the overall utility. Therefore, we can exploit such a system to handle critical scenarios where there is a lack of network resources or to estimate the reliability of a specific set of network slices. In the future, we will investigate call-admission control strategies to avoid system overloading.

To train the learning agents, we generate $N_{\text{train}} = 5 \cdot 10^4$ independent episodes using the same network topology. Each episode lasts $T = 50$ slots of $\tau = 0.1$ seconds. At the beginning of each episode, we generate a random number of information flows and associate each flow with a static route interconnecting its endpoints. Then, the A2C algorithm is used to train the learning agents Γ_σ^b, Γ_σ^c, Γ_σ^m, $\forall\ \sigma \in \Sigma$, according to the framework describe in Sec. 6.4. We exploit the Adaptive moment estimation (Adam) algorithm to optimize the NN weights, considering $\zeta_a = 10^{-5}$ and $\zeta_c = 10^{-5}$ as learning rates of the actor and the critic, respectively.

We note that the A2C algorithm updates the policies applied by the different agents only at the end of each episode (i.e., after a predetermined number of slots). Between two subsequent episodes, the different network elements forward local observations to the training manager depicted in Fig. 6.2. Hence, the new versions of the learning agents are shared within the network, updating the overall architecture. Besides, the exchange of control traffic among network elements is intrinsic to the SDN and NFV paradigms, which are becoming more and more widespread in modern networks. Therefore, communication within the learning units does not represent a limiting factor for the practical implementation of our approach.

6.6.2 Performance Analysis



(a) Dumbbell Network.

(b) Triangle Network.

Fig. 6.5: Training phase.

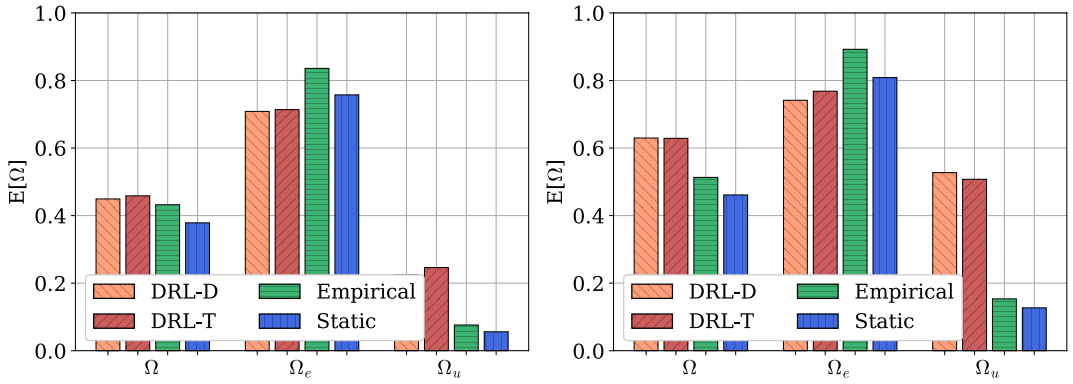
We consider two versions of our learning system, one trained in the Dumbbell Network (DRL-D) and the other in the Triangle Network (DRL-T). In Fig. 6.5 we plot the utility (for

the eMBB and URLLC slices) obtained during the training phase in the two scenarios, which makes it possible to assess the convergence time of the proposed framework.

In general, the convergence of a ML algorithm is determined by the slope of the *learning curve* given by the loss function of the trained model. In a DRL scenario, the convergence does not depend solely on the training loss but also on the *Q-values* associated with the different actions. Indeed, if the hierarchy between the Q-values does not change in time, the agent policy remains static, regardless of the learning process. Therefore it is more accurate to verify when the distribution of the agent actions becomes invariant. In such a case, the algorithm convergence coincides with the stabilization of the agent reward and the system utility.

In this work, we assume to reach convergence when the utility stabilizes, which means that any variation in the agents' actions does not lead to a better policy. We can appreciate that the average and the percentiles of the slice performance stop increasing after 30000 training episodes, which indicates that the agents have assumed a stable behavior after that point. We remark that the convergence of DRL systems cannot be proved mathematically unless under specific constraints. In particular, more complex scenarios (e.g., with a larger network topology) may exhibit longer convergence times. Therefore, network designers must accurately tune the number of training episodes case by case.

Looking at Fig. 6.5, we can observe that the performance of the eMBB slice (Ω_e) increases slowly but smoothly, ranging from 0.6 to 0.8 in 50% of the cases. Conversely, the performance of the URLLC slice (Ω_u) initially remains very low and suddenly increases after a certain number of episodes. Therefore, the agents need more time to learn how to address the requirements or the URLLC services. In particular, at the end of the training phase in the Triangle Network, Ω_u spans the full range of possible values, which indicates that the URLLC flows either get maximum or zero rewards in accordance with the step-like shape of their performance function. A similar phenomenon occurs in the Dumbbell Network where, however, the URLLC utility is lower due to the fewer available resources.



(a) Dumbbell Network. **(b)** Triangle Network.

Fig. 6.6: Expected utility.

To test the performance of our strategies, we carry out additional $N_{\text{test}} = 500$ episodes. In

Fig. 6.6, we report the expected utility achieved by each slice and by the whole system (see (6.6) and (6.5)) in the Dumbbell and Triangle Network for all the considered strategies. We can notice that, in both scenarios, the empirical algorithm tends to favor eMBB slices at the expense of URLLC flows, whose expected performance is always lower than 0.2. The static strategy behaves similarly but provides lower Ω_e than the empirical algorithm. In contrast, DRL-D and DRL-T slightly reduce the performance of the eMBB services but double the fraction of satisfied URLLC flows, significantly increasing the total utility.

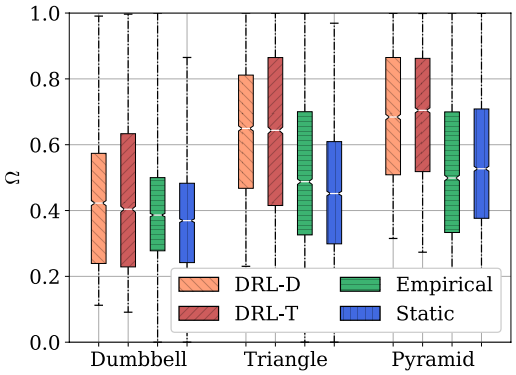


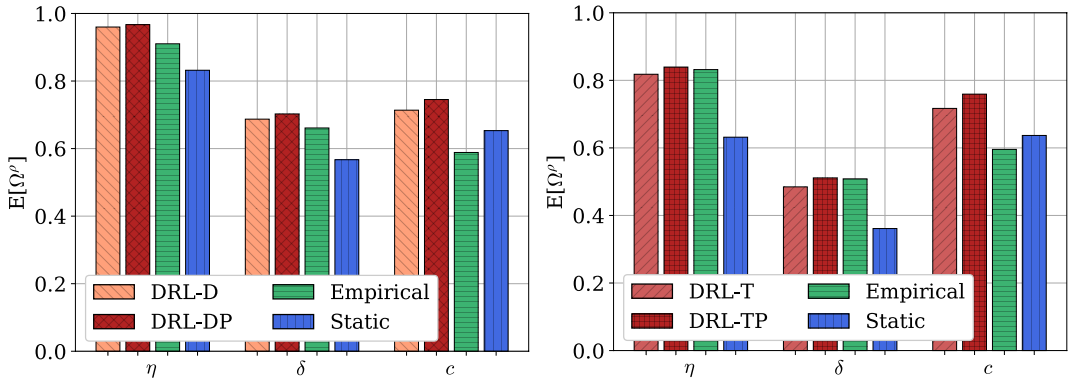
Fig. 6.7: Utility distribution.

In Fig. 6.7, we analyze the distribution of the system utility for all the different strategies and network scenarios. We adopt the boxplot representation, where the white line at the box center is the median of the distribution, and the box edges represent the 25th and the 75th percentile, respectively. It is clear that the static strategy always performs the worst because it cannot handle the variability of service demand, which is critical for URLLC traffic flows, whose performance goes to zero as soon as one requirement is not satisfied. The empirical algorithm works better but is still outperformed by the DRL strategies. In the Dumbbell Network, DRL-D and DRL-T ensure $\Omega > 0.4$ in almost 50% of the test episodes, with a 10% gain over the empirical algorithm. In the Triangle and Pyramid scenario, the lack of network resources is less striking and, consequently, the performance of all the strategies increases. In these scenarios, using the empirical algorithm, in 50% of the test episodes we get $\Omega > 0.5$ while, with the DRL strategies, we achieve $\Omega > 0.65$ and $\Omega > 0.7$, respectively.

We observe that the DRL strategies achieve similar results in all the testing scenarios, including the Pyramid Network, which differs from the environments seen during the learning phase. Therefore, the proposed distributed solution seems robust to topology changes, yielding reasonably good results in different scenarios without additional training. At the same time, we expect that a more specific learning framework can further improve the overall utility. In the rest of the section, we show how to leverage the TL paradigm for this purpose.

6.6.3 Transfer Learning

TL aims at speeding up the training of a ML algorithm by taking advantage of the structure learned by other algorithms trained in similar scenarios. In what follows, we exploit this technique to adapt our DRL strategy to different network topologies and traffic loads. Hence, we consider two new learning architectures, named DRL-DP and DRL-TP. We train these systems for $N_{\text{train}} = 3 \times 10^4$ episodes in the Dumbbell and Triangle Network, respectively, according to the framework described in Sec. 6.4. Then, they undergo additional training of $N_{\text{transfer}} = 2 \times 10^4$ episodes in the Pyramid Network.



(a) Pyramid Network. **(b)** Pyramid+ Network.

Fig. 6.8: Expected resource utility.

During the additional training, each controller is updated only with experience related to the specific network element it is associated with. For instance, a controller Γ_σ^b designed to manage a flow of class σ across a link l is trained using the state-action pairs for link l and information flows $\phi \in \Phi_\sigma$. In other words, each network element can autonomously update its own learning agents without involving the central manager depicted in Fig. 6.2. Besides, the training process does not require communication across the network and can be executed online, i.e., while the local controllers are being used in a real scenario. From a practical perspective, TL makes local controllers learn how to carry out more precise actions, thus increasing the overall utility. The drawback is that each controller can operate only in a specific location, making it impossible to implement the learning architecture in different network topologies.

The TL operations are repeated two times, considering different configurations for the URLLC services. First, we implement the same statistics presented in Tab. 6.4: in this case, the throughout required by each URLLC flow is in $[2.08, 10]$ Gbps. Then, we double the rate requirements of the URLLC slice (whose range of values becomes $[4.16, 20]$ Gbps) with the aim of assessing the ability of our strategy to adapt to new service specifications. In particular, we denote by Pyramid+ the scenario in which the required throughput of the URLLC flows is increased.

In Fig. 6.8a, we represent the expected performance obtained in the Pyramid Network by DRL-D and DRL-DP, and the benchmark strategies, while considering specific network re-

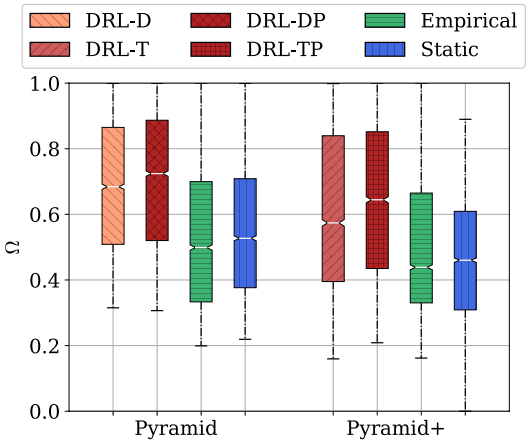


Fig. 6.9: Utility distribution.

sources. Hence, instead of using Ω to asses the system utility, we consider Ω^ρ, with $\rho \in \{b, c, m, \delta\}$, defined in (6.8). As expected, the algorithm obtained by exploiting TL performs better than that trained in the Dumbbell Network: for instance, using the DRL-DP strategy, the expectation of Ω^c increases by more than 5%. In this case, the performance gain is considerably lower than during the centralized training phase, and the system reaches convergence in about 10000 episodes. The higher speed in the system training is because the agents have already discovered a profitable policy and are just tuning their actions to the new scenario. In particular, the slice performance remains high during the whole TL phase, making it feasible to train the system online without degrading the user experience.

In Fig. 6.8b, we show the outcomes obtained in the Pyramid+ network scenario, considering DRL-T and DRL-TP. In this scenario, all the strategies yield a lower performance since the new URLLC requirements are more difficult to fulfill. Besides, we observe that the empirical strategy slightly outperforms DRL-T for what concerns the throughput and delay requirements. Nevertheless, DRL-T still provides good results, which means that the DRL approach is resilient to different traffic loads. Also in this case, TL proves to be worthwhile since DRL-TP outperforms the other algorithms in all three key performance indicators.

In Fig. 6.9, we report the distribution of the system utility in the Pyramid and Pyramid+ scenarios. We can appreciate how the algorithms obtained by TL always ensure higher utility, both considering the median and the percentiles of Ω. In particular, DRL-DP and DRL-TP outperform DRL-D and DRL-T, respectively, despite the total number of training episodes being the same for all the considered strategies.

To exemplify a possible application of the proposed solution in a real scenario, we consider a section of the high-capacity network interconnecting the main Italian universities and research centers owned by the national provider named GARR [209]. As depicted in Fig. 6.10, the considered network topology includes 19 core nodes, 10 edge nodes and 40 links. We assume that each link is provided with $B_l = 50$ Gbps of bandwidth capacity. Besides, we set the

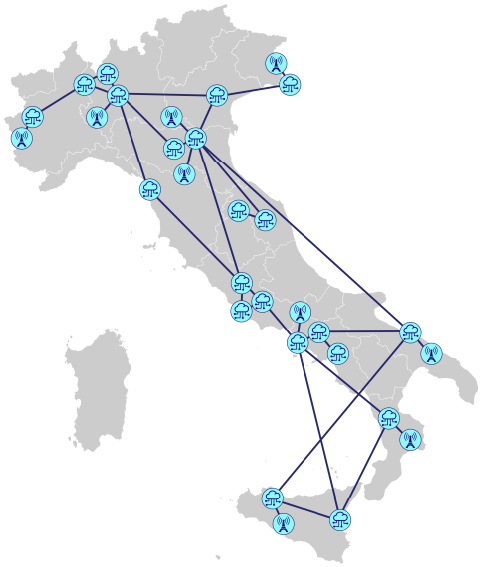


Fig. 6.10: GARR Network (Italy).

computational and memory capacities of the core nodes to $C_n^c = 30$ Gbps and $C_n^m = 30$ Gb, and those of the edge nodes to $C_n^c = 10$ Gbps and $C_n^m = 10$ Gb. * Finally, we relax the delay requirements of the eMBB and URLLC services, which are set to 100 ms and 5 ms, respectively. This choice allows us to deal with the longer routing paths, which yield higher end-to-end delays.

To implement our DRL strategy in this scenario, we carry out two consecutive learning stages: first, we train our system for 2×10^4 episodes using a centralized learning approach (as done for the DRL-D and DRL-T strategies). We recall that this phase can be (and shall be) performed offline, using all the experience gathered by all the network elements. Then, we carry out additional training of 10^4 episodes, defining a specific tuple of learning agents for each network element (as done for the DRL-DP and DRL-TP strategies). As already stated, this stage is performed online and does not require communication across the network.

In Fig. 6.11, we plot the expected performance of the eMBB and URLLC slices (i.e., $\mathrm{E}[\Omega_e]$ and $\mathrm{E}[\Omega_u]$) as a function of the number of information flows in the described scenario. When considering the eMBB flows, all the strategies have a similar behavior since $E[\Omega_e]$ declines gradually as the cardinality of Φ increases. Instead, the performance of the URLLC services is different depending on the chosen approach. The empirical algorithm maximizes the URLLC utility when $|\Phi| = 2$, while the same value abruptly drops to 0 as more flows are initialized over the network. Conversely, the static and DRL strategies ensure that $E[\Omega_u]$ follows a smoother trend.

In particular, the empirical algorithm outperforms the static approach when the number of information flows is limited ($|\Phi| \leq 3$). Beyond this point, it becomes more convenient to

*Note that, currently, the GARR network does not support NFV or SDN functionalities, so the resource capabilities considered in this study are realistic but arbitrary.

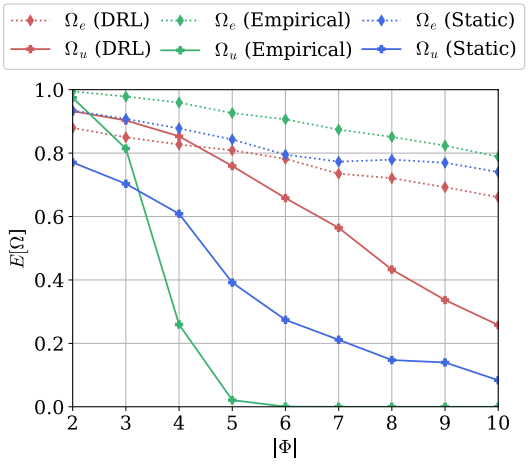


Fig. 6.11: Expected utility vs flow number.

maintain a static allocation of the system resources to prevent the degradation of the URLLC services. However, both benchmarks are outperformed by our DRL strategy, which gives priority to the URLLC services, ensuring a better total utility. In particular, using the empirical and the static strategies, $E[\Omega_u]$ falls below 0.3 for $|\Phi| = 6$, while DRL ensures $E[\Omega_u] > 0.6$ in the same conditions.

6.7 Conclusions and Future Work

In this chapter, we investigated the potential of DRL to orchestrate network resources in a NS scenario. Specifically, we developed a distributed DRL architecture where different units interact to meet the resource demands of multiple information flows. By means of simulations, we showed that the designed strategy consistently improves the management of network resources, especially when the system complexity, both in terms of network topology and service heterogeneity, increases. Our approach makes it possible to double the number of URLLC flows supported by the network without degrading the performance of the eMBB flows. Besides, we proved that TL further improve the behavior of the learning agents, thus increasing the overall utility at the cost of reduced adaptability of the agents to different network topologies.

As part of future work, we are interested in extending our NS model by considering more slice classes with different specifications. Hence, we want to test our framework with real communication traces to identify its potential limits. Finally, we will investigate the possibility of introducing additional learning units trained to coordinate the local controllers of our architecture, e.g., by varying the routing paths of the traffic flows.

7

Balancing Learning and Exploitation at the Network Edge

7.1 Introduction

The orchestration of next-generation mobile networks is beyond the capabilities of human-designed algorithms, as it is characterized by multiple objectives and fast dynamics, with several classes of traffic having specific activity patterns and Quality of Service (QoS) guarantees [167]. As we already saw in this dissertation, machine learning is essential to allow network protocols to dynamically adapt to different scenarios without manually reconfiguring the entire system [211]. In particular, the combination of Reinforcement Learning (RL) principles with deep learning, also known as Deep Reinforcement Learning (DRL) [118], is one of the most promising tools for the optimization of 5G and beyond networks [212]. However, the training of DRL models in complex environments is still very computationally expensive [213], and it is often impossible to perform it in advance because of the rapid changes in future mobile networks. Therefore, it is fundamental for 5$^{\text{th}}$ Generation (5G) and beyond systems to support online training on the network edge, exploiting a Continual Learning (CL) approach [214]. In this context, the training updates are either performed directly on the edge nodes by the Mobile Edge Computing (MEC) paradigm or offloaded to more powerful Cloud servers, reducing the consumption of local resources but requiring the transmission of large amounts of data.

In a MEC scenario, the training cost creates a fundamental trade-off: updating a DRL model with new experience can improve the learned policy, increasing its efficiency in the target task, but also subtracts some resources from that very same task. Disregarding this trade-off and assuming that training is a *free action* may have serious consequences, degrading the perfor-

The work presented in this chapter was published in the proceedings of *2022 IEEE International Conference on Communications* [210].

mance in the system that should be optimized [215]. The machine learning research community is starting to become aware of this issue [216], focusing on model compression and lightweight learning techniques to reduce the burden on the edge hardware [217] and considering the *cost of learning* in the design of neural networks meant to operate on resource-constrained devices [218]. In particular, the Federated Learning (FL) approach can reduce the computational load of learning systems by conducting the algorithm training in a distributed fashion [219]. In the past years, many frameworks to reduce the computation and communication cost of FL have been proposed: for a deeper review of these topics, we refer the reader to [220, 221].

To the best of our knowledge, the resource efficiency of DRL techniques in future network scenarios is a relatively unexplored topic [222], as most researches in the literature still neglect the cost of the training, separating the learning process from the optimization even in online applications. In this chapter, we attempt to model the cost of learning in an explicit manner by defining an optimization problem that balances learning and network functions. The objective is to identify learning strategies that maximize the system performance during training, accounting for the learning cost in terms of bandwidth or other types of resources. To this end, we design a theoretical model to represent a general telecommunication system whose resources are contended among user applications and learning agents. The increase of user resources enables to improve the immediate performance but prevents the agents from innovating their policies and gaining higher benefits in the future. Therefore, determining the optimal resource allocation is not trivial and depends on the specific working conditions of the target system. In particular, we implement the described model in a Network Slicing (NS) scenario, where the same channel is used to forward the data of end users and learning algorithms from a Base Station (BS) to a Core Network (CN). Our results show that adapting the number of training updates is essential for maximizing the user performance over a long time horizon while considering an ideal case with free learning actions leads to a significant overestimating of the system performance.

We organize the rest of this chapter as follows: Sec. 7.2 presents the cost of learning problem; Sec. 7.3 applies the proposed model in a NS use case; Sec. 7.4 describes the settings and results of our work; finally, Sec. 7.5 concludes the chapter and presents some possible avenues for future work.

7.2 Cost of Learning Model

We consider a network management scenario in which a learning agent tries to maximize system performance by allocating communication or computational resources to different users. To address such a scenario, we adopt a RL approach and, as explained in Appendix B, we discretize the time into slots $t \in \mathbb{Z}^+$. At any slot, the agent chooses a new action according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and receive a reward $r(t) \in \mathbb{R}$. In particular, $\pi(s, a)$ is the probability that the agent chooses action $a \in \mathcal{A}$ when the system is in state $s \in \mathcal{S}$, where $\mathcal{A}$ and $\mathcal{S}$ are the agent action and state space, respectively. The goal is to compute the optimal policy π^*, which

maximizes the expected discounted return $G(t)$:

$$G(t) = \mathbb{E}\left[\sum_{\tau=t}^{\infty} \lambda^{\tau-t} r(s(\tau), s(\tau+1), a(\tau)) \;\middle|\; \pi, s(t)\right], \tag{7.1}$$

where $\lambda \in [0, 1)$ is the so-called *discount factor*.

As in classical RL settings, we consider that the system evolution is organized into episodes $k = 0, 1, ...$, where each episode lasts T slots. We assume that the network dynamics change over time, which makes it impossible to explore the environment offline, i.e., collect data with a pre-determined policy and perform training on this static dataset before the agent deployment. In particular, the state transition probability $P(\cdot)$ and the reward function $r(\cdot)$ change over time in a step-wise manner, i.e., after a *coherence period* of K episodes. We can assume that the agent is aware of the environment's evolution and reinitializes its policy at the end of every coherence period. Accelerating the training might be possible by exploiting Transfer Learning (TL), using previous experience to avoid periodically starting a new learning process from scratch [223]. However, the design of such techniques is out of the scope of this chapter.

We define the expected reward for an episode as

$$\overline{R}(k) = \mathbb{E}\left[\sum_{t=0}^{T-1} \frac{r_{a(t)}(s(t), s(t+1))}{T}\right]. \tag{7.2}$$

Our goal is maximizing the expected reward $\overline{R}_K = \sum_{k=0}^{K-1} \overline{R}(k)$ during the K episodes constituting a single coherence period. To this end, we want to make the agent policy converge to the optimum in the shortest time possible. However, in our scenario, the agent training consumes resources (computational or communication, depending on the case) that should be assigned to the users. Hence, each learning update directly impacts the system performance, and there is a trade-off between convergence speed and cost of learning. Our problem is similar to the well-known dilemma between exploration and exploitation but in a different context. Adapting the amount of exploration cannot solve the problem of training costs as it increases the *variety* of the collected experience, but does not reduce their communication and computational costs.

To analyze the cost of learning, we divide each episode into two subsequent phases. During the first, named *exploitation* phase, the agent applies the learned strategy to both explore the environment and improve the system performance. During the second, named *update* phase, the available resources are fully devoted to the agent training. The order of these two phases is unimportant, as the update phase can be performed at the beginning of the episode with the same results, and they last T_π and $T_\rho = T - T_\pi$ slots, respectively, so that the agent's actions are determined by the learned policy π during the first T_π slots of any episode k, while the agent follows a pre-determined policy ρ during the last T_ρ slots. In other words, the probability

of choosing action a in state s is given by

$$\begin{cases} \pi(a,s), & t \in \{0,1,...,T_\pi - 1\}; \\ \rho(a,s), & t \in \{T_\pi, T_\pi + 1, ..., T-1\}, \end{cases} \tag{7.3}$$

where t is the slot index within the same episode.

The policy ρ ensures that all or part of the system resources are used for the agent training, making it possible to update the policy π with the new experience the agent has gained. On the other hand, since ρ subtracts some of the resources from the users, this strategy leads to sub-optimal performance, decreasing the reward collected during the episode.

From a practical perspective, the value of T_ρ determines the amount of time and resources devoted to the learning process. As T_ρ increases, so does the number of experience samples used to train the agent after each episode. In doing so, we make it possible to accelerate the convergence speed of the agent, thus improving the immediate reward gained during the exploitation phases of the next episodes. On the other hand, increasing T_ρ also shortens the average reward since the system spends more time in the update phases in which the sub-optimal policy ρ is used. In particular, the average reward during the k-th episode, considering a fixed T_ρ, is given by:

$$\overline{R}(k|T_\rho) = \left(\frac{T - T_\rho}{T} \overline{R}_{\pi_k}(k) + \frac{T_\rho}{T} \overline{R}_\rho(k) \right), \tag{7.4}$$

where π_k is the policy learned by the agent in the k-th episode, while $\overline{R}_{\pi_k}(k) = \mathbb{E}[\overline{R}(k); \pi_k]$ and $\overline{R}_\rho(k) = \mathbb{E}[\overline{R}(k); \rho]$ are the expected average rewards gained using policies π_k and ρ, respectively. The optimal value of T_p depends on multiple factors, including the coherence time of the underlying non-stationary Markov Decision Process (MDP).

In the rest of the chapter, we consider two possible approaches for balancing exploitation and training. First, we adopt the naive assumption that the amount of resources assigned to the learning task is constant over time, and we analyze how T_ρ affects the system performance. In the second approach, instead, we adapt the number of training resources over time depending on the convergence speed of the learning agent, and we study the trade-off between convergence speed and the effectiveness of the learned strategy. In the following, we formally define the optimization problems underlying the two approaches, which will be successively analyzed and compared in Sec. 7.4, considering a MEC system as a use-case scenario.

7.2.1 Constant Update Duration

Given the coherence period duration K, the first optimization problem aims at determining the optimal T_ρ to maximize $\overline{R}_K$ while assuming that T_ρ is constant over time:

$$T_\rho^* = \underset{T_\rho \in \{0,...,T\}}{\arg\max} \left(\sum_{k=0}^{K-1} \frac{\overline{R}(k|T_\rho)}{K} \right). \tag{7.5}$$

7.2.2 Adaptive Update Duration

The second approach assumes that the agent can determine when the policy π converges to the optimal one. In particular, after discovering the optimal policy, the agent sets T_ρ to zero, fully exploiting the system resources. If we define the number of episodes until convergence as $\eta(T_\rho)$, we have the following optimization problem:

$$T_\rho^* = \underset{T_\rho \in \{0,\dots,T\}}{\arg\max} \left(\sum_{k=0}^{\eta(T_\rho)-1} \frac{\overline{R}(k|T_\rho)}{K} + \left(1 - \frac{\eta(T_\rho)}{K}\right) \overline{R}_{\pi^*} \right). \tag{7.6}$$

In this case, we expect T_ρ^* to be higher since a quicker convergence to the optimal policy allows the system to terminate the training process and fully dedicate its resources to the users.

7.3 Use Case Scenario

To test the benefits of our cost-aware learning framework, we consider a NS scenario, where a set of users with heterogeneous requirements transmits data through the uplink. We assume to divide the traffic into *slices* that depend on the applications' QoS requirements, and the BS needs to allocate the capacity of a backhaul link to guarantee the best possible performance for each slice. The link resources are managed by a DRL agent, whose actions depend on the state of the slice buffers at the BS. For the reader's convenience, we report the main parameters of our model in Tab. 7.1.

Tab. 7.1: Model parameters.

Parameter	Description	Parameter	Description
t	Timeslot index	τ	Timeslot duration
T	Slots per episode	K	Coherence period
$r(\cdot)$	Reward function	$\overline{R}(k)$	Mean reward at episode k
$\mathcal{S}$	Agent state space	$\mathcal{A}$	Agent action space
T_π	Exploitation phase duration	T_ρ	Update phase duration
π_k	Learned policy at episode k	ρ	Update policy
Σ	Slice set	N_σ	Resource blocks assigned to σ
L_{pkt}	Packet length	$\delta(p,t)$	Delay of packet p at time t
$\mathcal{U}$	User set	$\mathcal{U}_\sigma$	Users associated with slice σ
C_{bh}	Total link capacity	N	Number of resource block
$\mathbf{x}_u$	Packet sent by u to the BS	$x_u(t,m)$	m-th packet in $\mathbf{x}_u(t)$
$\mathbf{q}_\sigma$	Packet collect in the buffer of slice σ	Q	Maximum buffer size
$\mathbf{y}_\sigma$	Delivered packet of slice σ	Y_σ	Size of $\mathbf{y}_\sigma$
$\mathbf{z}_\sigma$	Discarded packet of slice σ	Z_σ	Size of $\mathbf{z}_\sigma$
$f_u(\cdot)$	Utility function of the packets of user u	Δ_u	Packet delay budget of user u
Ω_u	Performance of user u	Ω	Overall system performance

7.3.1 Communication Model

We assume that time is discretized into slots $t \in \mathbb{Z}^+$ of length τ and that, in each slot t, each user $u \in \mathcal{U}$ (corresponding to a single application) transmits a vector of packets $\mathbf{x}_u(t)$ to the BS. All packets have the same length L_{pkt}, and we denote by $x_u(t,m)$ the m-th packet of $\mathbf{x}_u(t)$.

We also assume that each application is associated with a specific slice $\sigma \in \Sigma$, according to the application requirements. We denote by $\mathcal{U}_\sigma \subset \mathcal{U}$ the set of users associated with slice σ. In particular, all the packets belonging to the same slice σ are seen by the BS as a single stream of data sharing the same communication resources.

We assume that the BS maintains a *first in first out* buffer with a maximum size of Q packets for each slice $\sigma \in \Sigma$. The packets present in the buffer for slice σ at the beginning of slot t are collected in vector $\mathbf{q}_\sigma(t)$. We can assume that packets are added to the backhaul link buffer at the end of each slot. The buffer size condition is then $|\mathbf{q}_\sigma(t)| \leq Q$, where $|\mathbf{x}|$ represents the length of vector $\mathbf{x}$.

We assume that the backhaul link has a total capacity of C_{bh} divided into N resource blocks, each of which makes it possible to transmit $\frac{\tau C_{\text{bh}}}{N}$ bits per slot. In particular, the system can transmit packets over multiple subsequent slots. We now denote by $N_\sigma(t)$ the number of resource blocks assigned to slice σ during slot t. Naturally, any resource allocation scheme should comply with the condition $\sum_{\sigma\in\Sigma} N_\sigma(t) \leq N$.

Given that the agent allocates $N_\sigma(t)$ backhaul resources to slice σ, the number of packets from σ that can be delivered in slot t is then given by:

$$Y_\sigma(t) = \min\left(|\mathbf{q}_\sigma(t)|, \left\lfloor N_\sigma(t)\frac{\tau C_{\text{bh}}}{L_{\text{pkt}}N}\right\rfloor\right). \tag{7.7}$$

We denote by $\mathbf{y}_\sigma(t)$ the vector containing the $Y_\sigma(t)$ packets of slice σ that are transmitted during slot t. At the next step, the queue vector $\mathbf{q}_\sigma(t+1)$ contains the remaining queued packets, as well as the newly arrived ones. However, the queue cannot contain more than Q packets, and in case of buffer overflows, the oldest $Z_\sigma(t)$ packets are then discarded:

$$Z_\sigma(t) = \max\left(0, |\mathbf{q}_\sigma(t-1)| - Y_\sigma(t-1) + |\mathbf{x}_\sigma(t)| - Q\right). \tag{7.8}$$

We denote the vector of discarded packets by $\mathbf{z}_\sigma(t)$, so that $|\mathbf{z}_\sigma(t)| = Z_\sigma(t)$. Once the buffer states $\mathbf{q}_\sigma(t)$, $\forall\sigma \in \Sigma$ have been updated, the backhaul resources can be reallocated. To evaluate the system performance, we consider the queuing delay, defined as:

$$\delta(p,t) = \begin{cases} s : p \in \mathbf{x}_\sigma(t-s), & p \in (\mathbf{y}_\sigma(t) \cup \mathbf{q}_\sigma(t))\,; \\ \infty, & p \in \mathbf{z}_\sigma(t), \end{cases} \tag{7.9}$$

where the delay is computed for packet p from slice σ at time t. We define the delay as the number of slots since p was generated if the packet is transmitted or still in the buffer. Instead, we assume that the delay is infinite if p is discarded.

We then define a utility function $f_u(\cdot)$ for each user, which takes the packet delay as input and return a value in $[0,1]$. In particular, $f_u(\cdot)$ is monotonically decreasing, with $f_u(0) = 1$ and $\lim_{x\to\infty} f_u(x) = 0$, and depends on the delay requirements of the user, expressed in terms of the maximum delay Δ_u. Finally, the performance $\Omega_u(t)$ of u and the overall system performance

$\Omega(t)$ at slot t are given by:

$$\Omega_u(t) = \sum_{m=1}^{Y_\sigma(t)} \frac{f_{u(y_\sigma(t,m))}(\delta(y_\sigma(t,m),t))}{(Y_\sigma(t)+Z_\sigma(t))}, \tag{7.10}$$

$$\Omega(t) = \frac{1}{|\Sigma|}\sum_{\sigma\in\Sigma}\frac{1}{|\mathcal{U}_\sigma|}\sum_{u\in\mathcal{U}_\sigma}\Omega_u(t), \tag{7.11}$$

where $u(p)$ is the user that sent packet p.

7.3.2 Learning Framework

In order to optimize the system in a foresighted manner, we model the resource allocation problem as an MDP, implementing a DRL agent to manage the link resources. In particular, the state $s(t)$ at time t depends on the individual requirements of the users currently being served and the transmission buffer for each slice. Specifically, $s(t)$ is a tuple with $4\cdot|\Sigma|$ elements, namely:

- The number of packets contained in each slice buffer, i.e., $|\mathbf{q}_\sigma(t)|$, $\forall\ \sigma\in\Sigma$;
- The average remaining time before the packets contained in each slice buffer exceed the maximum allowed delay Δ_u, which is given by

$$\sum_{m=1}^{|\mathbf{q}_\sigma(t)|}\frac{\Delta_{u(q_\sigma(t,m))}-\delta(q_\sigma(t,m))}{|\mathbf{q}_\sigma(t)|},\ \forall\sigma\in\Sigma; \tag{7.12}$$

- The minimum remaining time among the packets in each slice buffer, i.e.,

$$\min_{m\in\{1,\dots,|\mathbf{q}_\sigma(t)|\}}\left(\Delta_{u(q_\sigma(t,m))}-\delta(q_\sigma(t,m))\right),\ \forall\sigma\in\Sigma; \tag{7.13}$$

- The number of packets that will be transmitted during the current slot for each slice, assuming that the resource allocation scheme does not change, which is:

$$\min\left(|\mathbf{q}_\sigma(t)|,\left\lfloor N_\sigma(t-1)\frac{\tau C_{\text{bh}}}{L_{\text{pkt}}N}\right\rfloor\right),\ \forall\sigma\in\Sigma. \tag{7.14}$$

We have $\mathcal{S}=\left\{0,\dots,\frac{Q}{L_{\text{pkt}}}\right\}^{2\cdot|\Sigma|}\times\mathbb{R}^{2\cdot|\Sigma|}$, while the action space $\mathcal{A}$ includes $1+|\Sigma|\cdot(|\Sigma|-1)$ different actions. Specifically, action 0 maintains the resource allocation constant, so that $N_\sigma(t)=N_\sigma(t-1)$, $\forall\ \sigma\in\Sigma$. Instead, each of the remaining actions is defined by the ordered tuple (i,j), $i\neq j$, and corresponds to taking one resource block from slice σ_i and assigning it to slice σ_j, so that $N_{\sigma_i}(t)=N_{\sigma_i}(t-1)-1$ and $N_{\sigma_j}(t)=N_{\sigma_j}(t-1)+1$. Naturally, resource allocation can never be negative, i.e., $N_\sigma(t)\geq 0\ \forall\sigma\in\Sigma$, and the total number of allocated resources is always N.

Hence, at each slot t, the agent observes state $s(t) \in \mathcal{S}$ and selects a new action $a(t) \in \mathcal{A}$ according to its current policy π, which determines the number of blocks $N_\sigma(t)$ assigned to each slice $\sigma \in \Sigma$. Then, the agent receives an instantaneous reward equal to $\Omega(t)$, i.e., the system utility defined in (7.11); we note that, by definition, $\Omega(t) \in [0, 1]$.

Tab. 7.2: Agent architecture.

Layer size (inputs × outputs)	Inter-layer operations
$4 \cdot \lvert\Sigma\rvert \times 64$	ReLU activation
64×32	ReLU activation
$32 \times (1 + \lvert\Sigma\rvert \cdot (\lvert\Sigma\rvert - 1))$	Linear activation

We adopt the Deep Q-Network (DQN) approach [118], in which the expected long-term value of each action, as given by (7.1) with $\Omega(t)$ as reward, is approximated by a Neural Network (NN) that, at each slot t, takes the current state $s(t)$ as input. In particular, we implement a fully connected feed-forward NN, whose input layer is formed by one neuron for each element of $s(t)$. The output of the NN is a scalar vector of size $|\mathcal{A}|$, representing the expected long-term reward of each possible action $a \in \mathcal{A}$ for the current state. In particular, when operating greedily, the agent will always pick the action corresponding to the highest output value. The main parameters of the learning architecture are reported in Tab. 7.2.

7.4 Simulation Settings and Results

Tab. 7.3: System settings.

Parameter	Value	Description	Parameter	Value	Description
τ	10 ms	Timeslot duration	T	1000	Slots per episode
K	10000	Coherence period	T_{agent}	10 slot	Interval between agent actions
N	10	Number of resource blocs	C_{bh}	1 Mb/s	Total link capacity
L_{pkt}	512 b	Packet length	Q	100	Maximum buffer size
$\lvert\mathcal{U}\rvert$	5	User number	$\lvert\Sigma\rvert$	5	Slice number
L_{exp}	704 b	Experience sample length	L_{NN}	92256 b	Learning model length
λ	0.95	Discount factor	ζ	10^{-5}	Maximum learning rate
T_ρ	$\{1, 2, 3, 4, 5\}$ slot	Update phase duration	K_{avg}	4000	Window for reward computation

In the following, we apply our model in the NS environment described in Sec. 7.3. Specifically, we assume that the DRL agent deployed at the BS cannot be trained locally and needs to exchange information with the core network to improve its policy. Part of the backhaul resources are then used to update the agent's architecture, possibly degrading the user performance. Hence, we investigate how the system utility changes when varying the resources dedicated to the training, both when the learning update is performed regularly and when it is stopped after a certain period (assuming the DRL agent has converged to the optimal policy). For the reader convenience, we resume the settings of our simulation in Tab. 7.3.

7.4.1 System Settings

The model described in Sec. 7.3.1 is very general and can suit multiple communication scenarios with different characteristics. In this work, for the sake of simplicity, we consider a simple case with only two slices, named *non-critical* (σ_{NC}) and *critical* (σ_{C}), respectively, with the following performance functions $f_u(\delta)$:

$$f_u(\delta) = \begin{cases} \min\left(1, \frac{\Delta_u}{\delta}\right), & \text{if } u \in \sigma_{\mathrm{NC}}; \\ \mathbb{1}(\Delta_u - \delta), & \text{if } u \in \sigma_{\mathrm{C}}; \end{cases} \tag{7.15}$$

where $\mathbb{1}(x)$ is the limit-step function, equal to 1 if $x \geq 0$ and 0 otherwise. Critical packets have a hard deadline, i.e., delivering them after the maximum delay Δ_u gives zero performance benefits. On the other hand, the utility of non-critical packets decreases gradually as the delay grows past the deadline.

We divide time into slots of $\tau = 10$ ms and that the backhaul link has a total capacity $C_{\mathrm{bh}} = 1$ Mb/s. The channel is divided into $N = 10$ resource blocks so that each block allows the delivery of exactly 1 kb. However, we assume that any allocation scheme remains constant over $T_{\mathrm{agent}} = 10$ slots, i.e., the agent takes a new action every 100 ms. This value is closer to the granularity of actual schedulers and avoids the *reward tampering* phenomenon [224]. We consider packets to have a constant length $L_{\mathrm{pkt}} = 512$ b and set the maximum buffer size to $Q = 100$ packets, equivalent to 64 kB. Besides, we assume that there are 5 different users, randomly associated with the critical and non-critical applications at the beginning of each episode. The Non-Critical Voice (NCVO) and Non-Critical Video (NCVI) applications are associated with σ_{NC} and have constant bitrate. The Critical Voice (CVO) and Critical Video (CVI) applications are associated with σ_{C} and generate new packets following an on-off process.

Tab. 7.4: Application requirements.

Application	Bit rate [kb/s]	Packet delay budget [ms]
NCVO	25	100
NCVI	384	300
CVO	25 (when active)	75
CVI	384 (when active)	100

More specifically, CVO and CVI behave according to a Markov chain with two state, namely *silent* (s) and *active* (a): in the silent states, no packets are generated, which implies that $|\mathbf{x}_u(t)| = 0$; conversely, in the active states, packets are generate with constant bitrate. We assume that a critical application can switch between states at the beginning of each slot t, and we set the state transition probability to $p_{ss} = p_{aa} = 0.9$ and $p_{sa} = p_{as} = 0.1$, respectively. The bitrate and the packet delay budget of the different applications are summarized in Tab. 7.4.

To orchestrate the backhaul link resources, we deploy the learning system presented in Sec. 7.3.2. Our implementation of DQN is distributed. The *inference network* is installed on the BS and is used to allocate resources in the exploitation phase of each episode. Instead, the *training network* is updated at every training step and is in a Cloud server. The transmis-

sion of the learning data and model occupies the backhaul link, following the general framework defined in Sec. 7.2.

At the beginning of each episode, we empty the slice buffers and associate each user with a random application. During the first T_π slots of each episode, the inference network sets the resource allocation policy and saves experience samples $(s(t), a(t), s(t+1), \Omega(t), a(t+1))$ in a local memory. Instead, during the training phase, no application data are transmitted, i.e., $N_\sigma(t) = 0$, $\forall\ \sigma \in \Sigma$, and we exploit the overall link capacity to exchange learning data over the channel. We assume that each experience sample can be encoded into $L_{\text{exp}} = 704$ b, while the NN architecture size is $L_{\text{NN}} = 92256$ b. Therefore, during the updating phase, $T_\rho \frac{\tau C_{\text{bh}}}{L_{\text{exp}}}$ transitions can be forwarded through the channel and used by the Cloud server to update the training network. Moreover, for every 10 learning steps, a copy of the training network is sent back to the BS, replacing the inference network and improving the current policy ρ. In those steps, the number of transitions forwarded through the channel is limited to $\frac{T_\rho \tau C_{\text{bh}} - L_{\text{NN}}}{L_{\text{exp}}}$.

To train the agent during the update phase, we implement the *on-policy* State-Action-Reward-State-Action (SARSA) algorithm [110] with a *softmax* exploration policy. We set the discount factor $\lambda = 0.95$, and we implement the Adaptive moment estimation (Adam) algorithm to optimize the NN weights, using $\zeta = 10^{-5}$ as maximum learning rate. We also include a penalty for the entropy of the output distribution, encouraging the exploration of the action space without setting a fixed temperature profile for the softmax function. Finally, we set the coherence period to $K = 10000$ episodes and assume that each episode lasts $T = 1000$ slots.

7.4.2 Performance Analysis

We now investigate different configurations of our learning system, varying the time $T_\rho \in \{1, \dots, 5\}$ devoted to the agent training. When T_ρ is low, most resources are assigned to the users, and agent training proceeds slowly. Conversely, as T_ρ increases, more learning data are transmitted through the link, taking up more resources but increasing the training speed. We compare the results with an ideal system, where all the learning transitions are instantaneously transmitted through a side-channel and used to update the inference network without impacting the users. Clearly, this represents an upper bound to the practically achievable performance.

We first consider the scheme with a constant update duration from Sec. 7.2.1. In Fig. 7.1, we represent the average performance during the training of the different strategies, obtained by aggregating the data of 100 independent simulations. It is easy to see that larger values of T_ρ lead to a quicker convergence toward the optimal policy and, in particular, training is rather slow if $T_\rho = 1$. On the other hand, the configurations with a higher T_ρ improve faster but have lower performance after convergence since they continue to devote a fixed amount of resources to the agent training.

We can have a better insight into the different strategies by looking at Fig. 7.2, which uses boxplots to represent the performance statistics during the beginning, intermediate and last episodes of the training phase. The whiskers represent the 5th and 95th percentiles of the performance distribution, while the box goes from the 25th to the 75th. In the first third of the

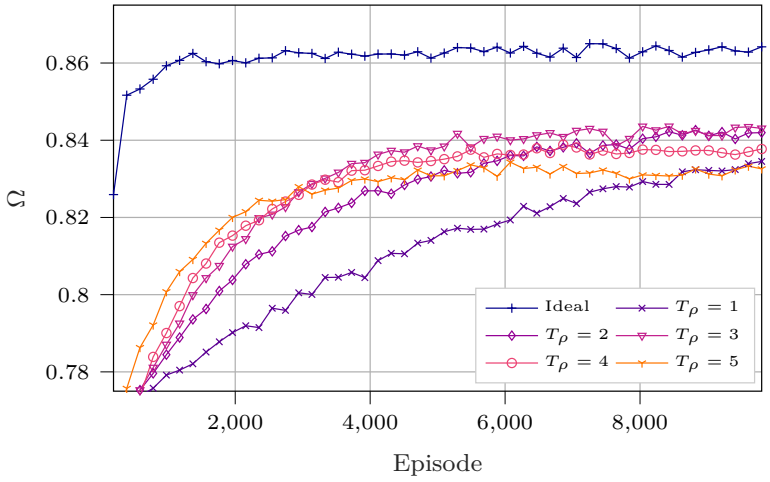


Fig. 7.1: Mean performance over time with fixed T_ρ.

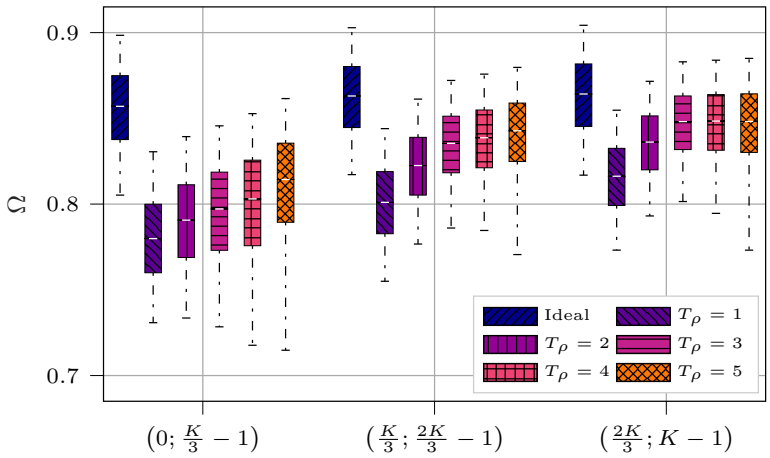


Fig. 7.2: Boxplots of the performance with fixed T_ρ.

coherence period, it is convenient to select a high value for T_ρ to learn the optimal policy faster. After a sufficient number of episodes, the configurations with a lower T_ρ also converge and waste fewer system resources on further training, but the configuration with $T_\rho = 1$ is outperformed by the others, as it needs more than K episodes to reach convergence.

We can now look at the adaptive system from Sec. 7.2.2. We use a simple heuristic strategy to infer convergence. If the average reward over a rolling window of K_{avg} episodes stops increasing, the agent estimates that the optimal policy has been found and sets T_ρ to zero, assigning the whole capacity to the users. This technique allows the agent to use the first portion of the coherence period to learn the optimal policy, taking advantage of the acquired knowledge in the remaining episodes. Therefore, the choice of T_ρ will determine the time needed to switch to the exploitation phase.

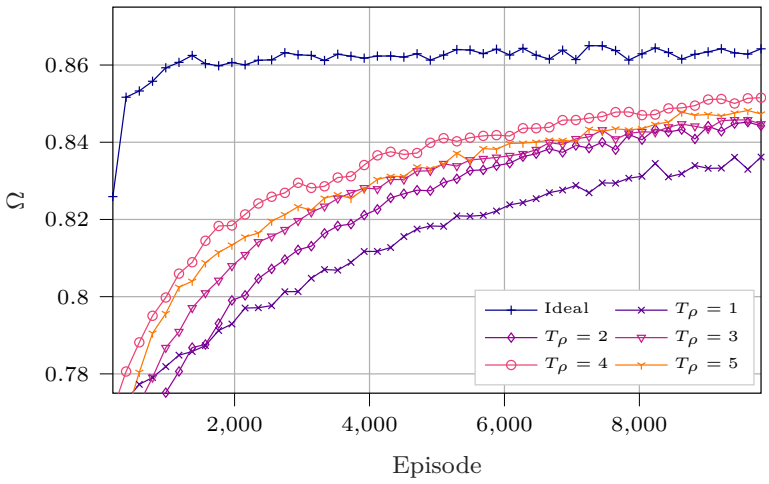


Fig. 7.3: Mean performance over time with adaptive T_ρ.

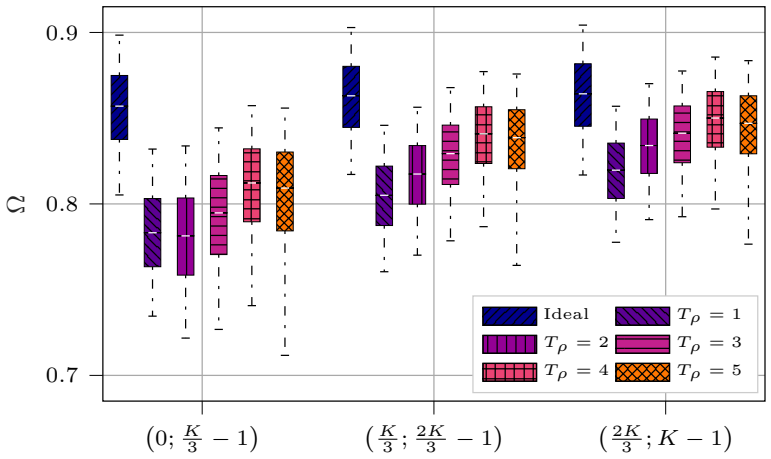


Fig. 7.4: Boxplots of the performance with adaptive T_ρ.

Fig. 7.3 shows the performance over time with this approach, considering $K_{\text{avg}} = 4000$. We observe that the performance of the strategies with $T_\rho \leq 3$ is almost identical to that of the previous scenario. The adaptive method does not reach convergence before the end of the coherence period with such a small T_ρ. On the other hand, setting $T_\rho = 5$ seems too aggressive, and the heuristic stops the learning process too soon, leading to suboptimal results. The approach with $T_\rho = 4$ outperforms all the other strategies during the entire coherence period, striking a balance between convergence speed and cost of learning. We confirm these findings by looking at Fig. 7.4, which shows that $T_\rho = 4$ achieves better performance, both considering the lower and the higher percentiles of the distribution.

7.5 Conclusions and Future Work

In this chapter, we analyzed the cost of exploiting DRL solutions for MEC network optimization. Specifically, we designed a novel cost of learning framework to optimize the number of resources that a learning agent allocates to its own improvement to balance the speed of convergence of the policy with the system performance during the training. We consider a test case based on a 5G system in which a DRL agent has to allocate the bandwidth of a backhaul link among multiple slices while consuming part of the network resources to transmit its training updates. Our results show a significant trade-off between the convergence speed and the communication overhead due to the training, stressing the importance of inferring agent convergence.

Future work on the subject may involve more general solutions based on hierarchical DRL or meta-learning techniques that can apply to different problems in which the cost of learning is a concern. In particular, we consider promising to combine our approach with adaptive exploration schemes, avoiding unnecessary exploration when the communication bottleneck limits the training process.

8 Conclusions

In this thesis, we tried to provide a comprehensive framework that can handle various aspects of dynamic networks, from topology mapping to resource distribution. The proposed architecture is based on a distribution of network functionalities among multiple intelligent units and leads to consistent advantages in numerous fields, from autonomous cars to Unmanned Aerial Vehicle (UAV) swarm applications. According to our vision, telecommunication systems must abandon their monolithic architecture in favor of solutions based on higher flexibility, capable of self-tuning and addressing the immediate needs of the end users. This shift enables new and specific services, from remote surgery to wildfire monitoring, over the current network facilities, accelerating technological progress toward the fourth industrial revolution. In particular, telecommunication systems must not solely keep pace with the current technology but stand ready to support innovative applications that we cannot even conceive at this moment. To this end, we should base network management on extremely adaptable protocols and techniques, promoting the continuous evolution of telecommunication technology according to human desires.

Inside the thesis, we addressed three different and complementary aspects of the orchestration of dynamic networks. At first, we focused on designing a new tracking framework to build accurate network maps while minimizing the consumption of communication resources. In our approach, a centralized entity collects information and tracks the overall network evolution while the communication control is distributed among all the network nodes. Hence, each node tunes its data rate according to a partial knowledge of the network status, initializing new transmissions whenever the urgency of its local information overcomes a certain level. We showed that adjusting communication to the information urgency is essential for improving the tracking accuracy, and our architecture outperforms a standard approach that broadcasts node states periodically. We test the proposed system in an autonomous driving scenario, where vehicles broadcast local measurements to avoid car collisions, and in a drone swarm system,

where UAVs explore a 3-dimensional environment. In the latter case, we also developed a novel mobility model for tracking aerial devices and analyzed different transmission configurations based on the LoRaWAN technology. In doing so, we verified that too advanced tracking frameworks are inconvenient when operating in a system with limited bandwidth, leading to degraded performance despite the higher accuracy of the model. Therefore, we should not solely adapt communication to the application but also perform the inverse operation, tuning the tracking parameters depending on the channel conditions.

The second part of this thesis takes a further step toward managing dynamic networks, analyzing how to exploit multi-agent Machine Learning (ML) for controlling the single network components. In this case, the goal was not to barely estimate the network status but to coordinate the different nodes, allowing them to accomplish missions such as area surveillance, emergency intervention, data muling, and many others. In our vision, each node acts as an autonomous unit that takes decisions in real-time, without central coordination but with only a partial vision of the overall network status. We faced this problem by adopting a Multi-Agent Reinforcement Learning (MARL) approach, associating each node with a learning agent that gains more rewards as it gets closer to the mission achievement. In doing so, we highlighted that the optimal control policy unavoidably depends on the communication process by which local agents acquire new information about the system. At first, we considered the ideal scenario where all the units share a unique perception of the environment, limiting the coordination problem to the estimation of other agents' actions. Hence, we designed a new ML framework that jointly addresses communication and control in networks, proving that adapting the information dissemination to the control task leads to impressive benefits in cyber-physical systems. We can naturally integrate these results with the framework presented in the first part of the thesis, taking advantage of the tracking algorithms we developed to improve the perception of environment conditions.

The last part of this thesis discussed how to manage the different types of resources that characterize the network scenario depicted in the first two parts. In this regard, we emphasized the benefits of the Network Slicing (NS) paradigm, which enables the softwarization of network resources and the definition of multiple logical networks over the same physical infrastructure. Hence, we designed a MARL architecture that assigns each network component to an intelligent unit encharged for allocating the local network resources. In our system, each agent must learn how to coordinate with the others, prioritize the traffic of critical services, e.g., remote control or emergency communication, and address specific Quality of Service (QoS) requirements. The distribution of the control tasks made it possible to increase the overall flexibility since the learned strategy can adapt to different network topologies without additional training. The proposed architecture naturally integrates with the framework outlined in the rest of the thesis, allowing us to optimize resources in real time depending on the requirements of tracking and control functions. At the same time, using intelligent units to manage dynamic networks does not come for free since part of the available resources must be devoted to training the learning algorithms. In the last chapter of the thesis, we designed a model for such a problem, named the cost of learning, and discussed some strategies for balancing resources among network users

and learning agents. Our results emphasize that speeding up the algorithm convergence is not always convenient since it increases the system flexibility but degrades performance on a short time horizon.

We expect multi-agent control systems, such as the one presented in this thesis, to provide future telecommunication networks with a flexibility never reached, supporting scientific progress toward a new kind of technology that continuously adapts to human needs. At the same time, our results showed that constant retraining of the agents is paramount for refining network functions and avoiding sticking to specific use cases. The scientific literature provides many instruments toward this goal, including the Transfer Learning (TL) and Continual Learning (CL) paradigms, which open up new optimization approaches based on adaptability and not strictly on efficiency. However, there is still a lack of thoughtful investigation of how to correctly balance the network resources among end users while gathering the experience necessary for training ML algorithms. The cost of the learning problem presented in the last chapter goes beyond telecommunications since it affects all types of resources, from computational power to communication bandwidth, used for supporting artificial intelligence. In future work, we want to pursue this path, studying new models for evaluating resource-constrained scenarios orchestrated by intelligent entities and proposing innovative solutions to manage the interactions between learning algorithms and end users. In doing so, we want to provide a new approach to designing engineering systems, focusing on the importance of technology innovation alongside immediate performance.

A

Unscented Kalman Filtering

The Kalman Filter (KF) is a well-known recursive algorithm to estimate an unknown variable from a series of consecutive noisy measurements. In particular, the KF exploits a data fusion process, providing an output that is more accurate than any estimate obtained with a single measurement alone. The algorithm assumes that the target variable, named *state*, evolves according to a function $f(\cdot)$ while the measurements are provided by a function $m(\cdot)$. Given the system state $\mathbf{x}(t)$ at time t, the future state $\mathbf{x}(t+T)$ and the state observation $\mathbf{o}(t)$ are given by the equations:

$$\begin{cases} \mathbf{x}(t+T) = f(\mathbf{x}(t)) + \zeta(t), \\ \mathbf{o}(t) = m(\mathbf{x}(t)) + \upsilon(t). \end{cases} \tag{A.1}$$

In particular, $\zeta(t)$ and $\upsilon(t)$ represent the process and measurement noises at time t and are usually modeled as independent Gaussian processes with zero mean and covariance matrices Q and R, respectively. In order to compute the state estimation $\hat{\mathbf{x}}$, it is necessary to define all the parameters in (A.1), including the functions $f(\cdot)$ and $m(\cdot)$. Hence, the KF algorithm is organized in two iterative phases: the *predictive step* uses the past information to compute the *a priori* state estimation $\hat{\mathbf{x}}(t+T|t)$; the *update step* integrates the new observation $\mathbf{o}(t+T)$ with $\hat{\mathbf{x}}(t+T|t)$, providing the *a posteriori* state estimation $\hat{\mathbf{x}}(t+T)$. This procedure does not solely provide the best state estimate but the estimation accuracy also, which is represented by the covariance matrix $\mathbf{P}_x$.

The original KF designed in [22] only enables us to consider linear equations for describing the system state evolution and measurement. In this thesis, we exploit more advanced functions for $f(\cdot)$, as the Constant Turn Rate and Acceleration (CTRA) and 3-Dimensional CTRA (3D-CTRA) models presented in Chapters 2 and 3, respectively. To overcome such a problem, we implement the Unscented Kalman Filter (UKF), an extended version of the classical KF that allows us to handle non-linear equation systems [23]. In this case, the state information is

summarized by a set of particles, named *sigma points*, which go through functions $f(\cdot)$ and $h(\cdot)$ at each step of the algorithm. Each particle is associated with a weight that decreases proportionally to its mathematical distance from the state, thus representing the estimation uncertainty. After the predictive and update steps, the best state estimate is computed as a weighted average of the sigma points, according to the so-called Unscented Transformation (UT). In the literature, there are different techniques for computing the sigma points and the related weights: in this thesis, we consider the UT proposed in [225].

In the following, we describe the operations involved by the UKF algorithm, assuming that the process and measurement noises are purely additive*. Let us call $\hat{\mathbf{x}}(t)$ and $\mathbf{P}_x(t) = E[(\mathbf{x}(t) - \hat{\mathbf{x}}(t))(\mathbf{x}(t) - \hat{\mathbf{x}}(t))^T]$ the system state estimation and the related covariance matrix at time t, respectively. First, the state is encoded into $2L+1$ sigma points $\chi_0(t), \chi_1(t), ..., \chi_{2L}(t)$, which are computed according to the following UT:

$$\chi_0(t) = \hat{\mathbf{x}}(t); \tag{A.2}$$

$$\chi_i(t) = \hat{\mathbf{x}}(t) + \left(\sqrt{(L+\lambda)\mathbf{P}_x(t)}\right)_i, \qquad i = 1, ..., L; \tag{A.3}$$

$$\chi_i(t) = \hat{\mathbf{x}}(t) + \left(\sqrt{(L+\lambda)\mathbf{P}_x(t)}\right)_{i-L}, \qquad i = L+1, ..., 2L. \tag{A.4}$$

In the above equations, L is the dimension of $\mathbf{x}$, $\left(\sqrt{(L+\lambda)\mathbf{P}_x(t)}\right)_i$ indicates the i-th row of the square root of $(L+\lambda)\mathbf{P}_x(t)$, λ is a scaling parameter, and $\alpha \in \mathbb{R}$ represents the sigma points spread around $\chi_0(t)$. The value of λ is computed as $\alpha^2(L+k) - L$, where k is another tunable parameter, customarily set to 0. Each of the sigma points is then associated to a set of scalar weights w_i^m and w_i^c, whose values are:

$$w_0^m = \lambda/(L+\lambda); \tag{A.5}$$

$$w_0^c = \lambda/(L+\lambda) + 1 - \alpha^2 + \beta; \tag{A.6}$$

$$w_i^m = w_i^c = \lambda/\left(2(L+\lambda)\right), \; i = 1, ..., 2L. \tag{A.7}$$

In the above equations, β is a constant value tuned according to the system state distribution. In the conventional case where $\mathbf{x}$ is Gaussian, the optimal value for β is 2.

When the UKF algorithm performs the *predictive* step, each sigma point is propagated through the (possibly non-linear) function $f(\cdot)$, given in the first equation of (A.1), thus generating a new set of sigma points: $f\left(\chi_i(t)\right), \; i = 0, 1, ..., 2L$. The new sigma points are used to compute the *a priori* state estimation $\hat{\mathbf{x}}(t+T|t)$ and the related covariance matrix $\mathbf{P}_x(t+T|t)$:

$$\hat{\mathbf{x}}(t+T|t) = \sum_{i=0}^{2L} w_i^m f\left(\chi_i(t)\right); \tag{A.8}$$

*The UKF formulation for the case in which $\zeta(t)$ and $\upsilon(t)$ are not additive is more complex and out of the scope of this thesis; for a deeper insight into the UKF algorithm, see [23].

$$\mathbf{P}_x(t+T|t) = Q + \sum_{i=0}^{2L} w_i^c (f(\chi_i(t)) - \hat{\mathbf{x}}(t+T|t)) \times (f(\chi_i(t)) - \hat{\mathbf{x}}(t+T|t))^T, \tag{A.9}$$

where Q is the process noise covariance.

At the beginning of the *update* step, the original sigma points are propagated through the measurement function $h(\cdot)$ given in the second equation of (A.1). The new points $h(\chi_i(t))$, $i = 0, 1, ..., 2L$, are used to predict the future observation $\hat{\mathbf{o}}(t+T|t)$ and the related covariance matrix $\mathbf{P}_o(t+T|t)$:

$$\hat{\mathbf{o}}(t+T|t) = \sum_{i=0}^{2L} w_i^m h(\chi_i(t)); \tag{A.10}$$

$$\mathbf{P}_o(t+T|t) = R + \sum_{i=0}^{2L} w_i^c (h(\chi_i(t)) - \hat{\mathbf{o}}(t+T|t)) \times (h(\chi_i(t)) - \hat{\mathbf{o}}(t+T|t))^T; \tag{A.11}$$

where R is the measurement noise covariance. The optimal *Kalman gain* is then computed as $K = \mathbf{P}_{xo}(\mathbf{P}_{xo})^{-1}$, where $(\cdot)^{-1}$ indicates the inversion operation and $\mathbf{P}_{xo}$ is the residual covariance matrix:

$$\mathbf{P}_{xo} = \sum_{i=0}^{2L} w_i^c (\chi_i(t+T|t) - \hat{\mathbf{x}}(t+T|t)) \times (\xi_i(t+T|t) - \hat{\mathbf{o}}(t+T|t))^T. \tag{A.12}$$

Finally, given the new observation $\mathbf{o}(t+T)$, the *a posteriori* state estimation $\hat{\mathbf{x}}(t+T)$ and the related covariance matrix $\mathbf{P}_x(t+T)$ are obtained as

$$\hat{\mathbf{x}}(t+T) = \hat{\mathbf{x}}(t+T|t) + K(\mathbf{o}(t+T) - \hat{\mathbf{o}}(t+T|t)); \tag{A.13}$$

$$\mathbf{P}_x(t+T) = \mathbf{P}_x(t+T|t) - K\mathbf{P}_o(t+T|t)K^T. \tag{A.14}$$

Hence, as new observations become available, the algorithm repeats the above steps iteratively.

B

Reinforcement Learning

The Reinforcement Learning (RL) paradigm is considered one of the main branches of Machine Learning (ML), together with the *supervised* and *unsupervised* frameworks. Differently from other approaches, RL does not solely aim to solve specific classification or regression problems, but its goal is to estimate the optimal decision-making strategy to maximize the long-term performance of a target system [110]. In particular, RL algorithms do not need labeled data to carry out the training phase but associate each possible agent action with a quality value.

In a RL scenario, the target system is modeled as a Markov Decision Process (MDP), which is a powerful mathematical tool used to represent decision-making problems [226]. In particular, a MDP is described by a tuple $(\mathcal{S}, \mathcal{A}, P(\cdot), r(\cdot))$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P : \mathcal{S}^2 \times \mathcal{A} \to [0, 1]$ is the state transition probability function and $r : \mathcal{S}^2 \times \mathcal{A} \to \mathbb{R}$ is the reward function. In particular, $P(s, s', a)$ is the probability that action a in state s will lead to state s', while $r(s, s', a)$ is the immediate reward received after a transition from s to s' due to action a. The time is discretized into slots $t \in \mathbb{Z}^+$ so that, at any slot, the agent observes the system state $s(t) \in \mathcal{S}$, performs an action $a(t) \in \mathcal{A}$ and receives a reward $r(t) \in \mathbb{R}$. Under these assumptions, the future state $s(t + 1)$ depends uniquely on the previous state $s(t)$ and the last agent action $a(t)$.

We consider that the agent chooses new action according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(s, a)$ is the probability to take the action a from state s. The aim of any RL algorithm is to determine a policy that maximizes the *discounted return*, defined as

$$G(t) = \sum_{\tau=0}^{\infty} \lambda^{\tau} r(t + \tau), \tag{B.1}$$

where $\lambda \in (0, 1)$ is the so-called *discount factor*. Particularly, if $\lambda \to 0$, the algorithm will behave

in a myopic manner, taking actions that can acquire high reward in the short term; instead, if $\lambda \to 1$, the algorithm will prioritize the actions that bring more benefits in the long term. If the environment is partially observable, the agent perceives the state through a function $P_o : \mathcal{S} \to \mathcal{O}$, associating each state $s \in \mathcal{S}$ with an observation $o \in \mathcal{O}$. In this case, the domain of the policy function is then given by the observation space $\mathcal{O}$.

To maximize the expected $G(t)$, we need to estimate the quality of each possible state-action pair, associating each state with the optimal action. Given a policy π, an RL algorithm associates each state $s \in \mathcal{S}$ with a *state-value* $V_\pi(s)$ and each state-action pair (s, a) with a *Q-value* $Q_\pi(s, a)$. Hence, $V_\pi(s)$ represents the expected discounted return achieved following the strategy π from state s:

$$V_\pi(s) = E\left[G(t)|s(t) = s, \pi\right]. \tag{B.2}$$

Instead, $Q_\pi(s, a)$ represents the expected discounted return obtained by choosing action a in state s and following the policy π in the future:

$$Q_\pi(s, a) = E[G(t)|s(t) = s, a(t) = a, \pi], \tag{B.3}$$

In particular, the *Bellman optimality equation* ensures that the optimal policy π^* provides the maximum value of both $V_\pi(s)$ and $Q_\pi(s, a)$, $\forall\ s \in \mathcal{S}$, $a \in \mathcal{A}$:

$$V_{\pi^*}(s) = \max_\pi V_\pi(s), \quad \forall\ s \in \mathcal{S}. \tag{B.4}$$

$$Q_{\pi^*}(s, a) = \max_\pi Q_\pi(s, a), \quad \forall\ (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{B.5}$$

An explicit solution of the Bellman equation is only possible in straightforward problems, while realistic scenarios require adopting more complex techniques. One of the most popular classes of RL methods is temporal difference learning, which enables learning the optimal policy according to a *bootstrapping* approach. In this case, the algorithm updates the values of states and actions according to both the experience samples and the current estimate of $V_\pi(\cdot)$ and $Q_\pi(\cdot)$. Two examples of temporal difference methods are the well-known State-Action-Reward-State-Action (SARSA) and *Q-Learning* algorithms [227]. The first is an *on-policy* technique that updates $Q(\cdot)$ as

$$Q_{\text{new}}(s(t), a(t)) = (1 - \alpha)Q(s(t), a(t)) + \alpha\left(r(t) + \lambda Q(s(t + 1), a(t + 1))\right). \tag{B.6}$$

Instead, Q-learning follows an *off-policy* approach, and updates $Q(s(t), a(t))$ as

$$Q_{\text{new}}(s(t), a(t)) = (1 - \alpha)Q(s(t), a(t)) + \alpha\left(r(t) + \lambda \max_a Q(s(t + 1), a)\right). \tag{B.7}$$

In the above equations, the parameter $\alpha \in (0, 1)$ is called learning rate and determines the convergence speed of the algorithm. Therefore, SARSA updates $Q(s(t), a(t))$ using $Q(s(t + 1), a(t + 1))$, i.e., the estimated Q-value associated with the state-action pair seen at slot $t + 1$. Instead, Q-learning performs the same operation considering the maximum Q-value associated

with the state seen at slot $t+1$. It was proven that both algorithms converge to the optimal policy if every observation-action pair is visited infinitely often.

Another well known RL algorithms is *Double Q-Learning*, which is an extension of (B.7) with faster convergence [130]. In this case, we define two functions $Q^{\text{primary}}(\cdot)$ and $Q^{\text{target}}(\cdot)$, named primary and target Q-value functions, respectively. While the agent exploits the outputs of $Q^{\text{primary}}(\cdot)$ to take new actions in the environment, $Q^{\text{target}}(\cdot)$ is used to estimate future Q-values. Hence, $Q^{\text{primary}}(\cdot)$ is updated as:

$$\begin{aligned} Q^{\text{primary}}_{\text{new}}(s(t),a(t)) =&(1-\alpha)Q^{\text{primary}}(s(t),a(t)) \\ &+\alpha\left(r(t)+\lambda Q^{\text{primary}}(s(t+1),\underset{a\in\mathcal{A}}{\arg\max}\, Q^{\text{target}}(s(t+1),a))\right). \end{aligned} \tag{B.8}$$

As the training goes on, the $Q^{\text{target}}(\cdot)$ architecture is periodically replaced with that of $Q^{\text{primary}}(\cdot)$. This scheme reduces the probability that the outputs of the Q-value function diverge, making the learning process more stable.

If the state and action spaces get too complex, conventional RL approaches may fail to determine the optimal policy because of the *curse of dimensionality* [228]. To address such a problem, the Deep Reinforcement Learning (DRL) paradigm makes use of deep Neural Networks (NNs) to estimate the value functions $V(\cdot)$ and $Q(\cdot)$. In particular, DRL algorithms are capable of handling continuous state and action spaces, which means that $|\mathcal{S}\times\mathcal{A}|\to\infty$.

References

[1] F. Mason, M. Giordani, F. Chiariotti, A. Zanella, and M. Zorzi, "An adaptive broadcasting strategy for efficient dynamic mapping in vehicular networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5605–5620, Aug. 2020.

[2] H. Hartenstein and L. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Communications Magazine*, vol. 46, no. 6, pp. 164–171, Jun. 2008.

[3] E. Hossain, G. Chow, V. C. Leung, R. D. McLeod, J. Mišić, V. W. Wong, and O. Yang, "Vehicular telematics over heterogeneous wireless networks: A survey," *Computer Communications*, vol. 33, no. 7, pp. 775–793, May 2010.

[4] M. Amadeo, C. Campolo, and A. Molinaro, "Enhancing IEEE 802.11 p/WAVE to provide infotainment applications in VANETs," *Ad Hoc Networks*, vol. 10, no. 2, pp. 253–269, Sep. 2012.

[5] 3GPP, "Service requirements for enhanced V2X scenarios," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.186, 2017.

[6] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark, "Connected Vehicles: Solutions and Challenges," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 289–299, Aug. 2014.

[7] National Highway Traffic Safety Administration (NHTSA), "Federal motor vehicle safety standards; V2V communications," *Federal Register*, vol. 82, no. 8, pp. 3854–4019, Jan. 2017.

[8] N. Alam and A. G. Dempster, "Cooperative Positioning for Vehicular Networks: Facts and Future," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 4, pp. 1708–1717, Dec. 2013.

[9] L. N. Balico, A. A. F. Loureiro, E. F. Nakamura, R. S. Barreto, R. W. Pazzi, and H. A. B. F. Oliveira, "Localization Prediction in Vehicular Ad Hoc Networks," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 2784–2803, May 2018.

[10] S. Yousefi and M. S. Mousavi and M. Fathy, "Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives," in *6th International Conference on ITS Telecommunications*, Jun. 2006, pp. 761–766.

[11] C. Joo and A. Eryilmaz, "Wireless scheduling for information freshness and synchrony: Drift-based design and heavy-traffic analysis," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2556–2568, Dec. 2018.

[12] L. Wischhof and H. Rohling, "Congestion control in vehicular ad hoc networks," in *International Conference on Vehicular Electronics and Safety*, Oct. 2005, pp. 58–63.

[13] G. Caizzone, P. Giacomazzi, L. Musumeci, and G. Verticale, "A power control algorithm with high channel availability for vehicular ad hoc networks," in *IEEE International Conference on Communications (ICC)*, May 2005, pp. 3171–3176.

[14] C. L. Huang, Y. P. Fallah, and R. Sengupta, "Analysis of aggregated power level and rate-power control designs for status update messages in VANETs," in *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Oct. 2009, pp. 615–620.

[15] C. Bisdikian, L. M. Kaplan, and M. B. Srivastava, "On the quality and value of information in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 9, no. 4, pp. 1–26, Jul. 2013.

[16] M. Giordani, A. Zanella, T. Higuchi, O. Altintas, and M. Zorzi, "Investigating Value of Information in Future Vehicular Communications," in *2nd IEEE Connected and Automated Vehicles Symposium (CAVS)*, Sep. 2019, pp. 1–5.

[17] M. Giordani, T. Higuchi, A. Zanella, O. Altintas, and M. Zorzi, "A framework to assess value of information in future vehicular networks," in *1st ACM MobiHoc Workshop on Technologies, mOdels, and Protocols for Cooperative Connected Cars*, Jul. 2019, pp. 31–36.

[18] T. Higuchi, M. Giordani, A. Zanella, M. Zorzi, and O. Altintas, "Value-anticipating V2V communications for cooperative perception," in *IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2019, pp. 1947–1952.

[19] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO-Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3, pp. 128–138, Dec. 2012.

[20] A. Boukerche, H. A. Oliveira, E. F. Nakamura, and A. A. Loureiro, "Vehicular ad hoc networks: A new challenge for localization-based systems," *Computer Communications*, vol. 31, no. 12, pp. 2838–2849, Jul. 2008.

[21] H. S. Ramos, A. Boukerche, R. W. Pazzi, A. C. Frery, and A. A. F. Loureiro, "Cooperative target tracking in vehicular sensor networks," *IEEE Wireless Communications*, vol. 19, no. 5, pp. 66–73, Oct. 2012.

[22] R. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of basic Engineering*, vol. 82, pp. 35–45, Jan. 1960.

[23] E. A. Wan and R. V. D. Merwe, "The unscented Kalman filter for nonlinear estimation," in *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, Oct. 2000, pp. 153–158.

[24] P. Del Moral, "Nonlinear filtering: Interacting particle resolution," *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, vol. 325, no. 6, pp. 653–658, Sep. 1997.

[25] P. Lytrivis, G. Thomaidis, M. Tsogas, and A. Amditis, "An advanced cooperative path prediction algorithm for safety applications in vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 669–679, Sep. 2011.

[26] A. U. Peker, O. Tosun, and T. Acarman, "Particle filter vehicle localization and map-matching using map topology," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 248–253.

[27] A. T. Akabane, R. W. Pazzi, E. R. M. Madeira, and L. A. Villas, "Modeling and Prediction of Vehicle Routes Based on Hidden Markov Model," in *IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Sep. 2017, pp. 1–5.

[28] A. B. Poritz, "Hidden markov models: a guided tour," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Apr. 1988, pp. 7–13.

[29] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[30] M. Roth, G. Hendeby, and F. Gustafsson, "EKF/UKF maneuvering target tracking using coordinated turn models with polar/Cartesian velocity," in *17th International Conference on Information Fusion (FUSION)*, Jul. 2014, pp. 1–8.

[31] C. M. Kang, S. J. Jeon, S. Lee, and C. C. Chung, "Parametric trajectory prediction of surrounding vehicles," in *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Jun. 2017, pp. 26–31.

[32] T. King, H. Füßler, M. Transier, and W. Effelsberg, "Dead-reckoning for position-based forwarding on highways," in *International Workshop on Intelligent Transportation (WIT)*, Mar. 2006, pp. 199–204.

[33] D. Wang, J. Liao, Z. Xiao, X. Li, and V. Havyarimana, "Online-SVR for vehicular position prediction during GPS outages using low-cost INS," in *IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Aug. 2015, pp. 1945–1950.

[34] J. Park, D. Li, Y. L. Murphey, J. Kristinsson, R. McGee, M. Kuang, and T. Phillips, "Real time vehicle speed prediction using a neural network traffic model," in *The International Joint Conference on Neural Networks*, Jul. 2011, pp. 2991–2996.

[35] N. Deo, A. Rangesh, and M. M. Trivedi, "How Would Surround Vehicles Move? A Unified Framework for Maneuver Classification and Motion Prediction," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 2, pp. 129–140, Jun. 2018.

[36] C. Hermes, C. Wohler, K. Schenk, and F. Kummert, "Long-term vehicle motion prediction," in *IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2009, pp. 652–657.

[37] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 4363–4369.

[38] Y. J. Li, "An overview of the DSRC/WAVE technology," in *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Nov. 2010, pp. 544–558.

[39] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an international standard for wireless access in vehicular environments," in *Vehicular Technology Conference (VTC-Spring)*, May 2008, pp. 2036–2040.

[40] G. Bansal, J. B. Kenney, and C. E. Rohrs, "LIMERIC: A Linear Adaptive Message Rate Algorithm for DSRC Congestion Control," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 9, pp. 4182–4197, Nov. 2013.

[41] H. A. Omar, W. Zhuang, and L. Li, "VeMAC: A TDMA-Based MAC Protocol for Reliable Broadcast in VANETs," *IEEE Transactions on Mobile Computing*, vol. 12, no. 9, pp. 1724–1736, Sep. 2013.

[42] N. Taherkhani and S. Pierre, "Centralized and localized data congestion control strategy for vehicular ad hoc networks using a machine learning clustering algorithm," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3275–3285, Nov. 2016.

[43] S. Rezaei, R. Sengupta, H. Krishnan, X. Guan, and R. Bhatia, "Tracking the position of neighboring vehicles using wireless communications," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 3, pp. 335–350, Jun. 2010.

[44] C. Huang, Y. P. Fallah, R. Sengupta, and H. Krishnan, "Adaptive intervehicle communication control for cooperative safety systems," *IEEE Network*, vol. 24, no. 1, pp. 6–13, Jan. 2010.

[45] Y. P. Fallah, C. Huang, R. Sengupta, and H. Krishnan, "Analysis of information dissemination in vehicular ad-hoc networks with application to cooperative vehicle safety systems," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 1, pp. 233–247, Jan. 2011.

[46] F. Richards, "A flexible growth function for empirical use," *Journal of experimental Botany*, vol. 10, no. 2, pp. 290–301, Jun. 1959.

[47] M. Tsogas, A. Polychronopoulos, and A. Amditis, "Unscented Kalman filter design for curvilinear motion models suitable for automotive safety applications," in *7th International Conference on Information Fusion*, Jul. 2005, pp. 1295–1302.

[48] B. Cheng, A. Rostami, M. Gruteser, J. B. Kenney, G. Bansal, and K. Sjoberg, "Performance evaluation of a mixed vehicular network with CAM-DCC and LIMERIC vehicles," in *IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Jun. 2015, pp. 1–6.

[49] J. Benin, M. Nowatkowski, and H. Owen, "Vehicular network simulation propagation loss model parameter standardization in ns-3 and beyond," in *IEEE Southeastcon*, Mar. 2012, pp. 1–5.

[50] F. Abrate, A. Vesco, and R. Scopigno, "An analytical packet error rate model for WAVE receivers," in *IEEE Vehicular Technology Conference (VTC Fall)*, Sep. 2011, pp. 1–5.

[51] M. Boban, X. Gong, and W. Xu, "Modeling the evolution of line-of-sight blockage for v2v channels," in *IEEE 84th Vehicular Technology Conference (VTC-Fall)*, Sep. 2016, pp. 1–7.

[52] B. Kim and K. Yi, "Probabilistic and holistic prediction of vehicle states using sensor fusion for application to integrated vehicle safety systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2178–2190, Oct. 2014.

[53] T. Driver, "Long-Term Prediction of GPS Accuracy: Understanding the Fundamentals," *20th International Technical Meeting of the Satellite Division of The Institute of Navigation*, vol. 2007, pp. 152–163, Sep. 2007.

[54] G. Falco, M. Pini, and G. Marucco, "Loose and tight GNSS/INS integrations: Comparison of performance assessed in real urban scenarios," *Sensors*, vol. 17, no. 2, pp. 255–280, Jan. 2017.

[55] 3GPP, "Study on NR positioning support," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.855, 2018.

[56] F. Mason, M. Capuzzo, D. Magrin, F. Chiariotti, A. Zanella, and M. Zorzi, "Remote tracking of UAV swarms via 3D mobility models and LoRaWAN communications," *IEEE Transactions on Wireless Communications*, vol. 21, no. 5, pp. 2953–2968, May 2022.

[57] U. F. A. Administration, "FAA Aerospace Forecast, Fiscal Years 2019-2039," 2019. [Online]. Available: https://www.faa.gov/news/updates/?newsId=93646

[58] A. Fotouhi, H. Qiang, M. Ding, M. Hassan, L. G. Giordano, A. Garcia-Rodriguez, and J. Yuan, "Survey on UAV cellular communications: Practical aspects, standardization advancements, regulation, and security challenges," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3417–3442, Mar. 2019.

[59] S. Sekander, H. Tabassum, and E. Hossain, "Multi-tier drone architecture for 5G/B5G cellular networks: Challenges, trends, and prospects," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 96–103, Mar. 2018.

[60] H. Huang and A. V. Savkin, "Towards the Internet of Flying Robots: A survey," *MDPI Sensors*, vol. 18, no. 11, pp. 4038–4058, Nov. 2018.

[61] T. Long, M. Ozger, O. Cetinkaya, and O. B. Akan, "Energy neutral Internet of Drones," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 22–28, Jan. 2018.

[62] B. Hament and P. Oh, "Unmanned aerial and ground vehicle (UAV-UGV) system prototype for civil infrastructure missions," in *IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2018, pp. 1–4.

[63] V. Sharma, I. You, G. Pau, M. Collotta, J. Lim, and J. Kim, "LoRaWAN-based energy-efficient surveillance by drones for intelligent transportation systems," *MDPI Energies*, vol. 11, no. 3, p. 573, Mar. 2018.

[64] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: An IoT platform for data-driven agriculture," in *14th Symposium on Networked Systems Design and Implementation (NSDI)*, Sep. 2017, pp. 515–529.

[65] B. Arbanas, A. Ivanovic, M. Car, M. Orsag, T. Petrovic, and S. Bogdan, "Decentralized planning and control for UAV–UGV cooperative teams," *Autonomous Robots*, vol. 42, no. 8, pp. 1601–1618, Dec. 2018.

[66] A. Ismail, B. Bagula, and E. Tuyishimire, "Internet-of-things in motion: A UAV coalition model for remote sensing in smart cities," *Sensors*, vol. 18, no. 7, pp. 2184–2209, Jul. 2018.

[67] S. Zakaria, M. R. Mahadi, A. F. Abdullah, and K. Abdan, "Aerial platform reliability for flood monitoring under various weather conditions: A review," in *GeoInformation for Disaster Management Conference*, Mar. 2018, pp. 295–314.

[68] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner, "An autonomous multi-UAV system for search and rescue," in *1st Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet)*, May 2015, pp. 33–38.

[69] J. Chen, T. Ma, and C. Su, "The study of UAV intelligent support mode based on battlefield networks," in *2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Mar. 2017, pp. 1912–1915.

[70] C. V. Angelino, V. R. Baraniello, and L. Cicala, "UAV position and attitude estimation using IMU, GNSS and camera," in *15th International Conference on Information Fusion*, Jul. 2012, pp. 735–742.

[71] J. Dentler, S. Kannan, M. A. O. Mendez, and H. Voos, "A real-time model predictive position control with collision avoidance for commercial low-cost quadrotors," in *Conference on Control Applications (CCA)*, Sep. 2016, pp. 519–525.

[72] J. C. Hodgson, S. M. Baylis, R. Mott, A. Herrod, and R. H. Clarke, "Precision wildlife monitoring using unmanned aerial vehicles," *Scientific Reports*, vol. 6, no. 1, pp. 1–7, Mar. 2016.

[73] O. Alvear, C. T. Calafate, N. R. Zema, E. Natalizio, E. Hernández-Orallo, J. C. Cano, and P. Manzoni, "A discretized approach to air pollution monitoring using UAV-based sensing," *Mobile Networks and Applications*, vol. 23, no. 6, pp. 1693–1702, Dec. 2018.

[74] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis, "A review on UAV-based applications for precision agriculture," *MDPI Information*, vol. 10, no. 11, p. 349, Nov. 2019.

[75] J. Langhammer, J. Bernsteinová, and J. Miřijovskỳ, "Building a high-precision 2D hydrodynamic flood model using UAV photogrammetry and sensor network monitoring," *MDPI Water*, vol. 9, no. 11, p. 861, Nov. 2017.

[76] B. D. Anderson, B. Fidan, C. Yu, and D. Walle, "UAV formation control: Theory and application," in *Recent advances in learning and control*. Springer, Jan. 2008, pp. 15–33.

[77] J. Haxhibeqiri, F. Van den Abeele, I. Moerman, and J. Hoebeke, "LoRa scalability: A simulation model based on interference measurements," *MDPI Sensors*, vol. 17, no. 6, p. 1193, May 2017.

[78] J.-D. M. M. Biomo, T. Kunz, and M. St-Hilaire, "An enhanced Gauss-Markov mobility model for simulations of unmanned aerial ad hoc networks," in *7th IFIP Wireless and Mobile Networking Conference (WMNC)*, May 2014, pp. 1–8.

[79] O. Bouachir, A. Abrassart, F. Garcia, and N. Larrieu, "A mobility model for UAV ad hoc network," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 383–388.

[80] Y. Wan, K. Namuduri, Y. Zhou, and S. Fu, "A smooth-turn mobility model for airborne networks," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 7, pp. 3359–3370, Mar. 2013.

[81] J. Tiemann, F. Schweikowski, and C. Wietfeld, "Design of an UWB indoor-positioning system for UAV navigation in GNSS-denied environments," in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Oct. 2015, pp. 1–7.

[82] R. Opromolla, G. Inchingolo, and G. Fasano, "Airborne visual detection and tracking of cooperative UAVs exploiting deep learning," *MDPI Sensors*, vol. 19, no. 19, pp. 4332–4359, Jan. 2019.

[83] D. Fan, F. Gao, B. Ai, G. Wang, Z. Zhong, Y. Deng, and A. Nallanathan, "Channel estimation and self-positioning for UAV swarm," *IEEE Transactions on Communications*, vol. 67, no. 11, pp. 7994–8007, Aug. 2019.

[84] R. Mendrzik, D. Cabric, and G. Bauch, "Error bounds for terahertz MIMO positioning of swarm UAVs for distributed sensing," in *International Conference on Communications Workshops (ICC Workshops)*, May 2018, pp. 1–6.

[85] L. Vangelista, A. Zanella, and M. Zorzi, "Long-range IoT technologies: The dawn of LoRa™," in *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*, Sep. 2015, pp. 51–58.

[86] F. Cuomo, M. Campo, A. Caponi, G. Bianchi, G. Rossini, and P. Pisani, "EXPLoRa: Extending the Performance of LoRa by suitable spreading factor allocations," in *13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2017, pp. 1–8.

[87] N. Benkahla, H. Tounsi, S. Ye-Qiong, and M. Frikha, "Enhanced ADR for LoRaWAN networks with mobility," in *15th International Wireless Communications and Mobile Computing Conference (IWCMC)*, Jun. 2019, pp. 1–6.

[88] K. Kousias, G. Caso, Ö. Alay, and F. Lemic, "Empirical analysis of LoRaWAN adaptive data rate for mobile Internet of Things applications," in *MobiCom Workshop on Wireless of the Students, by the Students, and for the Students (S3)*, Nov. 2019, pp. 9–11.

[89] A. Farhad, D. H. Kim, S. Subedi, and J. Y. Pyun, "Enhanced LoRaWAN adaptive data rate for mobile Internet of Things devices," *MDPI Sensors*, vol. 20, no. 22, p. 6466, Nov. 2020.

[90] Archimedes of Syracuse, "On Spirals," ∼225BC.

[91] LoRa Alliance, "LoRaWAN™ 1.1 Specification," Oct. 2017.

[92] D. Croce, M. Gucciardo, S. Mangione, G. Santaromita, and I. Tinnirello, "Impact of LoRa imperfect orthogonality: Analysis of link-level performance," *IEEE Communications Letters*, vol. 22, no. 4, pp. 796–799, Apr. 2018.

[93] T. Saadawi and A. Ephremides, "Analysis, stability, and optimization of slotted ALOHA with a finite number of buffered users," *IEEE Transactions on Automatic Control*, vol. 26, no. 3, pp. 680–689, Jun. 1981.

[94] V. Anantharam, "The stability region of the finite-user slotted ALOHA protocol," *IEEE Transactions on information Theory*, vol. 37, no. 3, pp. 535–540, May 1991.

[95] D. Sant, "Throughput of unslotted ALOHA channels with arbitray packet interarrival time distributions," *IEEE Transactions on Communications*, vol. 28, no. 8, pp. 1422–1425, Aug. 1980.

[96] R. D. Yates, Y. Sun, D. Richard Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of Information: An introduction and survey," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1183–1210, 2021.

[97] Semtech Corporation, "Sx1272 datasheet 2.01," 2015.

[98] M. Fonder and M. V. Droogenbroeck, "Mid-air: A multi-modal dataset for extremely low altitude drone flights," in *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, Jun. 2019, pp. 553–562.

[99] N. S. Verification and M. Branch, "Global Positioning System (GPS) Standard Positioning Service (SPS) performance analysis report," William J. Hughes Technical Center, Tech. Rep. 86, 2014.

[100] D. Magrin, M. Capuzzo, and A. Zanella, "A thorough study of LoRaWan performance under different parameter settings," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 116–127, Jan. 2020.

[101] C. Goursaud and J. M. Gorce, "Dedicated Networks for IoT: PHY / MAC State of the Art and Challenges," *EAI Endorsed Transactions on Internet of Things*, vol. 1, no. 1, Oct. 2015.

[102] F. Venturini, F. Mason, F. Pase, F. Chiariotti, A. Testolin, A. Zanella, and M. Zorzi, "Distributed reinforcement learning for flexible and efficient uav swarm control," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 3, pp. 955–969, Sep. 2021.

[103] N. Hossein Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based Internet of Things services: Comprehensive survey and future perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, Sep. 2016.

[104] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned Aerial Vehicles (UAVs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, Sep. 2019.

[105] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3340–3385, 4th quarter 2019.

[106] S. J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 837–855, Aug. 2018.

[107] D. Baldazo, J. Parras, and S. Zazo, "Decentralized multi-agent deep reinforcement learning in swarms of drones for flood monitoring," in *European Signal Processing Conference (EUSIPCO)*. EURASIP, Sep. 2019, pp. 1–5.

[108] D. Albani, D. Nardi, and V. Trianni, "Field coverage and weed mapping by UAV swarms," in *International Conference on Intelligent Robots and Systems*, Sep. 2017, pp. 4319–4325.

[109] K. D. Julian and M. J. Kochenderfer, "Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1768–1778, Aug. 2019.

[110] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998.

[111] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *International work-conference on artificial neural networks*, Jun. 2005, pp. 758–770.

[112] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," in *3rd International Workshop on Conflict Resolution in Decision Making (COREDEMA)*, Jul. 2017, pp. 1–64.

[113] L. Busoniu, R. Babuška, and B. De Schutter, "Multi-agent Reinforcement Learning: An Overview," *Innovations in Multi-Agent Systems and Applications*, vol. 310, pp. 113–147, Nov. 2010.

[114] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, Nov. 2019.

[115] R. Zanol, F. Chiariotti, and A. Zanella, "Drone mapping through multi-agent reinforcement learning," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2019, pp. 1–7.

[116] J. Cui, Y. Liu, and A. Nallanathan, "The application of multi-agent reinforcement learning in UAV networks," in *IEEE International Conference on Communications Workshops (ICC)*, May 2019, pp. 1–6.

[117] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *30th International Conference on Neural Information Processing Systems (NeurIPS)*, Dec. 2016, pp. 2145–2153.

[118] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[119] F. A. Oliehoek, C. Amato *et al.*, *A concise introduction to decentralized POMDPs*. Springer, 2016.

[120] C. Wu, B. Ju, Y. Wu, X. Lin, N. Xiong, G. Xu, H. Li, and X. Liang, "UAV autonomous target search based on deep reinforcement learning in complex disaster scene," *IEEE Access*, vol. 7, pp. 117 227–117 245, 2019.

[121] Y. H. Hsu and R. H. Gau, "Reinforcement learning-based collision avoidance and optimal trajectory planning in UAV communication networks," *IEEE Transactions on Mobile Computing*, pp. 306–320, Jun. 2020.

[122] U. Challita, W. Saad, and C. Bettstetter, "Deep reinforcement learning for interference-aware path planning of cellular-connected UAVs," in *IEEE International Conference on Communications (ICC)*, May 2018.

[123] C. H. Liu, Z. Chen, J. Tang, J. Xu, and C. Piao, "Energy-efficient UAV control for effective and fair communication coverage: A deep reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 2059–2070, Aug. 2018.

[124] X. Liu, Y. Liu, and Y. Chen, "Reinforcement learning in multiple-UAV networks: Deployment and movement design," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8036–8049, Jun. 2019.

[125] J. Hu, H. Zhang, L. Song, Z. Han, and H. V. Poor, "Reinforcement learning for a cellular internet of UAVs: protocol design, trajectory control, and resource management," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 116–123, Feb. 2020.

[126] T. Liu, Z. Zhang, H. Jiang, Y. Qian, K. Liu, J. Dang, and L. Wu, "Measurement-based characterization and modeling for low-altitude UAV air-to-air channels," *IEEE Access*, vol. 7, pp. 98 832–98 840, 2019.

[127] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, "Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs," in *19th AAAI Conference on Artificial Intelligence*, Jul. 2005, pp. 133–139.

[128] A. Kumar, S. Zilberstein, and M. Toussaint, "Scalable multiagent planning using probabilistic inference," in *International Joint Conference on Artificial Intelligence (IJCAI)*, Jul. 2011, pp. 2140–2146.

[129] C. Zhang and V. R. Lesser, "Coordinated multi-agent reinforcement learning in networked distributed POMDPs," in *25th AAAI Conference on Artificial Intelligence*, Aug. 2011, pp. 1–7.

[130] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *30th AAAI Conference on Artificial Intelligence*, Mar. 2016, pp. 2094–2100.

[131] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *International Conference of Learning Representations (ICLR)*, Sep. 2020, pp. 1–13.

[132] K. He and J. Sun, "Convolutional neural networks at constrained time cost," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5353–5360, Jun. 2015.

[133] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018.

[134] J. Cheng, W. Chen, F. Tao, and C. L. Lin, "Industrial IoT in 5G environment towards smart manufacturing," *Journal of Industrial Information Integration*, vol. 10, pp. 10–19, Jun. 2018.

[135] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and Industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, Mar. 2017.

[136] D. Zhang, P. Shi, Q.-G. Wang, and L. Yu, "Analysis and synthesis of networked control systems: A survey of recent advances and challenges," *ISA Transactions*, vol. 66, pp. 376–392, Jan. 2017.

[137] M. Klügel, M. H. Mamduhi, S. Hirche, and W. Kellerer, "AoI-penalty minimization for networked control systems with packet loss," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 189–196.

[138] O. Ayan, M. Vilgelm, M. Klügel, S. Hirche, and W. Kellerer, "Age-of-information vs. value-of-information scheduling for cellular networked control systems," in *IEEE 10th International Conference on Cyber-Physical Systems (CPS/IoT)*, Apr. 2019, pp. 109–117.

[139] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[140] D. Ritchie, "Shannon and Weaver: Unravelling the paradox of information," *Communication research*, vol. 13, no. 2, pp. 278–298, Apr. 1986.

[141] T. Y. Tung, S. Kobus, J. P. Roig, and D. Gündüz, "Effective communications: A joint learning and communication framework for multi-agent reinforcement learning over noisy channels," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2590–2603, Jun. 2021.

[142] A. Lazaridou and M. Baroni, "Emergent multi-agent communication in the deep learning era," *arXiv preprint*, pp. 1–24, Jun. 2020.

[143] M. Y. I. Zia, J. Poncela, and P. Otero, "State-of-the-art underwater acoustic communication modems: Classifications, analyses and design challenges," *Wireless Personal Communications*, vol. 116, no. 2, pp. 1325–1360, Jan. 2021.

[144] Q. Fu, A. Song, F. Zhang, and M. Pan, "Reinforcement learning-based trajectory optimization for data muling with underwater mobile nodes," *IEEE Access*, vol. 10, pp. 38 774–38 784, Apr. 2022.

[145] A. Kosta, N. Pappas, and V. Angelakis, "Age of information: A new concept, metric, and tool," *Foundations and Trends in Networking*, vol. 12, no. 3, pp. 162–259, Nov. 2017.

[146] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *International Conference on Computer Communications (INFOCOM)*, Mar. 2012.

[147] A. Maatouk, S. Kriouile, M. Assaad, and A. Ephremides, "The Age of Incorrect Information: A new performance metric for status updates," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2215–2228, Jul. 2020.

[148] F. Chiariotti, A. A. Deshpande, M. Giordani, K. Antonakoglou, T. Mahmoodi, and A. Zanella, "QUIC-EST: A QUIC-enabled scheduling and transmission scheme to maximize VoI with correlated data flows," *IEEE Communications Magazine*, vol. 59, no. 4, pp. 30–36, May 2021.

[149] X. Zheng, S. Zhou, and Z. Niu, "Urgency of Information for context-aware timely status updates in remote control systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 11, pp. 7237–7250, Jul. 2020.

[150] L. Zhang, Y. Shi, T. Chen, and B. Huang, "A new method for stabilization of networked control systems with random delays," *IEEE Transactions on automatic control*, vol. 50, no. 8, pp. 1177–1181, Aug. 2005.

[151] J. Lam, H. Gao, and C. Wang, "Stability analysis for continuous systems with two additive time-varying delay components," *Systems and Control Letters*, vol. 56, no. 1, pp. 16–24, Jan. 2007.

[152] D. Yue, Q. L. Han, and J. Lam, "Network-based robust h∞ control of systems with uncertainty," *Automatica*, vol. 41, no. 6, pp. 999–1007, Jun. 2005.

[153] K. Liu, E. Fridman, and L. Hetel, "Networked control systems in the presence of scheduling protocols and communication delays," *SIAM Journal on Control and Optimization*, vol. 53, no. 4, pp. 1768–1788, Aug. 2015.

[154] E. Fridman and M. Dambrine, "Control under quantization, saturation and delay: An LMI approach," *Automatica*, vol. 45, no. 10, pp. 2258–2264, Oct. 2009.

[155] L. X. Zhang, Z. P. Ning, and W. X. Zheng, "Observer-based control for piecewise-affine systems with both input and output quantization," *IEEE Transactions on Automatic Control*, vol. 53, no. 4, pp. 5858–5865, Nov. 2017.

[156] Y. Zhang, J. Liu, and X. Ruan, "Iterative learning control for uncertain nonlinear networked control systems with random packet dropout," *International Journal of Robust and Nonlinear Control*, vol. 29, no. 11, pp. 3529–3546, Jul. 2019.

[157] C. Wu, J. Liu, X. Jing, H. Li, and L. Wu, "Adaptive fuzzy control for nonlinear networked control systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 8, pp. 2420–2430, May 2017.

[158] H. Xu, Q. Zhao, and S. Jagannathan, "Finite-horizon near-optimal output feedback neural network control of quantized nonlinear discrete-time systems with input constraint," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 8, pp. 1776–1788, May 2015.

[159] C. Zhang and V. Lesser, "Coordinating multi-agent reinforcement learning with limited communication," in *International Conference on Autonomous agents and multi-agent systems (AAMAS)*, May 2013, pp. 1101–1108.

[160] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *arXiv preprint*, pp. 1–81, Aug. 2019.

[161] D. Kim, S. Moon, D. Hostallero, W. J. Kang, T. Lee, K. Son, and Y. Yi, "Learning to schedule communication in multi-agent reinforcement learning," in *7th International Conference on Learning Representations (ICLR)*, May 2019, pp. 1–17.

[162] M. H. M. Alkilabi, A. Narayan, and E. Tuci, "Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies," *Swarm Intelligence*, vol. 11, no. 3, pp. 185–209, Dec. 2017.

[163] M. Doniec, I. Topor, M. Chitre, and D. Rus, "Autonomous, localization-free underwater data muling using acoustic and optical communication," in *13th International Symposiumon Experimental Robotics (ISER)*, Jun. 2012, pp. 841–857.

[164] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, pp. 1–15, Jan. 2017.

[165] F. Mason, G. Nencioni, and A. Zanella, "Using distributed reinforcement learning for resource orchestration in a network slicing scenario," *IEEE/ACM Transactions on Networking*, pp. 1–15, 2022.

[166] 3GPP, "Service requirements for next generation new services and markets," 3rd Generation Partnership Project (3GPP), Technical Specification 22.261, 2016.

[167] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "A survey on 5G usage scenarios and traffic models," *IEEE Communications Surveys and Tutorials*, vol. 22, no. 2, pp. 905–929, Second quarter 2020.

[168] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, "Software-defined and virtualized future mobile and wireless networks: a survey," *Mobile Networks and Applications*, vol. 20, no. 1, pp. 4–18, Feb. 2015.

[169] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on computing, network, and storage resource constraints," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3298–3304, Oct. 2018.

[170] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega *et al.*, "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Communications magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[171] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 3, pp. 2429–2453, Third quarter 2018.

[172] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55 765–55 779, Sep. 2018.

[173] M. Richart, J. Baliosian, J. Serrat, and J. L. Gorricho, "Resource slicing in virtual wireless networks: A survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 462–476, Sep. 2016.

[174] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez, "Multi-tenant radio access network slicing: Statistical multiplexing of spatial loads," *IEEE Transactions on Networking*, vol. 25, no. 5, pp. 3044–3058, Oct. 2017.

[175] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, Jul. 2016.

[176] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, "Network resource management and QoS in SDN-enabled 5G systems," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2015, pp. 1–7.

[177] H. Halabian, "Distributed resource allocation optimization in 5G virtualized networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, Mar. 2019.

[178] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. C. Kozat, "A resource allocation framework for network slicing," in *IEEE International Conference on Computer Communications (INFOCOM)*, Apr. 2018, pp. 2177–2185.

[179] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, Jul. 2011.

[180] Y. Xiao, M. Hirzallah, and M. Krunz, "Distributed resource allocation for network slicing over licensed and unlicensed bands," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2260–2274, Oct. 2018.

[181] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Learning driven computation offloading for asymmetrically informed edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1802–1815, Aug. 2019.

[182] F. Fossati, S. Moretti, P. Perny, and S. Secci, "Multi-resource allocation for network slicing," *IEEE Transactions on Networking*, vol. 28, no. 3, pp. 1311–1324, Jun. 2020.

[183] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, "Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks," *Journal of Optical Communications and Networking*, vol. 9, no. 9, pp. 19–30, Sep. 2017.

[184] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "GAN-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 334–349, Feb. 2020.

[185] A. Thantharate, R. Paropkari, V. Walunj, and C. Beard, "Deepslice: A deep learning approach towards an efficient and reliable network slicing in 5G networks," in *IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference*, Oct. 2019, pp. 762–767.

[186] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, "Distributed deep learning at the edge: A novel proactive and cooperative caching framework for mobile edge networks," *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1220–1223, Aug. 2019.

[187] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "Gan-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 334–349, Feb. 2020.

[188] K. Suh, S. Kim, Y. Ahn, S. Kim, H. Ju, and B. Shim, "Deep reinforcement learning-based network slicing for beyond 5G," *IEEE Access*, vol. 10, pp. 7384–7395, 2022.

[189] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, "Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6063–6078, Sep. 2021.

[190] Y. Kim and H. Lim, "Multi-agent reinforcement learning-based resource management for end-to-end network slicing," *IEEE Access*, vol. 9, pp. 56 178–56 190, 2021.

[191] Y. Shao, R. Li, Z. Zhao, and H. Zhang, "Graph attention network-based drl for network slicing management in dense cellular networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2021, pp. 1–6.

[192] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[193] T. Kimura, T. Kimura, A. Matsumoto, and K. Yamagishi, "Balancing quality of experience and traffic volume in adaptive bitrate streaming," *IEEE Access*, vol. 9, pp. 15 530–15 547, 2021.

[194] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *12th International Conference on Neural Information Processing Systems*, Nov. 1999, pp. 1008–1014.

[195] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, Jun. 2016, pp. 1928–1937.

[196] H. Zhang, T. W. Weng, P. Y. Chen, C. J. Hsieh, and L. Daniel, "Efficient neural work robustness certification with general activation functions," in *Advances in neural information processing systems*, Dec. 2018, pp. 4939–4948.

[197] T. Jaakkola, S. P. Singh, and M. I. Jordan, "Reinforcement learning algorithm for partially observable Markov decision problems," in *Advances in neural information processing systems*, Nov. 1995, pp. 345–352.

[198] X. S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver press, 2008.

[199] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, Jun. 1994.

[200] L. Cominardi, L. M. Contreras, C. J. Bcrnardos, and I. Berberana, "Understanding QoS applicability in 5G transport networks," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, Jun. 2018, pp. 1–5.

[201] 3GPP, "System architecture for the 5G System (5GS)," 3rd Generation Partnership Project (3GPP), Technical Specification 23.501, 2017.

[202] D. Sattar and A. Matrawy, "DSAF: Dynamic slice allocation framework for 5G core network," *arXiv preprint*, May 2019.

[203] A. Chiha, M. Van der Wee, D. Colle, and S. Verbrugge, "Network slicing cost allocation model," *Journal of Network and Systems Management*, vol. 28, pp. 627–659, March 2020.

[204] F. Voigtländer, A. Ramadan, J. Eichinger, C. Lenz, D. Pensky, and A. Knoll, "5G for robotics: Ultra-low latency control of distributed robotic systems," in *International Symposium on Computer Science and Intelligent Controls*, oct 2017, pp. 69–72.

[205] J. Sachs, L. A. Andersson, J. Araújo, C. Curescu, J. Lundsjö, G. Rune, E. Steinbach, and G. Wikström, "Adaptive 5G low-latency communication for tactile internet services," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 325–349, Feb. 2019.

[206] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, "Low-latency networking: Where latency lurks and how to tame it," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 280–306, Feb. 2019.

[207] 3GPP, "Policy and charging control architecture," 3rd Generation Partnership Project (3GPP), Technical Specification 23.203, 2005.

[208] C. J. Geyer, "Practical Markov chain Monte Carlo," *Statistical science*, vol. 7, no. 4, pp. 473–483, Nov. 1992.

[209] "GARR high capacity network," https://www.garr.it/en/infrastructures/network-infrastructure/network-map, accessed: 2021-03-31.

[210] F. Mason, F. Chiariotti, and A. Zanella, "No free lunch: Balancing learning and exploitation at the network edge," in *IEEE International Conference on Communications (ICC)*, 2022, pp. 631–636.

[211] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions," *IEEE Access*, vol. 7, pp. 137 184–137 206, Sep. 2019.

[212] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. C. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 44–52, Apr. 2019.

[213] O. Gómez-Carmona, D. Casado-Mansilla, F. A. Kraemer, D. López-de Ipiña, and J. García-Zubia, "Exploring the computational cost of machine learning at the edge for human-centric Internet of Things," *Future Generation Computer Systems*, vol. 112, pp. 670–683, Nov. 2020.

[214] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*, Jul. 2017, pp. 3987–3995.

[215] R. S. Villaça and R. Stadler, "Online learning under resource constraints," in *International Symposium on Integrated Network Management*, May 2021, pp. 134–142.

[216] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint*, Jul. 2020.

[217] D. Liu, H. Kong, X. Luo, W. Liu, and R. Subramaniam, "Bringing AI to edge: From deep learning's perspective," *arXiv preprint*, Nov. 2020.

[218] M. Chan, D. Scarafoni, R. Duarte, J. Thornton, and L. Skelly, "Learning network architectures of deep CNNs under resource constraints," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2018, pp. 1703–1710.

[219] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. Nguyen, and C. S. Hong, "Federated learning for edge networks: Resource optimization and incentive mechanism," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 88–93, Nov. 2020.

[220] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[221] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," in *Conference on Computer Communications (INFOCOM)*, May 2021, pp. 1–10.

[222] I. Jang, H. Kim, D. Lee, Y.-S. Son, and S. Kim, "Knowledge transfer for on-device deep reinforcement learning in resource constrained edge computing systems," *IEEE Access*, vol. 8, pp. 146 588–146 597, Aug. 2020.

[223] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *27th International Conference on Artificial Neural Networks (ICANN)*, Oct. 2018, pp. 270–279.

[224] T. Everitt, M. Hutter, R. Kumar, and V. Krakovna, "Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective," *Synthese*, vol. 198, pp. 6435—6467, May 2021.

[225] S. J. Julier, "The scaled unscented transformation," in *American Control Conference*, May 2002, pp. 4555–4559.

[226] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley and Sons, 1994.

[227] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, Jan. 1992.

[228] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *30th Annual ACM symposium on Theory of computing*, 1998, pp. 604–613.

List of Publications

International Journals

J1. **F. Mason**, G. Nencioni, and A. Zanella, "Using Distributed Reinforcement Learning for Resource Orchestration in a Network Slicing Scenario," in *IEEE Transactions on Networking*, early access, doi: 10.1109/TNET.2022.3187310.

J2. **F. Mason**, F. Chiariotti, M. Capuzzo, D. Magrin, A. Zanella, and M. Zorzi, "Remote tracking of UAV swarms via 3D mobility models and LoRaWAN communications," in *IEEE Transactions on Wireless Communications*, vol. 21, no. 5, pp. 2953-2968, Oct. 2021.

J3. F. Venturini, **F. Mason**, F. Pase, F. Chiariotti, A. Testolin, and A. Zanella, "Distributed Reinforcement Learning for Flexible and Efficient UAV Swarm Control," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 3, pp. 955-969, Sep. 2021.

J4. **F. Mason**, M. Giordani, F. Chiariotti, A. Zanella, and M. Zorzi, "An Adaptive Broadcasting Strategy for Efficient Dynamic Mapping in Vehicular Networks," in *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5605-5620, May 2020.

J5. M. Dalla Cia, **F. Mason**, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, and A. Zanella, "Using Smart City Data in 5G Self-Organizing Networks," in *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 645-654, Apr. 2018.

Conference Proceedings

C1. M. Drago, T. Zugno, **F. Mason**, M. Giordani, M. Boban, and M. Zorzi, "Artificial Intelligence in Vehicular Wireless Networks: A Case Study Using ns-3," in *Workshop on ns-3 (WNS3)*, Jun. 2022, pp. 1-8.

C2. **F. Mason**, F. Chiariotti, and A. Zanella, "No Free Lunch: Balancing Learning and Exploitation at the Network Edge," in *IEEE International Conference on Communications (ICC)*, Seoul, South Korea, May 2022, pp. 1-6.

C3. **F. Mason**, M. Drago, T. Zugno, M. Giordani, M. Boban, and M. Zorzi, "A Reinforcement Learning Framework for PQoS in a Teleoperated Driving Scenario," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2022, pp. 114-119.

C4. **F. Mason**, G. Nencioni, and A. Zanella, "A Multi-Agent Reinforcement Learning Architecture for Network Slicing Orchestration," in *19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, Jun. 2021, pp. 1-8.

C5. **F. Mason**, F. Chiariotti, F. Campagnaro, A. Zanella, and M. Zorzi, "Low-cost AUV Swarm Localization Through Multimodal Underwater Acoustic Networks," in *OCEANS 2020: Singapore – U.S. Gulf Coast*, Oct. 2020, pp. 1-7.

C6. F. Venturini, **F. Mason**, F. Pase, F. Chiariotti, A. Testolin, and A. Zanella, "Distributed reinforcement learning for flexible UAV swarm control with transfer learning capabilities," in *6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet '20)*, Toronto, Canada, Jun. 2020, pp. 1-6.

C7. **F. Mason**, F. Chiariotti, M. Capuzzo, D. Magrin, A. Zanella, and M. Zorzi, "Combining LoRaWAN and a New 3D Motion Model for Remote UAV Tracking," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Toronto, Canada, Jul. 2020, pp. 412-417.

C8. **F. Mason**, M. Giordani, F. Chiariotti, A. Zanella, and M. Zorzi, "Quality-aware broadcasting strategies for position estimation in VANETs," in *25th European Wireless (EW)*, Aarhus, Denmark, May 2019, pp. 1-8.

C9. M. Dalla Cia, **F. Mason**, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, and A. Zanella, "Mobility-aware handover strategies in smart cities," in *2017 International Symposium on Wireless Communication Systems (ISWCS)*, Bologna, Italy, Aug. 2017, pp. 438-443.

Book Chapters

B1. F. Pase, **F. Mason**, P. Testolina, M. Lecci, A. Zanella, and M. Zorzi, "Smart Data Gathering for Network Optimization," in *Machine Learning and 5G/6G Networks: Interplay and Synergies*, Texmat, 2021, pp. 147–170.

B2. A. Zanella, **F. Mason**, P. Pluchino, G. Cisotto, V. Orso, and L. Gamberini, "DOMHO: A Smart Assisted-Living Solution for Fragile People," in *ICT for Health: Sensing, Data Analysis, Applications*, Texmat, 2021, pp. 19-37.

Acknowledgments

I could never complete this work without the invaluable support of the people that have been by my side for the past three years and the rest of my life.

First, I want to express my gratitude to Andrea Zanella, who led me step by step in my scientific career taking care to push me toward what was best suited to my spirit. I am very lucky to have found a true friend as well as a guide in my supervisor.

I thank Federico Chiariotti, my inseparable companion in projects and articles since my master's thesis. Not continuing to work side by side will compromise my confidence in all future scientific activities.

I thank Giorgio Quer, my supervisor at Scripps Research in San Diego, for leading my research during the months in California, offering me an immense growth opportunity, and supporting me both inside and outside the office.

I thank Michele Zorzi, Gianfranco Nencioni, Alberto Testolin, Michele Polese, Filippo Campagnaro, Giulia Cisotto, Davide Magrin, Martina Capuzzo, Francesco Pase, Federico Venturini, Matteo Gadaleta, and all the other colleagues with whom I have collaborated over the years.

I thank my family, who continue to give me courage without the slightest sign of hesitation. I thank my mother for the infinite love she always donates to others. I thank my father for the serenity with which he keeps us united against all difficulties. I thank my brother for the curiosity with which he helps me to question the world.

I thank my cousin Chiara, my grandmother Bianca, and my uncles Vania and Gaspare, for always being present, and loving me, each one in his own way. I thank my grandfather Dino and my aunt Monica, whose memory fills me with love more than any physical hugs.

I thank the friends who welcomed me to California: Clara, Bodhi, Mike, and Raiaan. Having met you is worth more than any professional satisfaction I could ever receive.

I thank the colleagues of ICTea: Matteo, Francesca, Tommaso, Marco, and Alberto. The dream of these three years, from the living room of my house to the halls of Miramare in Trieste, has enormously enriched my life and made me proud every day.

I thank Davide and Carlotta who have been my inseparable classmates since the second day of my bachelor's degree. I thank all the special people with whom I have shared hours of desperation and serenity before and after the exams: Adami, Pitt, DB, Fabio, Marta, Massimo, Davide, Jessica, Emiliano, Cristina, Coccolo, Lovi, Enrico, and Matilde.

I thank Mattia and Paolo, with whom I learned to whip eggs without electric mixers: I like to think that tiramisù is much better this way. I thank Marta, Alex, Ana, Hanne, and Lorenzo,

who accompanied me to discover Catalonia and have continued to be by my side to discover the world. I thank Negra and Analia, my Uruguayan sisters, who made me feel part of a family from the very first second.

I thank the squirrels of the Clan Universitario of Padua: Fatt, Laura, Edo, Pitto, Ilaria, Otta, and the honorary scout Teresa. Every time I look back, I am amazed at how magical the friendship we have built is.

I thank all those who have shared a roof with me over all these years: Giulia, Cianfa, Pianta, Plaz, and Yann. I thank the founding members of Portello 24: Albert, Ale, and Grazio. I will never be grateful enough to destiny for bringing us together.

I thank the old friends of San Vito al Tagliamento. I thank Carlo, the most reliable travel companion ever. I thank Pier, who is always available for a beer, possibly around a bonfire on the grava's banks.

I thank all my schoolmates and especially Alice, Giulia, and Genny: I know I can always count on you, especially when there is a film to see or an escape room to overcome.

I thank the fellows of QNG production: Piero, Emilio, Dj, and Dario. The afternoons spent dreaming of the cinema with you, between cameras, PlayStation games, and Age of Mythology derbies, represent the happiest moments of my life.

I thank Scouting for the values and friends it has given me. I thank the educators who have accompanied me since I was a child: Checco, Elisa, Guru, and Marina. I thank the companions of routes and summer camps: Michele, Currywurst, Chiara, Silvia, Francesca, Beatrice, Marco, Samuele, and Cuc. I thank the people with whom I share the vocation for service: Sofia, Francesco, Daniela, and all the other members of the San Vito 1 community.

I thank Eugi who, despite my persistence in making mistakes, wants the best for me.