Deep Reinforcement Learning and Generative Adversarial Modeling in Traffic Applications

A Dissertation

submitted to the designated

by the Assembly

of the Department of Computer Science and Engineering

Examination Committee

by

Christos Spatharis

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

University of Ioannina

School of Engineering

Ioannina 2023

Advisory Committee:

- **Konstantinos Blekas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **George Vouros**, Professor, Department of Digital Systems, University of Piraeus

Examining Committee:

- **Konstantinos Blekas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **George Vouros**, Professor, Department of Digital Systems, University of Piraeus
- **Andreas G. Stafylopatis**, Professor, School of Electrical and Computer Engineering, National Technical University of Athens (NTUA)
- **George Paliouras**, Senior Researcher, Head of the Intelligent Information Systems division of the Institute of Informatics and Telecommunications at NCSR Demokritos, Athens
- **Christoforos Nikou**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Kostas Vlachos**, Assistant Professor, Department of Computer Science and Engineering, University of Ioannina

Dedication

I would like to dedicate this thesis to my family for their unconditioned support on this journey.

Acknowledgements

I would like to express my sincere gratitude and appreciation to all those who have contributed to the completion of this thesis.

First and foremost, I would like to extend my deepest graditude to my thesis supervisor Professor Konstantinos Blekas, whose expertise, guidance, and unwavering support have been instrumental throughout this endeavor. Also, I would like to thank him for his seamless cooperation and the time he spent for my progress throughout these years, dating back to 2015. The lessons I have learned from him, will always accompany me throughout my career.

Furthermore, I would like to thank Professor George Vouros for the opportunity he gave me to collaborate with him and his lab. I really appreciate his support valuable advice, and I hope to meet again in the future.

I am also grateful to Professor Aristidis Lykas and Assistant Professor Kostas Vlachos for being in my supervising committee and for their valuable time, and suggestions. I would also like to thank Professor Christoforos Nikou, Professor Andreas G. Stafylopatis, and Senior Researcher George Paliouras for their dedicated involvement in reviewing my dissertation.

I am deeply thankful to all my friends and colleagues for their continuous support and encouragement throughout this journey. I would like to thank Piyabhum Chaysri, Georgios Spithakis and Theodore Tranos for their collaboration in our office, and I hope to work with them again in the future.

Furthermore, I would like to express my heartfelt appreciation to my parents Georgios and Aggeliki, as well as to my sister Efi for their love, encouragement, and sacrifices. I am grateful for their understanding, and patience, which have allowed me to pursue my academic dreams.

Lastly, I want to say a special thank you to Eirini for supporting me and believing in me throughout the entire process of completing this thesis.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Glossary

AI	Artificial Intelligence
AIL	Adversarial Imitation Learning
AE	Autoencoder
A3C	Asynchronous Advantage Actor-Critic
ATC	Air Traffic Control
ATE	Along Track Error
ATM	Air Traffic Management
BC	Behavioral Cloning
CNN	Convolutional Neural Network
col-MARL	Collaborative Multi-agent Reinforcement Learning
CTE	Cross Track Error
cVAE	Conditional Variational Auto-Encoder
DCB	Demand Capacity Balance
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DL	Deep Learning
DPL	Direct Policy Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DTW	Dynamic Time Warping
ETA	Estimated Time of Arrival
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Network
HRL	Hierarchical Reinforcement Learning
HCMARL	Hierarchical Collaborative Multi-Agent Reinforcement Learning
HMAMDP	Hierarchical Multi-agent Markov Decision Process

HMIRL	Hierarchical Multi-agent Independent Reinforcement Learning
IL	Imitation Learning
i-MARL	Independent Multi-agent Reinforcement Learning
IRL	Inverse Reinforcement Learning
IDM	Intelligent Driver Model
LSTM	Long Short Term Memory
MAMDP	Multi-agent Markov Decision Process
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-agent System
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-layer Perceptron
MMIL	Multi-module Multi-modal Imitation Learning
MCTS	Monte Carlo Tree Search
NM	Network Manager
NN	Neural Network
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
RMSE	Root Mean Squared Error
SMDP	Semi-Markov Decision Process
SMIL	Single-module Multi-modal Imitation Learning
TBO	Trajectory Based Operations
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
VAE	Variational Autoencoder

Abstract

Christos Spatharis, Ph.D., Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2023.
Deep Reinforcement Learning and Generative Adversarial Modeling in Traffic Applications.
Advisor: Konstantinos Blekas, Professor.

Artificial intelligence (AI) has brought significant transformations in various domains of everyday life, revolutionizing human-machine interactions. Intelligent agents, the fundamental components of AI systems, have the ability to perceive, reason, and act in their environment to achieve specific goals. Ranging from simple rule-based systems to complex deep learning models, these agents can be trained using a variety of learning schemes falling under the broader umbrella of machine learning (ML). One influential sub-field of ML that has gained considerable attention is reinforcement learning (RL). RL focuses on training intelligent agents to make sequential decisions by interacting with an environment, drawing inspiration from trial-and-error learning observed in humans and animals. Another ML technique, known as imitation learning (IL), combines supervised learning and RL principles by learning from expert demonstrations.

This dissertation explores the application of both RL and IL techniques in the context of traffic applications, addressing the significant challenges of (a) congestion management and (b) trajectory modeling. Traffic applications play a vital role in modern society, as they encompass a wide range of systems and technologies aimed at managing and optimizing the behavior and the navigation of vehicles including various modes of transportation such as ground vehicles, roadway systems, air vehicles, and sea vessels. Congestion occurs when the demand for shared resources exceeds supply, leading to reduced efficiency of the overall system. Trajectory modeling involves analyzing and predicting expert behaviors based on historical demonstrated

data. These challenges are not limited to the traffic domain but may also be extended to other fields where similar optimization and decision-making problems arise. This PhD is separated in three parts where the following problems are studied: (a) urban traffic navigation, (b) air traffic management, and (c) aircraft trajectory prediction.

In the first part, the aim is to create efficient multi-agent systems for controlling and navigating fleets of vehicles in unsignalized large-scale urban road networks with complex scenarios and noise. By employing multi-agent reinforcement learning (MARL) techniques, the study seeks to navigate vehicles safely, preventing collisions, and minimizing traveling time. The proposed research contributes to the advancement of intelligent traffic management systems by utilizing RL techniques to optimize traffic flow and reduce congestion in urban areas.

The second part focuses on tackling congestion problems in the aviation domain, particularly focusing on the demand and capacity balance (DCB) problem in air traffic management (ATM). By employing MARL schemes and leveraging hierarchical frameworks, the study seeks to minimize flight delays, optimize airspace utilization, and reduce fuel consumption and operating costs. The cooperative behavior of the involved flights is enabled to achieve more efficient use of the airspace and enhance overall performance of the multi-agent system.

The final part studies generative models and trajectory modeling techniques in the aviation domain. Trajectory prediction is of an utmost importance for congestion management, and IL techniques offer a promising approach by training agents to imitate expert behaviors. Multi-modal imitation learning can further enhance trajectory prediction by capturing various behavioral patterns exhibited during flight execution. By leveraging expert data and modeling distinct patterns, the proposed approach improves the accuracy and robustness of trajectory prediction systems, leading to enhanced air traffic management, optimized route planning, and safer and more efficient flights.

Εκτεταμενη Περιληψη

Χρήστος Σπαθάρης, Δ.Δ., Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2023.
Βαθιά Ενισχυτική Μάθηση και Παραγωγική Ανταγωνιστική Μοντελοποίηση για Εφαρμογές Διαχείρισης και Ελέγχου Ροής Κυκλοφορίας.
Επιβλέπων: Κωνσταντίνος Μπλέκας, Καθηγητής.

Η τεχνητή νοημοσύνη (ΤΝ) έχει επιφέρει σημαντικές αλλαγές σε διάφορους τομείς της καθημερινής ζωής, φέρνοντας επανάσταση στις αλληλεπιδράσεις ανθρώπου-μηχανής. Οι ευφυείς πράκτορες, που αποτελούν τα θεμελιώδη συστατικά των συστημάτων ΤΝ, έχουν την ικανότητα να αντιλαμβάνονται, να συλλογίζονται και να ενεργούν στο περιβάλλον τους για να επιτύχουν συγκεκριμένους στόχους. Από απλά συστήματα βασισμένα σε κανόνες μέχρι πολύπλοκα μοντέλα βαθιάς μάθησης, οι ευφυείς μπορούν να εκπαιδευτούν χρησιμοποιώντας μια ποικιλία τεχνικών μάθησης που εμπίπτουν στο ευρύτερο πεδίο της μηχανικής μάθησης. Ενα σημαντικό υποπεδίο της μηχανικής μάθησης που έχει αντλήσει σημαντική προσοχή, είναι η ενισχυτική μάθηση. Η ενισχυτική μάθηση επικεντρώνεται στην εκπαίδευση ευφυών πρακτόρων για τη λήψη διαδοχικών αποφάσεων αλληλεπιδρώντας με ένα περιβάλλον, αντλώντας έμπνευση από τη μάθηση δοκιμής και σφάλματος που παρατηρείται σε ανθρώπους και ζώα. Μια άλλη τεχνική μηχανικής μάθησης, γνωστή και ως μάθηση μέσω μίμησης, συνδυάζει την εποπτευόμενη μάθηση και τις αρχές της ενισχυτικής μάθησης μαθαίνοντας από τις επιδείξεις ειδικών.

Η παρούσα διατριβή διερευνά την εφαρμογή τόσο των τεχνικών ενισχυτικής μάθησης όσο και αυτών της μάθησης μέσω μίμησης στο πλαίσιο των εφαρμογών κυκλοφορίας, αντιμετωπίζοντας τις σημαντικές προκλήσεις της (α) διαχείρισης συμφόρησης και (β) μοντελοποίησης τροχιάς. Οι εφαρμογές της κυκλοφορίας διαδραματίζουν ζωτικό ρόλο στη σύγχρονη κοινωνία, καθώς περιλαμβάνουν ένα ευρύ φάσμα συστημάτων και τεχνολογιών που αποσκοπούν στη διαχείριση και βελτιστοποίηση

της συμπεριφοράς και της πλοήγησης των οχημάτων, συμπεριλαμβανομένων των διαφόρων τύπων μεταφοράς, όπως τα οχήματα εδάφους, τα οδικά συστήματα, τα εναέρια οχήματα και τα θαλάσσια σκάφη. Η συμφόρηση συμβαίνει όταν η ζήτηση για διαμοιραζόμενους πόρους υπερβαίνει την προσφορά, οδηγώντας σε μειωμένη αποτελεσματικότητα του συνολικού συστήματος. Η μοντελοποίηση της τροχιάς περιλαμβάνει την ανάλυση και την πρόβλεψη συμπεριφορών από ειδικούς με βάση ιστορικά δεδομένα παραδειγμάτων. Αυτές οι προκλήσεις δεν περιορίζονται στον τομέα της κυκλοφορίας, αλλά μπορούν επίσης να επεκταθούν και σε άλλους τομείς όπου προκύπτουν παρόμοια προβλήματα βελτιστοποίησης και λήψης αποφάσεων. Το παρόν διδακτορικό χωρίζεται σε τρία μέρη όπου μελετώνται τα ακόλουθα προβλήματα: (α) πλοήγηση αστικής κυκλοφορίας, (β) διαχείριση εναέριας κυκλοφορίας, και (γ) πρόβλεψη τροχιάς αεροσκαφών.

Στο πρώτο μέρος, ο στόχος είναι να δημιουργηθούν αποτελεσματικά πολυπρακτορικά συστήματα για τον έλεγχο και την πλοήγηση στόλων οχημάτων σε μη σηματοδοτούμενα μεγάλης κλίμακας αστικά οδικά δίκτυα με σύνθετα σενάρια και θόρυβο. Με τη χρήση τεχνικών πολυπρακτορικής ενισχυτικής μάθησης, η μελέτη επιδιώκει να πλογήσει με ασφάλεια τα οχήματα, αποτρέποντας τις συγκρούσεις και ελαχιστοποιώντας τον χρόνο ταξιδιού. Η προτεινόμενη έρευνα συμβάλλει στην πρόοδο των ευφυών συστημάτων διαχείρισης της κυκλοφορίας με τη χρήση τεχνικών ενισχυτικής μάθησης για τη βελτιστοποίηση της ροής της κυκλοφορίας και τη μείωση της συμφόρησης στις αστικές περιοχές.

Το δεύτερο μέρος επικεντρώνεται στην αντιμετώπιση των προβλημάτων συμφόρησης στον τομέα των αερομεταφορών, με ιδιαίτερη έμφαση στο πρόβλημα ισορροπίας μεταξύ ζήτησης και χωρητικότητας στην διαχείριση του εναέριας κυκλοφορίας. Χρησιμοποιώντας πολυπρακτορικά συστήματα ενισχυτικής μάθησης και αξιοποιώντας ιεραρχικές δομές, η μελέτη επιδιώκει να ελαχιστοποιήσει τις καθυστερήσεις των πτήσεων, να βελτιστοποιήσει τη χρήση του εναέριου χώρου και να μειώσει την κατανάλωση καυσίμων και το λειτουργικό κόστος. Η συνεργατική συμπεριφορά των εμπλεκόμενων πτήσεων επιτρέπει την αποτελεσματικότερη χρήση του εναέριου χώρου και την ενίσχυση της συνολικής διαχείρισης του πολυπρακτορικού συστήματος.

Το τελευταίο μέρος μελετά παραγωγικά μοντέλα μάθησης και τεχνικές μοντελοποίησης τροχιών στον εναέριο τομέα. Η πρόβλεψη τροχιάς είναι υψίστης σημασίας για τη διαχείριση της συμφόρησης και οι τεχνικές μάθησης μέσω μίμησης προσφέρουν μια πολλά υποσχόμενη προσέγγιση εκπαιδεύοντας πράκτορες με σκοπό να μι-

μηθούν τις συμπεριφορές των ειδικών. Η πολυτροπική μάθηση μέσω μίμησης μπορεί να ενισχύσει περαιτέρω την πρόβλεψη της τροχιάς, αναλύοντας διάφορα πρότυπα συμπεριφοράς που εμφανίζονται κατά την εκτέλεση των πτήσεων. Αξιοποιώντας τα δεδομένα των ειδικών και μοντελοποιώντας διαφορετικά μοτίβα, η προτεινόμενη προσέγγιση βελτιώνει την ακρίβεια και την ευρωστία των συστημάτων πρόβλεψης τροχιάς, οδηγώντας σε βελτιωμένη διαχείριση της εναέριας κυκλοφορίας, βελτιστοποιημένο σχεδιασμό διαδρομών και ασφαλέστερες και αποδοτικότερες πτήσεις.

Chapter 1

Introduction

Artificial intelligence (AI) has emerged as a technology that brings enormous changes in the every-day life, revolutionizing various fields and reshaping the way we interact with machines. In today's world, AI is ubiquitous, from self-driving cars and medical applications, to recommendation systems and personal assistance, and much more. At its core, AI seeks to develop intelligent agents that can perceive, reason, and act in their environment to achieve specific goals.

1.1 Artificial intelligent agents

Machine learning (ML) is a sub-field of AI that plays a fundamental role in enabling intelligent agents to learn and adapt from data [1]. AI encompasses the broader concept of creating intelligent systems that can simulate human-like intelligence, while ML focuses on algorithms and techniques that allow these systems to automatically learn and improve from experience (data), i.e. without being explicitly programmed via training. Using ML algorithms, intelligent agents are able to process huge amounts

of data, extract patterns, and make sequential decisions based on their knowledge. This synergy between ML and AI empowers intelligent agents to become more autonomous, adaptive, and capable of tackling complex tasks in diverse domains.

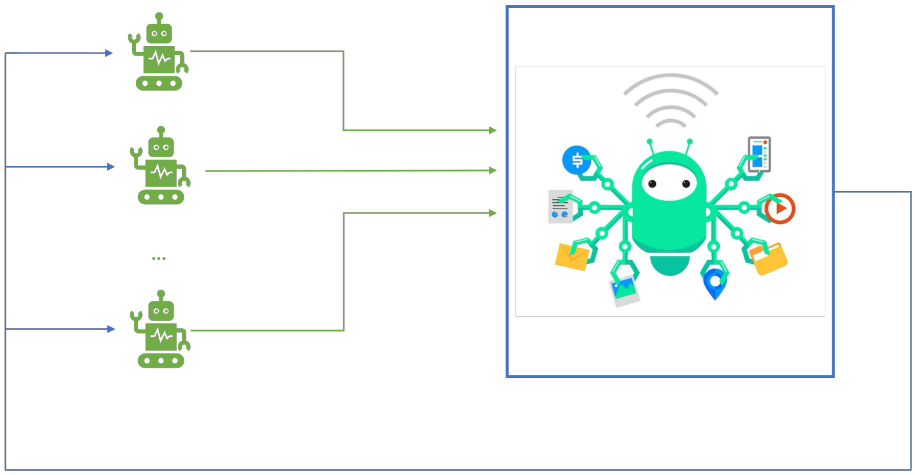


Figure 1.1: Intelligent agents acting on a shared environment.

Intelligent agents (Fig. 1.1) are the building blocks of AI systems. They are capable of perceiving their environment through sensors, processing the information, and taking actions in order to achieve desirable outcomes. Methodologically, they can range from simple rule-based systems to complex deep learning models. Various learning schemes can be utilized in order to train intelligent agents, which fall under the broader umbrella of ML. Specifically, a basic taxonomy may include the following: supervised learning, unsupervised learning and reinforcement learning.

A multi-agent system (MAS) is a group of autonomous, interacting entities sharing an environment, which they perceive with sensors and act with actuators. It provides an alternative to a centralized system controlled by a single agent, when achieving centralized control might be either costly or non-viable. Besides, the potential benefits from MAS include:

- computational speed-up due to parallel processing,
- enhanced robustness as the neighboring agents may take on the task of a malfunctioning agent,
- more effective policies due to the use of coordination mechanisms, allowing to maximize a common goal without having a centralized system,

- a way to decompose a complex task into sub-tasks, thus allowing the system to handle larger problems by adding more agents and scale quickly, and
- reduced learning cycles due to communication and information sharing.

Supervised learning is a ML approach where the algorithm learns from labeled training data: input examples accompanied with their corresponding target labels. The goal is to build a predictive model that can accurately generalize and make predictions on unseen data. This type of learning has been prevalent in various applications, such as image recognition, spam detection, medical diagnosis, natural language processing, and sentiment analysis. Supervised learning is commonly associated with discriminative modeling, where the primary objective is to classify or predict the labels of new, unseen data points based on the learned patterns and relationships from the labeled training set.

On the other hand, **unsupervised learning** is a ML approach where the model learns from unlabeled data (i.e. without target labels). The objective is to discover hidden patterns, structures, or relationships within the data. Unsupervised learning finds applications in clustering, anomaly detection, dimensionality reduction, and recommendation systems. By analyzing the inherent structure of the data, unsupervised learning algorithms can group similar data points, identify outliers, reduce the complexity of high-dimensional data, and uncover hidden patterns. This type of learning is often associated with *generative modeling*, where the aim is to figure out the underlying probability distribution of the input data. Unsupervised learning algorithms can be used to generate new samples that resemble the characteristics of the original data distribution in tasks such as data synthesis and data augmentation.

Generative modeling includes approaches like generative adversarial networks (GANs), variational autoencoders (VAEs) and diffusion models. It focuses on capturing the underlying data distribution and generating new samples that mimic the training data. Discriminative modeling, conversely, aims to learn the decision boundary that best separates different classes, solving accurately classification and prediction tasks.

Another influential sub-field of ML that has gained more and more attention especially during the last two decades is **reinforcement learning** (RL). It focuses on training intelligent agents to make sequential decisions based on their interactions with the environment, and draws inspiration from how humans and animals learn

by trial and error. In RL, an agent learns to maximize a reward function by taking actions in an environment. Through repeated exploration and feedback in the form of reward or penalty, the agent can gradually improve its decision-making capabilities and optimize its behavior.

This learning framework has found significant success in various domains, including robotics, gaming, autonomous vehicles, finance, and resource management. RL has been used to teach agents to perform complex tasks, achieve superhuman performance in games (e.g. Go, chess and Atari), and optimize resource allocation, among others. Its flexibility and ability to learn from experience make it a powerful technique for tackling real-world problems.

Imitation learning (IL), also known as *learning from demonstrations*, is another ML technique that lies at the intersection of supervised learning and reinforcement learning. It leverages the principles of supervised learning by learning from expert demonstrations, where the expert actions are the label to the input states. The connection with RL arises from the interactions of the imitator within an environment. In inverse reinforcement learning (IRL) - which is a sub-field of imitation learning - the agent learns from expert demonstrations by extracting the expert policy, and then fine-tunes its behavior through typical RL methods. This allows IL to benefit from both schemes by combining the efficiency of supervised learning and the adaptability of RL to deal with complex real-world tasks.

Moreover, IL can be particularly useful when it is difficult to design a reward function for a RL problem. This technique has found applications in various fields, including robotics, autonomous driving, and trajectory modeling. For example, IL can be used in robotics to teach robots how to perform complex tasks by observing human demonstrations, while in autonomous driving, it can help train self-driving cars to navigate safely and efficiently by learning from the behavior of expert drivers.

Multi-modal learning constitutes a new type of learning that has made its appearance over the last years. It refers to the process of training models to understand and process information from multiple modalities, such as text, images, audio, and video. By combining different types of data, multi-modal learning aims to capture deeper representations of the underlying information. As an instance, in image captioning, multi-modal modeling could be used to generate descriptive captions by understanding both some visual content of an image and a textual information. Additionally, multi-modal training can be used in cases with data that belong to multiple clusters

following different patterns, where the goal is to identify their distinct modalities. This can provide valuable insights into the underlying factors driving the different behaviors observed in the data. For example, in transportation planning, identifying different clusters of trajectories can help optimize route planning, resource allocation, and traffic management strategies for each specific modality. Multi-modal learning allows models to leverage the specific information present in different modalities, leading to more robust and accurate predictions.

1.2 Traffic applications

Traffic applications play a vital role in modern society, as they encompass a wide range of systems and technologies aimed at managing and optimizing the behavior and the navigation of vehicles including various modes of transportation such as ground vehicles, roadway systems, air vehicles, and sea vessels. However, two of the major challenges are *congestion management* and *trajectory modeling*. Congestion occurs when the demand for road space exceeds its capacity, leading to delays, increased travel times, and reduced overall efficiency. Trajectory modeling involves the analysis and prediction of vehicle movements based on historical and real-time data.

Both problems are not only very significant in the traffic domain, but also they are applicable in a general-purpose manner and can be relevant in various domains where similar challenges arise. These problems are characterized by the need to optimize movements, predict patterns, and make informed decisions based on available data. For instance, in logistics and supply chain management, congestion can occur in distribution centers, warehouses, or ports, impacting the flow of goods and causing delays. By applying congestion management strategies and leveraging trajectory modeling techniques, it is possible to optimize the movement of goods, streamline operations, and reduce bottlenecks.

1.2.1 Congestion management

Congestion problems pose a significant challenge in everyday life concerning various fields, including transportation networks, communication networks, and resource management. As communities continue to grow and urban areas become more densely populated, congestion becomes a considerable trouble for individuals, busi-

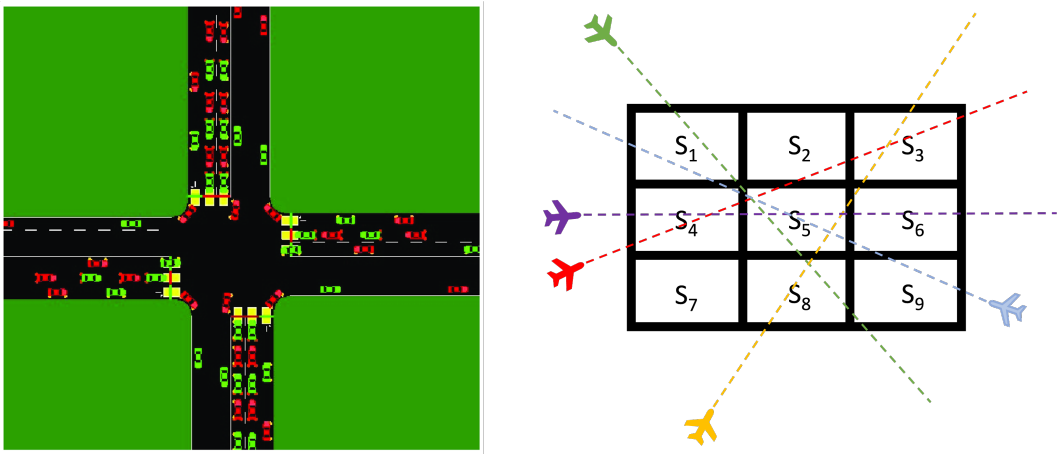


Figure 1.2: Congestion problems in urban and aviation domains.

nesses, and overall economic productivity.

In urban road networks, congestion has become an inconvenient issue in many cities. Traffic congestion not only leads to frustration and stress for drivers, but also results in significant economic costs due to increased fuel consumption and traveling time. Traditional traffic management systems with hard-coded rules have limited adaptability in dynamically changing traffic conditions and fail to optimize the traffic flow in complex scenarios. This is where RL could offer a convenient platform for flexible solutions and play a key role to smart cities and future plans of societies.

In the context of urban traffic congestion, RL algorithms can be utilized in order to develop intelligent traffic management systems that will guide the autonomous vehicles through roads and intersections without creating bottlenecks and hindering the traffic flow. By using RL, traffic management systems can learn either from historical traffic data or real-time sensor information, which can be then utilized to make data-driven decisions and reduce the congested situations and the traveling time. Such systems will be able to identify traffic patterns, forecast bottlenecks, and adjust the strategies of the participating autonomous vehicles to eliminate congestion. Through optimization of the traffic flow, the traveling time can be significantly reduced, leading to reduced fuel consumption, while also promoting safer driving behaviors.

The motivation of the research study of this application is to create efficient multi-agent systems that can simultaneously serve a society of intelligent agents to control and navigate fleets of vehicles inside unsignalized urban road networks with complex scenarios under the presence of noise. Effectively dealing with these problems can lead to better utilization of the multi-agent system's resources and prevent undesired

congested situations. This can be achieved by devising multi-agent reinforcement learning (MARL) techniques, aiming to navigate the vehicles safely (i.e. without collisions) and reduce their traveling time.

Similarly, congestion problems may also arise in the aviation domain, where overcrowded airspace sectors require the enforcement of delays to flights in order to avoid hazardous situations. Inevitably, imposing delays gives rise to problems concerning additional operating costs, extended work hours for the operators, and frustration for the passengers. Formally, when the demand for an airspace sector exceeds its maximum capacity, then it results in congested areas known as *hotspots*, and this ends up forming the *demand and capacity balance (DCB)* problem.

The motivation of studying this problem is to design MARL schemes that are based on the co-operation of the involved flights, so as to minimize the imposed delays, which will eventually lead to more efficient use of the airspace, and reduce the fuel consumption and the operating costs. In addition, this interesting problem allows to address state abstraction representational issues in the structure of the RL agents through the use of hierarchical frameworks, which enable effective planning and exploration with improved generalization.

1.2.2 Trajectory modeling

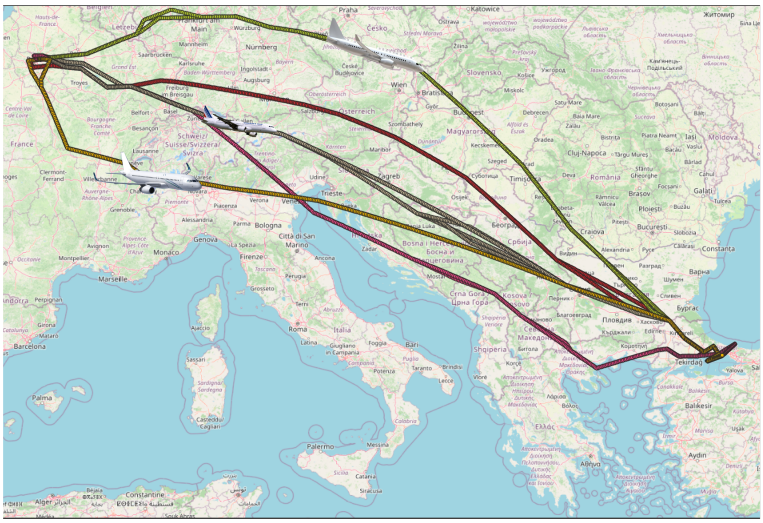


Figure 1.3: Trajectory modeling of long-distance flights.

It is apparent that congestion problems demand the development of effective solutions to optimize resource allocation, enhance efficiency, and minimize disruptions. In

cases of autonomous vehicles (air, land, or water), accurate trajectory prediction plays a key role in congestion management. By designing generative models that allow the construction of future plans with trajectories, decision-makers can proactively identify potential congested areas, mitigate bottlenecks, and optimize resource allocation. Trajectory modeling enables efficient routing, and coordination, thereby reducing delays, improving resource utilization, and enhancing the overall system performance.

Specifically in the aviation domain, one of the critical aspects of congestion management is to accurately predict aircraft trajectories. Modeling trajectories before the operation can reduce significantly the imposed delays, and optimize the airspace utilization. IL techniques offer a promising approach to tackle the trajectory prediction problem, by training intelligent agents to imitate expert pilot behaviors by observing them flying the aircraft from an origin to a destination airport. An important issue that must be taken into consideration is that in real-world applications the resulting trajectories usually present various behavioral patterns during execution. In this direction, multi-modal imitation learning can provide advantageous solutions since it not only tries to generate trajectories, but also identify the modalities of the demonstrated examples.

The aircraft trajectory prediction problem is challenging due to various factors such as airspace constraints, weather conditions, hidden information from the companies and the presence of multiple modalities for executing the same task. This leverages the motivation for studying this task and trying to offer improved solutions. Traditional trajectory prediction methods often rely on simplified mathematical models and heuristics, which may not be able to capture the complexity of real-world flights. IL schemes with modular multi-modal characteristics may overcome these barriers by leveraging expert data that cover a multitude of actual situations.

Using modular multi-modal imitation learning in aircraft trajectory prediction can help the modeling process due to the complex and diverse nature of aircraft trajectories. Thus, we can capture and model the distinct patterns and behaviors exhibited by different clusters of trajectories, in different flight phases. Also, the accuracy and robustness of aircraft trajectory prediction systems can be significantly improved. This, in turn, can enhance air traffic management, optimize route planning, and contribute to safer and more efficient flights.

1.3 Thesis contributions

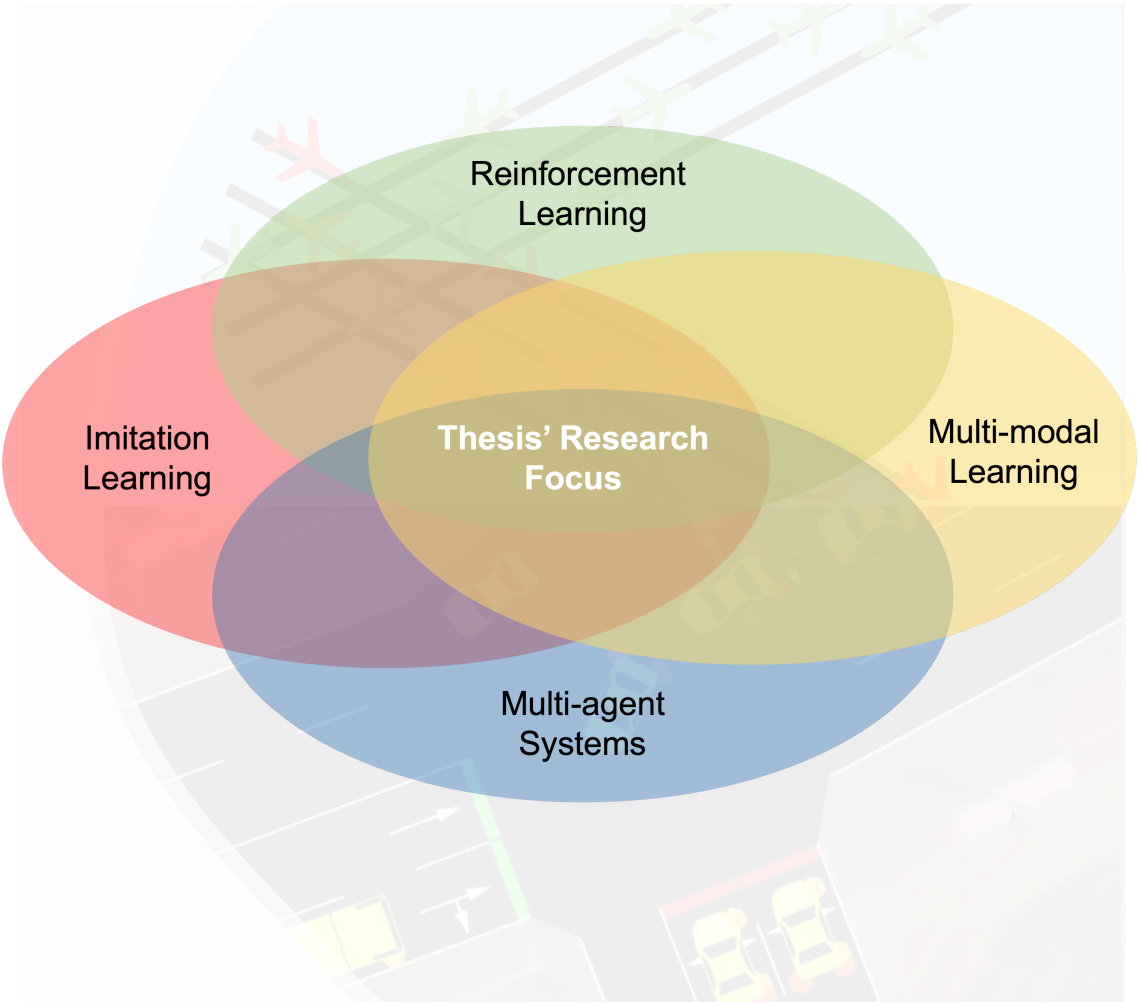


Figure 1.4: The research focus of this dissertation in relation to other fields.

In light of the above, this thesis lies in the intersection of the aforementioned fields of research, as shown in the Venn diagram presented in Fig. 1.4. The primary aim is to focus on the study and implementation of innovative learning strategies and algorithmic RL-based methodologies in order to develop advanced and robust intelligent entities, designed as either single-agent or multi-agent systems, which are aimed at addressing the complex challenges that arise in traffic applications. The research direction involves analyzing and understanding the dynamics of traffic systems, including factors such as traffic flow, congestion patterns, and the behavior of individual entities. These intelligent agents will possess the capability to adapt and make informed decisions to optimize traffic flow, reduce congestion, improve transportation efficiency, and finally enhance overall human safety and create more

sustainable transportation networks. In addition, this dissertation aims to be a significant step towards addressing complex problems that arise in traffic domains and mostly making novel RL algorithms applicable to many real world traffic applications.

More specifically, the focus is on developing reinforcement learning algorithms for intelligent agents that deal with the following problems:

- Urban traffic navigation
- Air traffic management
- Aircraft trajectory prediction

The first two problems concern the resolution of congestion problems in their respective domains. Specifically, in the urban environment we deal with the navigation of autonomous vehicles in unsignalized roads, and in the aviation domain we tackle the problem of demand-capacity in the utilization of airspace usage. In the third problem, we focus on intelligent agents capable of imitating historical expert flight data, in order to predict the future evolution of new trajectories.

The contributions of this thesis can be divided into three subsections addressing contributions to methodology and the general application area they belong to.

❒ Methodological contributions

✔ It adopts a methodological approach and a modeling perspective that are consistently applied to address the various problems encountered, ensuring the development of robust and effective solutions. Given the inherent complexity of each problem, the primary objective throughout the thesis was to establish a structured framework for problem-solving and enable a systematic analysis of the issues at hand. Regardless of the specific problems under consideration, significant efforts were initially dedicated to capturing the fundamental elements of each problem and establishing a clear and coherent structure. This approach facilitated thorough analysis and experimentation, ultimately leading to the generation of reliable and meaningful results. By employing these approaches, the proposed methodologies were rigorously and systematically executed, thereby minimizing potential bias and maximizing the potential for discovering innovative and practical solutions.

- ✔ In all the research problems addressed in this thesis, large-scale big-data have been employed to train the intelligent agents. More specifically, for the aviation domain, real-world historical aircraft trajectories from flights over Europe have been utilized. Additionally, for the urban traffic management, real-world road networks and sensor data are obtained through the well-known SUMO (Simulator of Urban MObility) traffic simulation package, which is designed to handle large-scale traffic scenarios. The utilization of large amounts of data has allowed intelligent agents to acquire profound insights, make informed decisions, and adapt dynamically to various scenarios. One of the goals of this thesis was the integration of large-scale big data in training intelligent agents in order to facilitate the development of highly accurate predictive models. Furthermore, efficient learning algorithms and modelling schemes were used for learning agents from experiences and leverage this knowledge to anticipate future trends, make accurate forecasts, and mitigate potential risks.

❒ Contributions in the area of congestion management

- ✓ To coordinate multiple vehicles in large-scale unsignalized traffic networks, the notion of *route-agents* is introduced which alleviates the complexities and constraints associated with treating each vehicle as an independent agent. Much effort has been made to design appropriate intelligent agents with rich explanatory state spaces and well-defined reward functions. Deep MARL structures and learning schemes are described with the aim to guide and serve efficiently and safely vehicles in complex traffic scenarios.
- ✓ It encourages and supports research to allow learning agents of deep MARL structures to automatically *reuse knowledge* and allow knowledge generalization. Under this prism, learned agents are able to operate in unknown traffic scenarios with much higher volumes of traffic and duration and increased stochasticity, and simultaneously reach comparable performance level of that was achieved during training.
- ✓ A general-purpose multi-agent hierarchical reinforcement learning framework is introduced that addresses various levels of abstraction in state and action spaces, either individually or simultaneously. Being focused on a real-world congestion problem, a comprehensive experimental study is

described.

❒ Contributions in the area of trajectory modeling

- ✓ The challenging problem of aircraft trajectory modeling and prediction was addressed using a collection of real-world flight trajectories data. An apprenticeship learning framework is used that allows the construction of optimal policies that manage to imitate demonstrated trajectories. Leveraging historical data, the method is capable of learning to mimic a uni-modal expert behavior and generate trajectories close to the original. In addition, it adopts an informative agent state representation that encompasses a rich feature space that affects significantly the learning and imitation process.
- ✓ An efficient generative trajectory modeling scheme is proposed that combines both modular and multi-modal characteristics in an imitation learning structure. The method allows processing a large amount of data by initially splitting the data into modules that correspond to different flight phases. Then, multi-modal imitation learning algorithms are employed independently at each phase in order to predict the modality, and the evolution of the aircraft trajectory. Finally, all these components are connected to form full trajectories between origin-destination airport pairs.

1.4 Structure of the dissertation

The rest of this dissertation is organized as follows.

In **Chapter 2**, an overview of the preliminary concepts and techniques in ML is provided. This chapter serves as a foundation for understanding the subsequent chapters and their applications in the thesis' examined problems. The chapter begins by introducing the concept of ML and explores GANs which are powerful frameworks for training generative models. Then, it transitions to RL that is concerned with training intelligent agents to make sequential decisions, and markov decision process (MDP) which provide a mathematical framework for modeling sequential decision-making under uncertainty. Moreover, details are given about deep reinforcement learning (DRL), which combines RL with deep neural networks and prominent algorithms in that field, such as deep Q-networks (DQN), are being discussed. Finally, the chapter

concludes with a comprehensive analysis of IL, that leverages expert demonstrations to train models that imitate expert behaviors.

Chapter 3 explores the problem of autonomous navigation of multiple vehicles in traffic networks consisting of multiple urban roads with unsignalized intersections. To deal with it, the previous problem is formulated as a deep MARL framework, and the notion of *route-agents* is introduced as the building block of the MAS. A value function approximation scheme is constructed through a weighted combination of two neural networks that serves as the decision mechanism for the proposed route-agents. Experiments are conducted on SUMO simulator, which is a widely used traffic simulator that plays a crucial role in transportation research and congestion management.

Chapter 4 presents an extension of the research work of the previous chapter for navigating autonomous vehicles in large-scale real-world scenarios with unsignalized intersections. The problem is tackled by employing an advanced DRL method that leverages a rich state space and a well-defined reward function in order to enable *route-agents* to navigate their corresponding vehicles safely and rapidly to their destination. Furthermore, a very important discovery is that the learned policies of *route-agents* can be exploited and re-used in unknown scenarios through the concept of transfer learning. To assess the performance of the proposed method, large-scale urban road networks (e.g. center of Athens) are used, and the approach is compared with standard car-following models that are part of the SUMO simulator. Simultaneously, the generalization capabilities of the method are also tested according to the reuse of the learned policies in unknown stochastic environments.

Chapter 5 discusses the problem of *demand-capacity balance* in the air traffic management. This challenge is being faced at the pre-tactical stage of operations (i.e. before flights take-off) by imposing delays to flights that are predicted to participate in future congested air sectors. The aforementioned problem is formulated as a hierarchical multi-agent markov decision process (HMAMDP), and a hierarchical reinforcement learning (HRL) method is proposed towards resolving it. The experiments are conducted using a real-world dataset containing flights above Spain, from Barcelona to Madrid.

Chapter 6 introduces the trajectory prediction problem in the aviation domain and proposes a solution though the prism of IRL. Specifically, the framework of *apprenticeship learning* is utilized to leverage an expert dataset of historical flown tra-

jectories from Barcelona to Madrid. To train the method, an informative state space is created using RBF kernel functions for the features, and the training procedure shifted between an IRL step to extract the expert reward function and an RL step to solve the full MDP. The proposed method is tested on a real-world dataset and its performance is evaluated against the expert pilots.

Chapter 7 tackles the trajectory prediction problem by proposing a modular multi-modal IL framework, which begins by segmenting aircraft trajectories into sub-trajectories corresponding to flight phases. Afterwards, state-of-the-art multi-modal IL algorithms are employed to identify the modalities and learn a mixture of policies per flight phase, towards predicting the evolution of the aircraft trajectory. The efficiency of the proposed framework is evaluated in the trajectory modeling task using a real-world dataset consisting of flown trajectories between Paris and Istanbul.

Finally, **Chapter 8** summarizes this dissertation and draws directions for future research work.

Chapter 2

Preliminaries

This chapter discusses the core concepts that form the foundation of this thesis and provides a comprehensive overview of several fundamental topics. It begins by introducing the basics of each concept, laying a solid ground for understanding their principles and applications. As it progresses, it explores step-by-step more advanced methodologies associated with these topics, ensuring a thorough comprehension of these complex techniques and their relevance to the thesis.

2.1 Generative adversarial modeling

2.1.1 Game theory

The inspiration of Generative Adversarial Network (GAN) comes from the zero-sum games in game theory [2]. Game theory is a branch of mathematics and economics that studies the strategic decision-making process in competitive situations. A zero-sum game is defined as a non-cooperative game in which two parties are strictly opposed to each other, where the gains of one party are bound to bring losses of the

other party, and the gains and losses of both parties add up to zero [3]. Examples of zero-sum games include chess, where a win for one player translates to a loss for the other. The study of zero-sum games helps shed light on strategies, equilibria, and optimal decision-making in competitive scenarios. In the field of generative learning, game theory can be applied to develop adversarial frameworks. For instance, GANs leverage the competitive dynamics between a generator and a discriminator to learn the underlying data distribution and generate new samples. The insights from game theory contribute to the design and training of generative models, leading to advancements in artificial intelligence and machine learning.

2.1.2 Artificial neural networks

The artificial neural networks, also known as *neural networks* (NN), are inspired by the human brain and developed in order to imitate the way that the biological neurons communicate with each other [4]. NNs consist of interconnected computational units called neurons, organized into layers. The information flows through these layers, with each neuron performing a weighted sum of its inputs, applying an activation function, and passing the output to the next layer.

A popular type of NN is the *multi-layer perceptron* (MLP) [5] which is a fully connected feed-forward network that consists of an input layer, one or more hidden layers, and an output layer. Each layer comprises multiple neurons, which are connected to other neurons in the network with weighted edges.

One of the key concepts of MLPs is the *activation function*, which introduces non-linearity into the network and allows the model to learn complex relationships about the data. Some of the most popular activation functions used in MLPs include the sigmoid function, hyperbolic tangent (tanh) function, and rectified linear unit (ReLU) function.

Typically, to train an MLP, the *backpropagation* algorithm is employed, which fine-tunes the weights of the network based on the error between the predicted and actual outputs, aiming to minimize a *loss function*. In addition to backpropagation, various *optimizers* have been proposed to update the network's weights, such as stochastic gradient descent (SGD), Adam [6], and RMSprop.

Training a ML model typically involves partitioning the dataset into training, validation and test sets. The training data are used to update the network's parameters,

while the validation set is used to monitor the model's performance and prevent unwanted situations (e.g. overfitting). Moreover, regularization techniques like L_1 or L_2 regularization, dropout, data augmentation, and early stopping can be employed to improve generalization.

MLPs have achieved remarkable results in a wide range of applications, including image and speech recognition, natural language processing, and time series analysis. Their ability to model complex relationships, and learn from the given data has made them the back-bone of the field of deep learning (DL).

2.1.3 Generative models

Autoregressive networks

Autoregressive networks is a generative modeling approach that focuses on generating data sequentially. This type of models aims to capture the conditional probability distribution of each data element given its predecessors. In other words, auto-regressive generative models define a distribution over sequences using the chain rule for conditional probability, where in each step the next sequence element is predicted given the previous elements. This allows the model to progressively generate the entire sequence. Well-known examples of auto-regressive models are the recurrent neural networks (RNN), the long short-term memory (LSTM) networks and the transformers.

Autoencoders

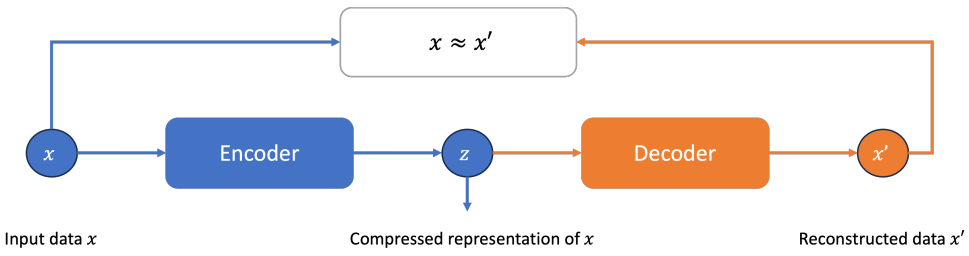


Figure 2.1: The architecture of autoencoder.

Autoencoder (AE) [7] is a popular type of generative model that takes high-dimensional data and compresses them into a small representation using neural network structures. It can be seen as a compression algorithm, in which the data

compression and decompression are realized by neural network self-learning. Any AE contains two types of networks: an encoder and a decoder. The encoder consists of a bunch of layers that take the input data and compress it down to a small demonstration, which has fewer dimensions. This low (or compressed) demonstration of input data is called a *bottleneck*. The decoder takes that bottleneck and tries to reconstruct the input data (see Figure 2.1). The AE calculates the reconstruction loss between encoder input and decoder output and its objective function is mathematically defined as:

$$L_{AE} = \frac{1}{n}\sum_{i}[x_i - f_{(\theta)}(g_{(\phi)}(x_i))]^2 \tag{2.1}$$

where $g_{(\phi)}$ represents the encoder network, $f_{(\theta)}$ represents the decoder network, x_i represents the input data, ϕ, and θ represents the network parameters.

Variational Autoencoders (VAEs)

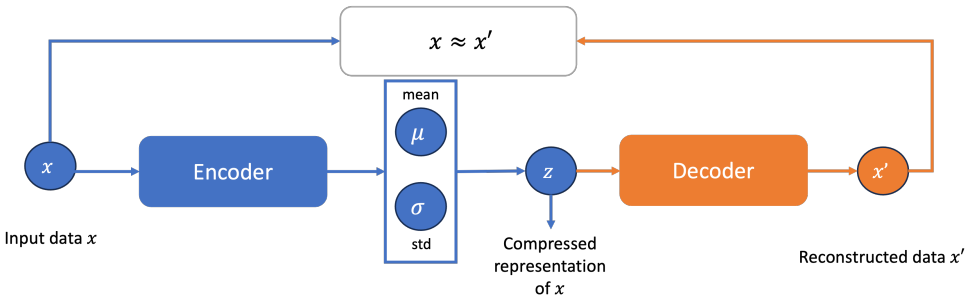


Figure 2.2: The architecture of variational autoencoder.

Variational Autoencoder (VAE) [8] is another widely used likelihood-based generative model. It includes a probabilistic encoder network (parameterized by ϕ), a probabilistic decoder (or generative) network (parameterized by θ) and loss functions. The probabilistic encoder $q_0(z|x)$ (also called latent variable generative model) embeds a data sample x into discrete latent variables, denoted by z, and the probabilistic decoder network $p_\theta(z|x)$ reconstructs the input sample based on the discrete latent vector z, without massive input data loss (see Figure 2.2).The cost function of the VAE is defined as:

$$L_{VAE} = E_{q_0(z|x)}[logp_\theta(xz)] - DKL[(q_0(zx)||p_\theta(z)] \tag{2.2}$$

where q_0 and p_θ represent the parameterize distributions for VAE probabilistic encoder-decoder.

2.1.4 Generative adversarial networks

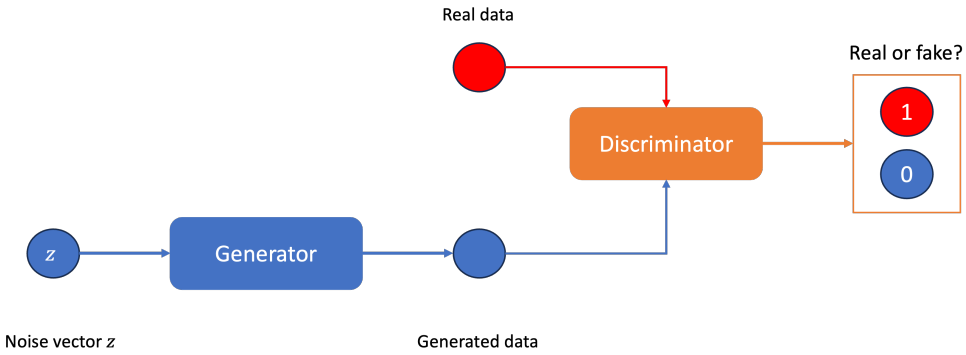


Figure 2.3: The architecture of generative adversarial networks.

GANs [9] (Fig. 2.3) are a class of DL models that build upon the foundation of MLPs and have revolutionized the field of generative modeling. They aim at estimating an unknown true distribution, denoted as p_r, by training a model on samples taken from that distribution. GANs are based on an architecture that consists of two NNs with competing objectives: a generator network G, and a discriminator network D.

The discriminator is a classifier that is trained to distinguish between samples drawn from the true distribution and samples generated by the generator. Its objective is to accurately identify if a given sample comes from the true distribution or not. On the other hand, the generator is trained to produce samples that would trick the discriminator into categorizing them as if they were originated from the true distribution. Hence, its goal is to generate realistic samples that are indistinguishable from real data.

The generator takes as input a noise vector $\mathbf{z} \sim p(z)$ drawn from a noisy prior distribution (e.g. Gaussian or uniform) and generates a sample $\mathbf{x} = G(\mathbf{z})$. The discriminator takes as input both generated samples from the generator's distribution, p_g, and real samples from the true distribution, p_r, and outputs a probability $D(\mathbf{x})$ that the sample $\mathbf{x}$ is real. The primary objective of the generator is to replicate the true data distribution (p_r) by generating data that closely resemble real data, while the discriminator is trained to differentiate between those data. Hence, the generator and the discriminator compete against each other in a min-max game which can be formulated as:

$$\min_G \max_D \mathbb{E}_{x \sim p_r}[logD(x)] - \mathbb{E}_{z \sim p_z}[log(1 - D(G_\theta(z)))] \tag{2.3}$$

The objective function of Eq. 2.3 is equivalent to minimizing the Jensen-Shannon (JS) divergence between p_r and p_g distributions. As each player in this min-max game independently optimizes its own objective function, the goal becomes to find a Nash equilibrium [10]. In game theory, a Nash equilibrium refers to a solution in a non-cooperative game involving two or more players, where no player can benefit by changing their strategy. In other words, it represents a stable state where each player's strategy is optimal given the strategies of the other players.

Training a GAN can be highly challenging in practice due to the nature of reaching a Nash equilibrium when dealing with non-convex objective functions and high-dimensional parameter spaces [11]. GANs commonly encounter two issues: vanishing gradients and mode collapse. *Vanishing gradients* refer to the problem where the gradients used to update the networks become extremely small, hindering the optimization process and the convergence of the training. *Mode collapse* is another common problem, where the generator fails to produce a diverse range of samples and instead focuses on only a limited set of outputs. For instance, when the task is to generate images with numbers, mode collapse could occur in case the generator consistently generates only few specific numbers, neglecting the rest.

Algorithm 1 summarizes the training process of GANs.

Algorithm 1: GAN algorithm

Input: Training data X, noise vector z, learning rate α, number of training iterations for generator (N_G) and discriminator (N_D)

Initialize generator G and discriminator D with random weights

for $i \leftarrow 1$ *to* N_G **do**

  for $j \leftarrow 1$ *to* N_D **do**

    Sample real data batch $x \sim X$

    Sample noise vector batch $z \sim \mathcal{N}(0, 1)$

    Generate fake data batch $G(z)$ using generator G

    Update discriminator D using gradient descent:

    $\theta_d \leftarrow \theta_d - \alpha \cdot \nabla_{\theta_d} \left[\frac{1}{m}\sum_{i=1}^{m} \log(D(x^{(i)})) + \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))\right]$

  Sample noise vector batch $z \sim \mathcal{N}(0, 1)$

  Update generator G using gradient descent:

  $\theta_g \leftarrow \theta_g - \alpha \cdot \nabla_{\theta_g} \left[\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))\right]$

2.1.5 Variants of GANs

Wasserstein GANs

Both vanishing gradients and mode collapse are obstacles that hinder the GAN model from capturing the full complexity and diversity of the real distribution. To address these issues, Arjovsky et al. [12] presented a variant of GANs, known as the Wasserstein GAN (WGAN). The key idea behind WGAN is to minimize the *Earth-Mover* or *Wasserstein distance* between the true distribution (p_r) and the generated distribution (p_g), replacing the JS divergence. This approach aims to overcome the issue of vanishing gradients by utilizing an objective function that is continuously differentiable and provides smoother gradient compared to the vanilla GAN objective. By minimizing the Wasserstein distance, WGAN encourages a more stable training process and can potentially improve the quality and diversity of the generated samples.

Intuitively, Wasserstein distance can be seen as the minimum work needed in order to transform one distribution into another. In this context, "work" refers to the product of the mass of the distribution that needs to be moved and the distance it needs to be moved. Moreover, it provides a measure of dissimilarity between two distributions by quantifying the effort required to redistribute the mass from one to match the other. Mathematically, it is formulated as:

$$D(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\| = \inf_{\gamma \in \Pi(p_r, p_g)} \sum_{(x,y)} \|x - y\| \gamma(x, y), \tag{2.4}$$

where $\Pi(p_r, p_g)$ denotes the set of all possible joint probability distributions with marginals p_r and p_g. However, the equation for the Wasserstein distance is highly intractable. By utilizing the Kantorovich-Rubinstein duality [13], Eq. 2.4 can be simplified as:

$$D(p_r, p_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)], \tag{2.5}$$

where the supremum is taken over all functions f that are *1-Lipschitz*. For a function to be *1-Lipschitz*, the following constraint must hold:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \tag{2.6}$$

Hence, to compute the Wasserstein distance between distributions p_r and p_g , we need to find the 1-Lipschitz function f that minimizes Eq. 2.5. An approximation of

f can be obtained by training a neural network with weights w. In the case of WGAN, the parameterized function f_w corresponds to the discriminator D_w. The discriminator used in WGAN - in contrast to vanilla GAN- does not have an activation function after its output layer, and its output is no longer a probability, but a scalar value that indicates "how real" a generated sample is.

Formally, the objective of WGAN is represented as a min-max game between discriminator and generator such as:

$$\min_{G} \max_{D \in \mathcal{F}} \mathbb{E}_{\tilde{x} \sim p_g}[D(\tilde{x})] - \mathbb{E}_{x \sim p_r}[D(x)], \tag{2.7}$$

where $\mathcal{F}$ is the set of 1-Lipschitz functions. To enforce that the discriminator is a 1-Lipschitz function, the original WGAN paper [12] introduces a simplistic way to apply the constraint. Specifically, it performs weight clipping to ensure that the weights w of the discriminator are within a range $[-c, c]$, where c is a hyperparameter. Algorithm 2 summarizes the training procedure of WGANs.

Algorithm 2: Wasserstein GAN algorithm

Input: Training data X, noise vector dimension z, learning rate α, number of training iterations for generator (N_G) and discriminator (N_D)

Initialize generator G and discriminator D with random weights

for $i \leftarrow 1$ *to* N_G **do**

  for $j \leftarrow 1$ *to* N_D **do**

    Sample real data batch $x \sim X$

    Sample noise vector batch $z \sim \mathcal{N}(0, 1)$

    Generate fake data batch $G(z)$ using generator G

    Update discriminator D using gradient ascent:

    $\theta_c \leftarrow \theta_c + \alpha \cdot \nabla_{\theta_c} \left[\frac{1}{m} \sum_{i=1}^{m} D(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} D(G(z^{(i)})) \right]$

    Clip weights of discriminator D in a range $[-c, c]$

  Sample noise vector batch $z \sim \mathcal{N}(0, 1)$

  Update generator G using gradient descent:

  $\theta_g \leftarrow \theta_g - \alpha \cdot \nabla_{\theta_g} \left[\frac{1}{m} \sum_{i=1}^{m} D(G(z^{(i)})) \right]$

WGAN with gradient penalty

Nevertheless, the authors of WGAN acknowledge that using weight clipping to enforce the Lipschitz constraint is not the optimal solution. They have observed that setting a large clipping parameter can make it challenging to achieve convergence, while a small clipping parameter may result in vanishing gradients.

Gulrajani et al. [14] introduced an extension to WGAN, called Wasserstein GAN with gradient penalty (WGAN-GP), which presents an alternative approach for enforcing the Lipschitz constraint on the discriminator, eliminating the weight clipping. A differentiable function is considered 1-Lipschitz if and only if the norm of its gradients is at most one (1) everywhere. In other words, it encourages the discriminator to be locally Lipschitz continuous by ensuring that the gradients are not too large.

Moreover, they show that interpolated points between the real and the generated data have a gradient norm of at most one (1). Hence, they modify the objective function of WGAN (Eq. 2.7), and instead of weight clipping, they penalize the model if the gradient norm of the weights of the discriminator deviates from one (1):

$$\min_{G} \max_{D \in \mathcal{F}} \mathbb{E}_{\tilde{x} \sim p_g}[D(\tilde{x})] - \mathbb{E}_{x \sim p_g}[D(x)] + \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2], \tag{2.8}$$

where λ is the gradient penalty coefficient. Moreover, $p_{\hat{x}}$ is the distribution of the interpolated data under $\hat{x} = \epsilon x + (1 - \epsilon)\tilde{x}$, for $x \sim p_r$ and $\tilde{x} \sim p_g$, where $\epsilon \in [0, 1]$. Algorithm 3 presents the training process of WGANs with gradient penalty.

Algorithm 3: Wasserstein GAN with gradient penalty algorithm

Input: Training data X, noise vector dimension z, learning rate α, number of training iterations for generator (N_G) and discriminator (N_D)

Initialize generator G and discriminator D with random weights

for $i \leftarrow 1$ *to* N_G **do**

  for $j \leftarrow 1$ *to* N_D **do**

    Sample real data batch $x \sim X$

    Sample noise vector batch $z \sim \mathcal{N}(0, 1)$

    Generate fake data batch $G(z)$ using generator G

    Compute interpolated samples $\hat{x}$ by sampling uniformly between x and $G(z)$

    Compute discriminator outputs for real and fake samples:

    $D(x) \leftarrow D(x) \quad D(G(z)) \leftarrow D(G(z))$

    Compute gradient penalty: $\hat{g} \leftarrow \nabla_{\hat{x}} D(\hat{x})$

    Compute penalty term $p \leftarrow \lambda \cdot (\|\hat{g}\|_2 - 1)^2$

    Update discriminator D using gradient ascent with penalty:

    $\theta_d \leftarrow \theta_d + \alpha \cdot \left[\frac{1}{m} \sum_{i=1}^{m} D(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} D(G(z^{(i)}))\right] + p$

  Sample noise vector batch $z \sim \mathcal{N}(0, 1)$

  Update generator G using gradient descent:

  $\theta_g \leftarrow \theta_g - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} D(G(z^{(i)}))$

InfoGAN

InfoGAN [15] is another interesting variant of GANs that provides additional latent information into the generator. Its main objective is to learn to disentangle and interpret representations of data. In vanilla GANs, the generator maps a random noise vector to generated samples, however, this latent space lacks any explicit structure or semantic meaning. On the other hand, InfoGAN addresses this limitation by maximizing the mutual information between a fixed small subset of the GAN's noise variables and the observations.

To achieve this, InfoGAN introduces a "latent code" in the input of the generator. The latent code captures specific attributes or characteristics of the generated samples. For example, in the case of face image generation, the latent code could represent attributes like facial expressions, glasses, or hair color.

The generator G takes as input both a noise vector z and a latent code c and produces a generated sample x. The discriminator D, on the other hand, aims to distinguish between real samples x and generated samples $G(z, c)$. The objective remains the same as in vanilla GANs, i.e. to simultaneously train G and D with competing objectives so as G generates samples that can trick D.

In addition to the vanilla GAN's loss, InfoGAN introduces a term that maximizes the mutual information between the latent code c and the generated sample x. This is done to encourage the generator to learn meaningful representations in the latent space. Formally, the mutual information $I(c; G(z, c))$ is maximized by minimizing the following objective:

$$\min_G \max_D \mathbb{E}_{x\sim p_r}[logD(x)] - \mathbb{E}_{z\sim p_z}[log(1 - D(G_\theta(z, c)))] - \lambda I(c; G_\theta(z, c)), \tag{2.9}$$

where λ is a hyperparameter that weights the contribution of the mutual information term in the objective of GAN.

However, in practice, maximizing directly the mutual information $I(c; G(z, c))$ is very hard, as it demands access to the posterior $P(c|x)$. Instead, InfoGAN employs an auxiliary network Q in order to estimate it. The Q network takes as input the generated sample x and predicts the latent code c. By minimizing the divergence between the true distribution of c and $Q(c|x)$, a lower bound of the mutual information can be obtained.

By incorporating latent codes, InfoGAN enables the disentanglement of specific attributes. This allows for better control over the generated samples by manipulating

the values of the latent code, and makes InfoGAN applicable in various domains, including image editing, style transfer, and data augmentation. Furthermore, InfoGAN's ability to learn without labeled data makes it valuable in scenarios where labeled examples are limited or expensive to obtain.

2.1.6 Diffusion models

Unlike GANs, diffusion models [16] do not depend on adversarial processes to generate outputs. Diffusion models generate data based on non-equilibrium thermodynamics. To generate synthetic samples mimicking real ones, diffusion models rely on the inversion of an additive noise process. The model takes as input a noisy image composed of white noise and image content, and generates progressively less noisy versions of it until reaching the desired noiseless output

More specifically, diffusion models are composed of two separate stages: the forward and reverse diffusion processes. The forward diffusion process is responsible for the addition of Gaussian noise to a given sample, while the backward diffusion process is the reconstruction of a sample from a noisier sample. Diffusion models operate on a series of time steps, where an increased time step indicates another addition of Gaussian noise. This process is treated as a Markov Chain, where the sample at time step t only depends on the sample from time step $t-1$. The forward process is fixed, however, the model attempts to learn the necessary operations to perform on a given sample at time step t to reconstruct the sample at $t-1$. Once training is complete, the model should be able to generate a sample similar to those within the original distribution from complete Gaussian noise.

2.2 Reinforcement learning

Reinforcement learning (RL) [17] is a field of the ML that focuses on training agents to learn by interacting with their environment though a trial-and-error process. It offers an alternative to supervised learning without using annotated data, but only a reward function that indicates good or bad behavior. Learning can take longer since typically no demonstrations of good behavior are given and the environment is completely unknown.

RL agent is an entity of a stochastic environment that uses sensors to perceive its

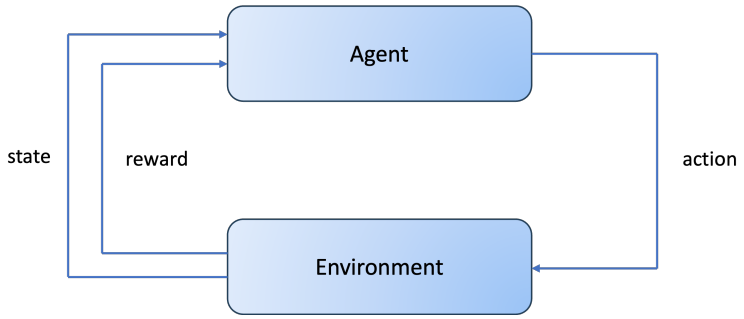


Figure 2.4: The basic reinforcement learning scheme.

state and takes appropriate actions. As a result, these actions along with the dynamics of the environment, cause a transition into a new state. The quality of the transitions is being evaluated by a scalar reward, and the goal of the agent is to maximize the cumulative reward. A general scheme of the interaction between the agent and the environment is depicted in Fig. 2.4.

2.2.1 Markov decision process

Markov Decision Processes (MDPs) are mathematical models that represent decision-making problems under uncertainty, while maintaining the fundamental Markovian property. This property states that the future state depends only on the current state and action, and is independent of the history of states and actions that led to the current state.

Typically, an MDP can be defined by a $5-$tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$:

- $\mathcal{S}$ is the state space, which can be either discrete or continuous
- $\mathcal{A}$ is the action space, which can be either discrete or continuous
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, which defines a probability distribution over reaching the next state $s' \in \mathcal{S}$ given the current state $s \in \mathcal{S}$ and the executed action $a \in \mathcal{A}$
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which yields a scalar value for the executed action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$
- $\gamma \in (0, 1)$ is a discount factor that weights future rewards

The rewards are short-term reinforcement signals, given as feedback after the agent has taken an action which led to the transition into a new state. Summing all future rewards and discounting them would lead to the *return*, G:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \tag{2.10}$$

The decision mechanism of an agent is guided by the *policy*, π, which defines a conditional distribution $\mathbb{P}(a \in A | s \in \mathcal{S})$ and can be either deterministic or stochastic. In a deterministic policy, $\pi : \mathcal{S} \to \mathcal{A}$, states are mapped directly to actions and $\pi(s)$ denotes the action to be taken in state s. On the other hand, a stochastic policy, $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, maps states to action selection probabilities.

One of the key concepts in RL is the use of *value functions* to represent the expected cumulative reward an agent can receive from a given state or state-action. The *state value function*, $V(s)$, is a measurement of how good it is for the agent to be in state s under the current policy π. It is defined as the expected cumulative reward obtained if the agent starts from state s and follows its policy, π, thereafter:

$$V^\pi(s) = \mathbb{E}^\pi \left[G_t | S_t = s \right] = \mathbb{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} | S_t = s \right] \tag{2.11}$$

Another important concept is the *action value function*, or *Q-function*, $Q^\pi(s, a)$, that defines the expected cumulative reward starting from state s, executing action a and continuing according to the policy π:

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} | S_t = s, A_t = a \right] \tag{2.12}$$

According to *Bellman equations* [18], the value function can be decomposed into two parts: the immediate reward plus the discounted future value function. If the reward and the transition functions are known, then Eq. 2.11, 2.12 can be re-written as:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V^\pi(s_{t+1}) | S_t = s] \end{aligned} \tag{2.13}$$

$$Q^{\pi}(s,a) = \mathbb{E}[R_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | S_t = s, A_t = a] \tag{2.14}$$

At this point, the ultimate goal can be defined as finding an optimal state-value function:

$$V^*(s) = \max_{\pi} Q_{\pi}(s) \tag{2.15}$$

or, more importantly, an optimal action-value function:

$$Q^*(s) = \max_{\pi} Q_{\pi}(s) \tag{2.16}$$

For this to be possible, a partial ordering over policies and value functions should be defined:

$$\pi \geq \pi' \iff V_{\pi}(s) \geq V'_{\pi}(s), \forall s \in S \tag{2.17}$$

According to [17], for any MDP there exists an optimal policy π^* that is better than or equal to all other policies, i.e. $\exists \pi^* : \pi^* \geq \pi, \forall \pi$.

In case that the values for all the state-action pairs of an MDP are known, then we could simply pick the policy which yields the highest value for all states and actions. To obtain the optimal policy, $\pi^*(s,a)$, we could assign probability one (1) for the action that has the maximum value for Q^* and zero (0) for every other action as:

$$\pi^*(a|s) = \begin{cases} 1 \text{ if } a = \arg\max_{a \in A} Q^*(s,a) \\ 0 \text{ otherwise} \end{cases} \tag{2.18}$$

2.2.2 Q-learning

Q-learning [19] is a model-free RL algorithm that allows an agent to learn an optimal policy in a MDP through trial and error.

In the aforementioned MDP framework, Q-learning aims to learn the optimal action-value function $Q(s,a)$. The Q-learning algorithm uses a table, often referred to as a Q-table, to store and update the estimated values of $Q(s,a)$ for each state-action pair (s,a). Initially, the Q-table is initialized with arbitrary values or zeros. The agent then interacts with the environment, observing states, taking actions, receiving rewards, and transitioning to the new states.

The Q-learning update rule is based on the Bellman equation, which states that the optimal value of $Q(s,a)$ is equal to the immediate reward plus the discounted

value of the best action to take in the next state. The update rule can be written as follows:

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \eta \left[r_t + \gamma \max_a Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t) \right], \tag{2.19}$$

where η is the learning rate that weights the contribution of new knowledge compared to the existing estimation, and the max operator represents the maximum value of Q for all possible actions in the next state.

By repeatedly applying the Q-learning update rule while interacting with the environment, the Q-table gradually converges towards the optimal values of $Q^*(s, a)$ for each state-action pair. Once the agent has learned the optimal Q-values, it can construct an optimal policy by choosing the action with the highest Q-value in each state.

2.2.3 Deep reinforcement Learning

Q-learning, in its traditional form, becomes inapplicable to complex state spaces due to the curse of dimensionality and its inability to efficiently explore and represent high-dimensional spaces. As the number of states increases exponentially, the tabular representation of Q-values becomes infeasible.

Nowadays, deep reinforcement learning (DRL) techniques are utilized in cases of continuous and/or complex state spaces, so as to model efficiently the Q-function. In these settings, the Q-function is considered as a parameterized function, $Q(s, a; \theta)$, where θ denotes the set of unknown model parameters that must be tuned.

Neural networks constitute an architecture for modeling complex functions and can be used in order to estimate the Q-function. These type of networks, called *Deep Q-Networks* (DQN) [20], provide a mechanism that successfully combines the Q-learning with the use of deep neural networks. The parameters θ of the function approximator can be found using optimizers by minimizing the temporal-difference (TD) loss at every time step t:

$$L_t(\theta_i) = \mathbb{E}(y_t - Q(s_t, a_t; \theta_i))^2 \tag{2.20}$$

$$y_t = r_t + \gamma \max_a Q(s_{t+1}, a; \theta_{i-1}) \tag{2.21}$$

In Equation 2.20, y_t is the TD target, $y_t - Q(s_t, a_t; \theta_{i-1})$ refers to the TD error and θ_i, θ_{i-1} are the network parameters at iteration i and $i-1$ respectively. The TD target is obtained by keeping the previous network parameters, θ_{i-1}, fixed.

The DQN algorithm is a model-free approach as it directly tackles the problem without explicitly knowing the dynamics of the environment. Additionally, DQN is an off-policy method as it learns a greedy policy, while utilizing an ϵ-greedy scheme to ensure sufficient exploration of the state space. However, one drawback of employing a neural network approximator is its training instability in the context of reinforcement learning. This instability arises from two the correlations among consecutive observations and the sensitivity of the policy to small changes in Q-values.

To improve stability and prevent the network from chasing a moving TD target, DQN introduces a separate *target network* with parameters θ'. This network is a copy of the main network, and its parameters are updated periodically with the weights of the main network. The target Q-values in Eq. 2.21 are computed using the target network, providing a more consistent target for the DQN update.

DQN also incorporates an *experience replay*, where past experiences $E_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a replay memory. To update the parameters of the DQN, a mini-batch of these tuples is randomly sampled from the replay memory. This approach ensures that the neural network is not trained on consecutive observations, which helps to avoid strong correlations between samples and reduces variance between updates.

By combining deep neural networks, experience replay, and target networks, DQN has achieved remarkable success in various domains. Its ability to handle high-dimensional state spaces and learn from raw sensory inputs has made DQN a significant advancement in the field of RL.

Double deep Q-network

While in many tasks DQN algorithm achieves human-level performance, there are still occasions that it performs poorly. The primary reason for the under-performance is its tendency to overestimate the Q-values. This overestimation arises from the positive bias caused by the max operator used in Q-learning and DQN update rules (Eq. 2.19, 2.20), which yields the maximum Q-value as an approximation of the expected optimal Q-value.

In order to address this problem, Hasselt [21] proposed the Double Q-Learning framework, which decouples the evaluation and selection process by adopting a dou-

ble estimator approach. This approach involves two Q-functions, Q^A and Q^B, where Q^A is updated with values from Q^B, while Q^B is updated with values from Q^A. In Double Q-Learning the update rule for the Q-values (Eq. 2.19) becomes:

$$Q^A(s_t, a_t) = Q^A(s_t, a_t) + \eta \left[r_t + \gamma Q^B(s_{t+1}, \arg\max_{a_t} Q^A(s_{t+1}, a_t)) - Q^A(s_t, a_t) \right] \quad (2.22)$$

$$Q^B(s_t, a_t) = Q^B(s_t, a_t) + \eta \left[r_t + \gamma Q^A(s_{t+1}, \arg\max_{a_t} Q^B(s_{t+1}, a_t)) - Q^B(s_t, a_t) \right] \quad (2.23)$$

An improved version of DQN in combination with double Q-learning, called *Double Deep Q-Network* (DDQN) [22], uses two identical neural networks in order to prevent the model from overestimating the Q-values. The idea behind DDQN is similar to that of double Q-learning, i.e. to decompose the max operation in the target into action selection and action evaluation. The two networks used in DDQN scheme are described as: the evaluation (online) network (θ_i) that evaluates the greedy policy, and the target network ($\hat{\theta}_i$) that is used to estimate its value. Following this architecture, the target value is replaced with the weights of the target network $\hat{\theta}_i$ in the update rule:

$$\max_a Q^\pi(s_{t+1}, a_t; \theta_i) \longrightarrow Q^\pi(s_{t+1}, \arg\max_{a_t} Q^\pi(s_{t+1}, a_t; \theta_i); \hat{\theta}_i) \quad (2.24)$$

As a result, the update rule for DDQN is defined as:

$$y_t = r_t + \gamma Q(s_{t+1}, \arg\max_{a_t} Q(s_{t+1}, a_t; \theta_i); \hat{\theta}_i) \quad (2.25)$$

Instead of introducing an additional network, the target network in DDQN is the perfect candidate to be utilized as the second Q-function approximator. This means that the weights from the i-th iteration are employed to evaluate the greedy policy, while the weights from the previous iteration are utilized to estimate its value. The update rule in DDQN remains unchanged, with the modification being the adjustment of the target as follows:

$$y_t = r_t + \gamma Q(s_{t+1}, \arg\max_{a_t} Q(s_{t+1}, a_t; \theta_i); \theta_{i-1}) \quad (2.26)$$

Finally, it is important to note that in both DQN and DDQN, the target network relies on the parameters from the previous iteration ($i-1$). However, for generalization

purposes, the target network can utilize parameters from any previous iteration ($i-k$). This allows flexibility in choosing which past iteration's parameters to use for the target network. Additionally, the parameters of the target network are periodically updated by making copies of the parameters from the online (main) network. This ensures that the target network remains synchronized with the latest updates of the online network throughout the training process.

2.2.4 Hierarchical reinforcement learning

Hierarchical reinforcement learning (HRL) extends the RL framework by introducing a hierarchical structure that decomposes tasks into subtasks, allowing for more efficient and scalable decision-making. Hierarchical markov decision process (HMDP) constitutes an extension of the traditional MDP framework that addresses the challenge of managing complex decision-making problems by decomposing them into multiple levels of abstraction.

In HRL, the environment is organized into a hierarchy of tasks, where each task represents a specific objective to be achieved. The tasks are arranged in a tree-like structure, with higher-level tasks representing more abstract and long-term goals, and lower-level tasks representing more specific and short-term objectives. The actions taken at each level influence the progression towards achieving the goals at higher levels.

Mathematically, a HMDP can be represented as a 7-tuple $\mathcal{M} = \{\mathcal{L}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \phi, \gamma\}$, where:

- $\mathcal{L}$ is a set of abstraction levels
- $\mathcal{S}^l$ is the state space at abstract level l (state abstraction)
- $\mathcal{A}^l$ is the action space at abstract level l (temporal abstraction)
- $\mathcal{T}_l : \mathcal{S}^l \times \mathcal{A}^l \to \mathcal{S}^l$ is the transition function for the abstract level l, which defines a probability distribution over reaching the next state $s' \in \mathcal{S}^l$ given the current state $s \in \mathcal{S}^l$ and the executed action $a \in \mathcal{A}^l$
- $\mathcal{R}_l : \mathcal{S}^l \times \mathcal{A}^l \to \mathbb{R}$ is the reward function for the abstraction level l, which yields a scalar value for the executed action $a \in \mathcal{A}^l$ at state $s \in \mathcal{S}^l$

- ϕ is an abstraction function that maps states of abstract level l to the states of the previous abstract level
- $\gamma \in (0, 1)$ is a discount factor that weights future rewards

The hierarchical structure introduces two types of policies: high-level policies and low-level policies. A high-level policy operates at the higher level of the hierarchy and selects subtasks to be executed. A low-level policy operates within a subtask and selects actions to execute based on the current state. In other words, the high-level policy determines the subtask, and the low-level policy selects actions to accomplish that subtask. The interaction between the high and low levels enables the agent to make decisions at different levels of granularity, allowing for more efficient exploration and exploitation of the environment.

2.2.5 Multi-agent reinforcement learning

RL has been successful in single-agent scenarios, however many real-world problems involve multiple agents that interact and influence other agents. This is where multi-agent reinforcement learning (MARL) comes into play. MARL extends RL to settings with multiple interacting agents, enabling them to learn how to cooperate, compete, or negotiate in complex environments.

A multi-agent markov decision process (MAMDP) is a framework used to model decision-making problems involving multiple autonomous agents in a shared environment. It extends the concept of traditional MDP, where a single agent interacts with an environment and makes decisions to maximize its own expected rewards, to a setting where multiple agents coexist and interact with each other and the environment. In MARL, the agents interact with the environment simultaneously and their actions may have interdependencies and influence the state transitions and rewards for other agents. The goal of each agent in a MARL system is to maximize its own and global expected cumulative rewards, taking into account the actions and policies of other agents.

Formally, a MAMDP can be described as a 6-tuple $\mathcal{M} = \{\mathcal{AG}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$ which comprises the following constituents:

- $\mathcal{AG}$ is a society of agents $A_i, i = 1 \dots N$

- $\mathcal{S}$ is the state space shared by all agents. Each agent may has observe its local state s_i or the shared state $\mathbf{s}$
- $\mathcal{A}_i$ is the set of actions of each agent A_i. The local action of each agent is denoted as a_i, and the global action as $\mathbf{a} = a_1 \times \cdots \times a_N$
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, which defines a probability distribution over reaching the next state $\mathbf{s}' \in \mathcal{S}$ given the current state $\mathbf{s} \in \mathcal{S}$ and the executed joint action $\mathbf{a} \in \mathbf{A}$
- $\mathcal{R}_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function for each agent A_i, which yields a scalar value for the executed action $a_i \in \mathcal{A}$ at state $s_i \in \mathcal{S}$
- $\gamma \in (0, 1)$ is a discount factor that weights future rewards

Ultimately, MARL serves as a framework that enables agents to make simultaneous transitions in a multi-agent environment. It can be sequentially described as follows: At each time step, every agent A_i observes a local state s_i (or the shared state $\mathbf{s}$), and executes its action a_i, at the same time with all the other agents. The joint action $\mathbf{a}$ from all agents causes the transition to a new state $\mathbf{s}' \sim P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and the environment yields a reward $R_i(s_i, a_i)$ to each agent. As in the single-agent MDP case, the goal of every agent is to solve the MAMDP, which can be achieved by finding a policy, $\pi_i \in \Pi_i : \mathcal{S} \to \mathcal{A}_i$, that selects optimal sequential actions for the agent in order to maximize its discounted expected cumulative reward.

2.3 Imitation learning

A primary obstacle in utilizing traditional RL approaches is the necessity of creating a well-defined description of the task in hand. The formulation of RL problem under the MDP framework requires the definition of a reward function that evaluates the "goodness" of the states and actions. Defining this reward function can be challenging, as it is often difficult to determine what constitutes an optimal reward for the task.

Imitation learning (IL) is a valuable technique when it becomes difficult to manually design a reward function, but there are available demonstrations from expert(s) performing the desired task. In this approach, an agent tries to behave optimally

by mimicking the expert demonstrations. There are three primary subfields of imitation learning: behavioral cloning (BC), inverse reinforcement learning (IRL) and adversarial imitation learning (AIL).

2.3.1 Behavioral cloning

BC [23, 24] is the simplest form of IL that involves the agent learning an expert policy through supervised learning. In this approach, expert demonstrations, consisting of state-action pairs, are utilized as samples, with the states serving as input and actions as target output.

The goal of BC is to learn a policy, π, which maps states s to actions a for each state-action pair in the dataset of the expert demonstrations. The policy can be approximated using a NN trained on the expert dataset by minimizing the mean squared error between expert and estimated actions.

While this method can provide solutions in some cases, it can also suffer from compounding errors that accumulate rapidly, causing the agent to diverge from its intended trajectory and end up in unexplored states where the expert's behavior is unknown.

2.3.2 Inverse reinforcement learning

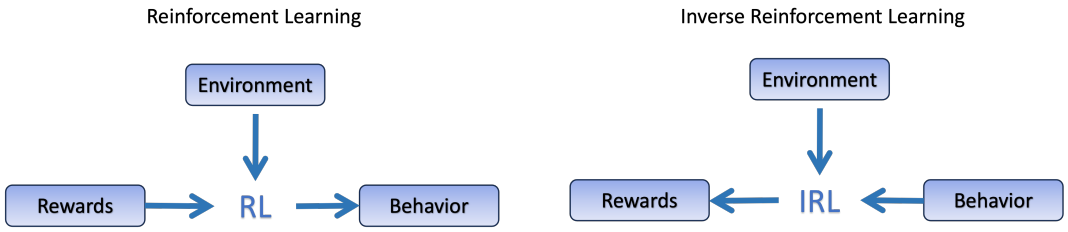


Figure 2.5: Differences between RL and IRL.

Rather than learning a mapping between expert states and actions through supervised learning, IRL offers a more general approach by trying to discover the underlying reward function, which explains the behavior of the experts [25]. Generally, the reward function specifies the goals of the experts and, as such, learning the reward function can be understood as learning the objectives of the experts towards solving the task in-hand. Once the reward function has been discovered, it can then

be applied to estimate the expert policy using RL approaches. Figure 2.5 depicts the differences in the procedures of RL and IRL.

Typically, the IRL problem is formulated as a MDP where the reward function is unknown, thus is denoted as MDP\R. The rest of the settings remain the same as the MDP definition of section 2.2.1.

In the context of IRL, it is presumed that the experts act in accordance with an underlying expert policy π_E, while the agent adheres to a stochastic policy π_θ that is parameterized by weights θ. The agent observes the experts' behavior in the form of trajectories given a dataset of expert demonstrations, $T_E = (T_i)_i^N$, where each T_i is an expert trajectory. IRL's objective is to find an estimate of the reward function that can most effectively explain the experts' observed behavior.

In general, IRL algorithms discover the unknown reward function via an iterative learning process which switches between two phases: (1) estimating the reward function, and (2) solving an RL problem using that reward function. Specifically, the MDP\R is initially solved to acquire the reward function, and then the full MDP is solved using this estimation of the reward. This process is repeated until the agent's policy that can best explain the expert demonstrations is discovered. However, IRL's problem definition is "ill-posed", because multiple reward functions can explain the same expert behavior.

Feature expectation matching

In *feature expectation matching*, the reward function can be modeled using any approximator (linear, NNs, and so on). In the simplest case, it is assumed to be a linear model over the feature vector of states, $\phi(s)$, such as:

$$R_w(s,a) = \mathbf{w}^T\phi(s,a) = w_1\phi_1(s,a) + w_2\phi_2(s,a) + \cdots + w_n\phi_n(s,a), \tag{2.27}$$

where $\mathbf{w}$ is a vector of linear coefficients of size equal to the dimensions of the feature space. The IRL process is focused on determining the proper weight vector in order to shape the expert's behavior.

Given the definition of value function in Eq. 2.11 and the reward function in Eq. 2.27, the value of a policy can be re-formulated as:

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_w(s_t, a_t)\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s_t, a_t)\right] \\
&= w^T \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t, a_t)\right]
\end{aligned}
\tag{2.28}
$$

The expert trajectories can be seen as walks through the state space made by the experts. These are guided by an expert ("true") reward function, $R^*(s) = w^{*T}\phi(s,a)$, assumed linear on $\phi(s,a)$. In order to estimate the linear weights, one can use the *feature expectations* [25], which are the expected discounted accumulated feature values:

$$
\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t, a_t)\right] \tag{2.29}
$$

The feature expectations can be seen as a representation of the policy and they can be utilized to compute the similarity between a generated policy and the expert policy. The expectation $\mathbb{E}[\cdot]$ is taken by sampling random state trajectories according to the current policy, π. On the other hand, the experts' feature expectations μ_E is calculated by its empirical estimate over the available M expert trajectories of the training set:

$$
\mu(\pi_E) = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^i, a_t^i) \tag{2.30}
$$

Abbeel et al. [25], showed that the linear weights **w** can be obtained by minimizing the distance between vectors of feature expectations $\mu(\pi)$ and $\mu(\pi_E)$. Hence, for a given threshold ϵ, the IRL problem becomes finding a policy π_θ that satisfies the following inequality:

$$
\|\mu(\pi_E) - \mu(\pi_\theta)\|_2 \leq \epsilon \tag{2.31}
$$

Maximum entropy IRL

Even though matching the feature expectations results into finding a policy π_θ that is similar to the expert policy π_E, there can still exist multiple solutions that explain the same expert behavior. Ziebart et al. [26] proposed the *maximum entropy IRL* approach by introducing an additional constraint to discover the best solution. The principle of maximum entropy [27] states that in the absence of prior knowledge, the most

unbiased probability distribution is the one with the maximum entropy. Applied to IRL, this principle encourages policies that explore a wide range of actions and states, avoiding excessive bias towards specific behaviors. Generally, the entropy of a policy is a measure of its randomness or uncertainty and is defined as:

$$H(\pi_\theta) = -\sum_{(s,a)} \pi_\theta(a|s) \log \pi_\theta(a|s) \tag{2.32}$$

Under the maximum entropy IRL settings, the policy π_θ is a distribution over trajectories, i.e. $p(T)$ where $T \sim \pi_\theta$. Ziebart et al. [26] suggested that when considering distributions over trajectories that match with the experts' feature expectations, the optimal choice is the distribution that has the maximum entropy. Thus, the objective of maximum entropy IRL is to find a policy π_θ that satisfies the following:

$$\max_\theta H(p) = \max_\theta \sum_{T \sim \pi_\theta} p(T) \log p(T) \tag{2.33}$$

subject to

$$\|\mu(\pi_E) - \mu(\pi_\theta)\|_2 \leq \epsilon, \tag{2.34}$$

$$\sum_{T \sim \pi_\theta} p(T) = 1, \tag{2.35}$$

$$p(T) > 0, \forall T \tag{2.36}$$

The first constraint (Eq. 2.34) refers to the feature expectation matching, while the other two (Eq. 2.35 and 2.36) ensure that p is a probability distribution. To capture the agent's policy π_θ, maximum entropy IRL uses a softmax function to assign probabilities to actions based on their associated values, which are determined by the action-value function $Q(s, a)$. The softmax policy can be expressed as:

$$\pi_\theta(a|s) = \frac{e^{Q(s,a)}}{\sum_a e^{Q(s,a)}} \tag{2.37}$$

2.3.3 Generative adversarial imitation learning

An alternative perspective on the IRL problem was presented in [25], where the authors demonstrated that the IRL problem can be formulated as a problem of matching occupancy measures. In this framework, the goal is to match the probability distributions of the states visited by an expert and, thus, the agent can learn to imitate the

expert's behavior and perform the same tasks. The *occupancy measure* ρ of the policy $\pi_\theta(a|s)$ can be defined as:

$$\rho_\pi(s,a) = \pi_\theta(a|s) \sum_{t=0}^{\infty} \gamma^t \mathbf{P}(s_t = s|\pi_\theta) \tag{2.38}$$

Generative Adversarial Imitation Learning (GAIL) [28] reformulated the IRL problem, by using the occupancy measure matching in conjunction with the maximum entropy principal [26]. Specifically, the new objective is to learn a policy that minimizes the difference between the agent's and the expert's occupancy measures, along with a regularizing term to account for entropy. Formally, this objective is stated as:

$$\min_{\pi_\theta \in \Pi} d(\rho_{\pi_\theta}(s,a), \rho_{\pi_E}(s,a)) - \lambda H(\pi_\theta) \quad , \tag{2.39}$$

where $d(.)$ is the distance between the occupancy measures of the policy and the expert, λ is a weighting factor and $H(\pi_\theta)$ is the entropy of policy π_θ. In GAIL, the Jensen-Shannon divergence is used as the distance function $d(.)$, and the optimum is met when this divergence is minimized.

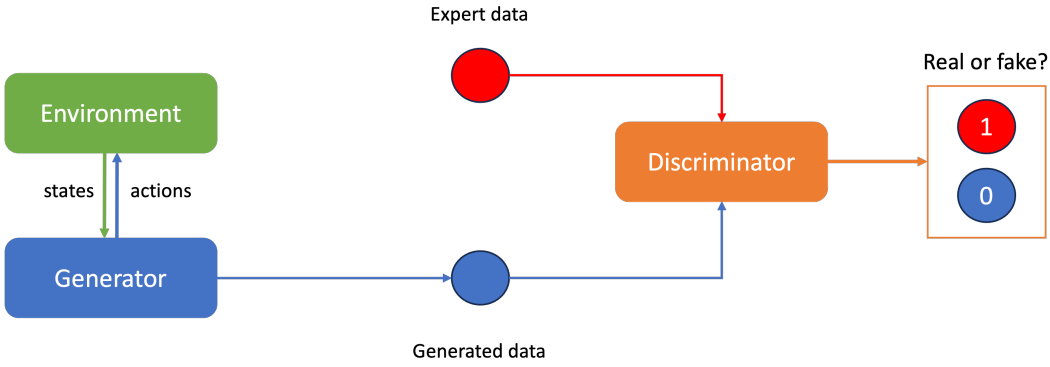


Figure 2.6: The architecture of generative adversarial imitation learning.

GAIL (Fig. 2.6) follows the GAN structure that comprises two neural networks: a generator network, G_θ, and a discriminator network, D_w. The generator produces samples that should follow the true data distribution, while the discriminator is trained to distinguish between the generated samples and the samples from the true data distribution.

In GAIL, the generator network follows an actor-critic architecture and models the conditional policy $\pi_\theta(a|s)$. On the other hand, the discriminator network, D_w,

is trained to separate expert and generated state-action samples, by outputting a probability that a given state-action sample comes from the true data distribution. In order to update the policy, GAIL uses a policy gradient method such as Trust Region Policy Optimization (TRPO) [29] or Proximal Policy Optimization (PPO) [30] and as a reward it utilizes the output signal from the discriminator. The formal objective of GAIL is defined as:

$$\min_{\pi_\theta} \max_{D} \mathbb{E}_{\pi_\theta}[logD_w(s,a)] + \mathbb{E}_{\pi_E}[log(1 - D_w(s,a))] - \lambda H(\pi_\theta) \tag{2.40}$$

According to this, the discriminator is trained by maximizing Eq. 2.40 with respect to the parameters w using Adam optimizer, while the generator is trained by minimizing with respect to the parameters θ using a TRPO or PPO step which concurrently maximizes the entropy of the policy ($H(\pi_\theta)$). Moreover, it is a common strategy to initialize the policy's weights with BC, and then use GAIL to refine them through the adversarial training process. BC provides an initial policy that is likely to be close to the expert's behavior, which can help GAIL to converge faster and produce better policies. Algorithm 4 specifies the training process of GAIL.

Algorithm 4: GAIL algorithm

Input: Expert trajectories T_E, number of number of iterations N, learning rate α, discriminator iterations per generator iteration N_D, number of samples K

Initialize the policy weights θ using BC and discriminator weights w randomly

for $i \leftarrow 1$ *to* N **do**

  while *not enough samples in generator's buffer* **do**

    Generate trajectories $T_i \sim \pi_{\theta_i}$

  for $j \leftarrow 1$ *to* k **do**

    Sample expert state-action pairs $(s_e, a_e) \sim T_E$

    Sample policy state-action pairs $(s_p, a_p) \sim T_i$

    Update discriminator D using gradient descent:

    $w \leftarrow w - \alpha \cdot \nabla_w \left[\frac{1}{K} \sum_{k=1}^{K} \log(D(s_e^{(k)}, a_e^{(k)})) + \frac{1}{K} \sum_{k=1}^{K} \log(1 - D(s_p^{(k)}, a_p^{(k)})) \right]$

  Sample policy state-action pairs $(s_p, a_p) \sim T_i$

  Update policy π using gradient ascent with TRPO/PPO:

  $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \left[\frac{1}{K} \sum_{k=1}^{K} \log(1 - D(s_p^{(k)}, a_p^{(k)})) \right]$

Chapter 3

Multi-agent Deep Reinforcement Learning for Traffic Congestion Management

In this chapter, an advanced deep multi-agent reinforcement learning approach is proposed for handling autonomous navigation of multiple vehicles in traffic networks consisting of road segments with unsignalized intersections. A key feature of the proposed method is the utilization of *route-agents* as the building block of the multi-agent system, which represent the possible routes that each vehicle can follow. This enables transfer learning and the re-usability of learned policies across different vehicles.

3.1 Overview

Autonomous driving constitutes a challenging environment for tasks such as perception, prediction and control [31, 32, 33, 34, 35]. In urban environments, the management of traffic and the autonomous navigation of vehicles are crucial problems and present interesting research and practical applications. It is expected that autonomous vehicles will exhibit intelligent behavior, including situational awareness, optimal path planning, and precise control, as they navigate urban areas to reach their destinations [36]. These vehicles will operate within traffic networks, guided by centralized and coordinated systems that prioritize safety and efficiency.

This chapter addresses a major challenge in autonomous driving, which is the traffic control and navigation of multiple vehicles in urban areas with unsignalized intersections. Unsignalized intersections are traffic intersections where there are no traffic signals or signs, and all directions have equal priority. In this scenario, vehicles enter a controlled traffic network and are automatically guided by the system to their pre-defined paths towards their destinations. Moreover, the system needs to handle a diverse range of vehicles, including passenger cars, buses, and trucks, simultaneously. Each vehicle has to select a driving policy from the available options that correspond to all possible routes. Subsequently, it executes a sequence of actions, such as levels of velocity, to navigate the road segments and intersections successfully, aiming to reach its destination without getting stuck, or causing collisions and deadlocks with other vehicles.

The task of autonomous navigation of multiple vehicles is formulated as a deep multi-agent reinforcement learning framework, which aims to train joint *route-agents*' policies and enable them to reconcile conflicting decisions. The framework utilizes a rich state space comprising features of the vehicles, as well as predictive information on the traffic flow at intersections, conveniently represented as a compact *collision matrix*. A value function approximation scheme is used through a combined weighted architecture consisting of two neural networks: a DQN for the vehicles' features and a convolutional neural network (CNN) connected to a DQN dealing with the predictive information. Additionally, the learning process incorporates the double Q-learning scheme, which enhances exploration within the search space, resulting in more robust and stable solutions.

Furthermore, a carefully designed reward function is introduced, with the objec-

tive of constructing policies that exhibit optimal navigation behavior, allowing vehicles to reach their destinations efficiently and safely. The proposed method has been evaluated through experiments conducted on both artificial and real urban road networks generated using the SUMO simulation environment [37]. The results demonstrate the effectiveness of the approach in successfully resolving traffic conflicts and congestion problems.

This chapter is organized as follows. In Section 3.2, a small literature on the research topic of autonomous driving is presented. The proposed collaborative multi-agent deep reinforcement learning scheme is described in Section 3.4. In Section 3.5 various simulated results are provided in order to evaluate the efficiency of the proposed method considering different scenarios, while Section 3.6 provides a brief summary of the chapter.

3.2 Related work

The literature in the field of autonomous vehicle control focuses on addressing congestion problems in traffic networks with intersections from various perspectives. One area of particular interest is the utilization of MAS, which has gathered significant attention.

Most of the research studies in this domain focus on controlling intersections using traffic lights. In some early works [38, 39], the authors presented a MARL algorithm which allowed global communication between traffic light agents in order to minimize the waiting time of vehicles in urban environments. Moreover, in [40] the problem of traffic signal control is formulated as a stochastic game where the agents employ distributed versions of the Q-Learning algorithm. A traffic control system using wireless senor networks has been proposed in [41] that gathers information from the wireless network and achieves real time adaptive traffic control improving the coordination between neighboring traffic lights.

A MARL framework is introduced in [42] that creates an efficient traffic signal control policy which minimizes the average delay, congestion and likelihood of intersection cross-blocking. Another study [43] employs the actor-critic framework to implement multi-crossroads traffic signal intelligent control. Additionally, the authors in [44] developed a collaborative reinforcement learning algorithm that uses Q-Learning

with a Boltzmann action selection scheme, and in [45] a cooperative mechanism between agents based on max-plus algorithm [46] is introduced, in order to achieve coordination between neighboring traffic lights. An integrated network of adaptive traffic signal controllers was introduced in [47], where the agent could either learn independently or according to the other neighboring agents. Also, another work in [48] proposes the use of a cooperative multi-agents with DQN for solving the problem of adaptive traffic signal control, where the neighboring agents share information about their latest action. Finally, many recent works [49, 50, 51, 52] approach the traffic signal control problem using DRL methods.

A new perspective has recently arisen in the literature by considering vehicles as intelligent agents instead of the traffic lights, aiming at offering safe navigation and reduced traveling time. These works specifically examine the behavior and decision-making of individual vehicles when navigating through intersection scenarios. One such methodology was presented in [53] where the authors formulated the intersection navigation as an optimization problem. They considered driverless vehicles as agents to be controlled by an intersection controller. In [54] a hierarchical policy gradient method is proposed to train the network with semi-markov decision process (SMDP) with temporal abstraction in traffic light passing scenarios. Furthermore, in [55] a deep deterministic policy gradient (DDPG) [56] algorithm is presented where sensor data is the state and the agent is demonstrated that was capable of driving efficiently.

In another study [57], a DRL approach is utilized for autonomous navigation and obstacle avoidance in self-driving cars. The input to the system consists of image data captured by sensors, which are then processed to determine the appropriate acceleration and orientation of the vehicle. Similarly, in [58], the asynchronous advantage actor-critic (A3C) [59] framework is employed for learning vehicle control. However, in this case, the control is learned solely based on RGB images captured by a forward-facing camera. Moreover, in [60] deep recurrent Q-networks are used to train a vehicle to cross an intersection with three other (human) vehicles by adjusting its acceleration after a negotiation process.

A DQN approach is used in [61] to steer a vehicle in a 3D physics simulation relying solely on camera image input, while in [62] the authors addressed the problem of autonomous traversing though an urban partially non-controlled stop-sign intersection by using DDPG and hierarchical approaches. Furthermore, [63] employed a

DQN to navigate a single vehicle-agent through occluded intersections, and in [64] a multi-objective deep reinforcement learning variant of thresholded lexicographic Q-learning is developed for the task of urban driving and collision avoidance on multi-lane roads and intersections. Finally, in a recent study [65] the authors take advantage of the concept of connected and automated vehicles in order to establish communication and information sharing between the vehicles. They implement a car following model which is centralized and can learn driving behaviors to improve travel efficiency at signalized intersections.

From another perspective, the works in [66, 67] deal with the problem of stochastic shortest path by maximizing the probability of arriving at the destination on time. They propose a multi-agent route guidance system, where the infrastructure agents are responsible for guiding the agent vehicles based on their intentions.

Summarizing, the research literature of learning autonomous vehicles to navigate through unsignalized intersections is fairly new and promising. Previous works on this area present four (4) main limitations:

- They use signalized intersections or traffic lights to facilitate the management of the traffic.
- Assume a single agent that learns to pass an intersection and avoids collision with other human controlled vehicles.
- Create multi-agent approaches of limited capabilities mainly due to the reason that they consider vehicles as agents.
- Deal with less complex environments that contain a limited number of intersections.

In the present study, a compact multi-agent methodology is implemented that aims at addressing all these issues and offering more robust solutions.

3.3 Problem setting and the notion of route agent

The objective of this work is for a fleet of vehicles to navigate through urban road networks with one or more unsignalized intersections and co-operate with each other in order to provide safety, while simultaneously trying to minimize the traveling time.

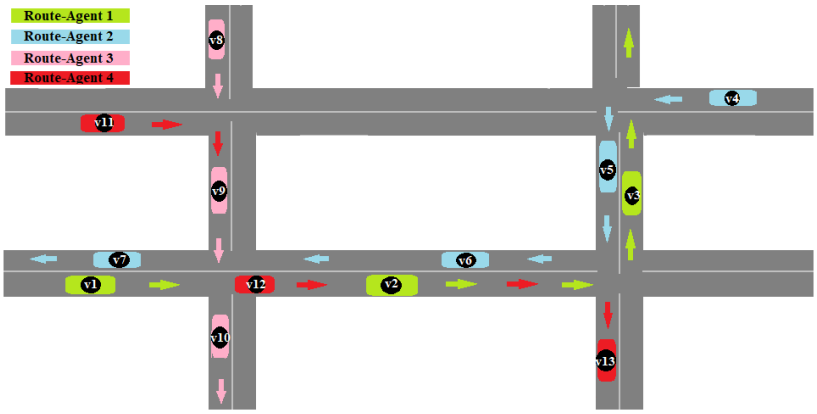


Figure 3.1: An example of a traffic zone with four route-agents that correspond to four paths. Every route-agent simultaneously controls more than one vehicles that follow the same path.

Traffic networks consisting of numerous roads that are connected with multiple intersections, constitute the basic environment for this study. Figure 3.1 gives an example of a traffic networks with two unsignalized intersections. This study considers as agents the various pre-defined routes (or paths) of the traffic network, called *route-agents*, that vehicles can select to follow in order to reach their destinations. Each *route-agent* models the driving behavior of any vehicle that follows the corresponding route by controlling its velocity. Vehicles that are simultaneously located on the same route and therefore are served by the same route-agent, take different actions (velocity values) according to their individual state.

Figure 3.1 illustrates an instance of a traffic zone consisting of four (4) intersections. In that snapshot, there are 13 vehicles that follow four (4) routes and thus are served by four (4) *route-agents*, correspondingly. Specifically, route-agent 1 controls simultaneously the three (3) green vehicles (v_1, v_2, v_3), route-agent 2 controls the four (4) cyan vehicles (v_4, v_5, v_6, v_7), and so on. According to the examined scenario, the vehicles sequentially enter the traffic zone from any of the entrance lanes (referred to as "entrance points") in order to be automatically navigated to their destination. Vehicles appear at the entrance points according to a pre-defined arrival rate. It is also assumed that vehicles are served immediately without any delay. Then, an *agent identification* process takes place that assigns an agent to every vehicle based on its specific route, i.e. a sequence of roads that the vehicle will cross in order to reach its destination. The corresponding *route-agent* will be activated and will guide the vehicle

in an optimal way.

Since the traffic network can be separated into several routes, the proposed scheme establishes a multi-agent framework assuming a group of route-agents that act together and try to resolve the scenario. The role of intersections is to coordinate route-agents and distribute information to all vehicles traveling towards them. Solving a traffic scenario means that the society of route-agents should learn to accomplish their goal within a sequence of actions and cooperate with each other for collective success, as well as to make decisions within limited local observations.

3.4 Multi-agent deep reinforcement learning structure

The task of autonomous navigation of multiple vehicles in a traffic network with unsignalized intersections is formulated under a collaborative multi-agent framework. The MAMDP is employed for modeling the agents and a weighted deep reinforcement learning scheme is proposed for solving it. A detailed description of the MAMDP framework has been presented in Chapter 2.2.5. The following sub-sections describe the state space, action space, the construction of the reward function and the proposed algorithm for tackling the problem.

3.4.1 State space with individual and predictive information

The state space should contain as many informative variables as possible in order to describe the environment. For this purpose, a unified state space, $S = S^{(1)} \bigcup S^{(2)}$, which comprises two components is designed. The first component, $S^{(1)}$, encapsulates individual information of the examined, or *ego vehicle*, while the second component, $S^{(2)}$, encompasses information pertaining to the anticipated traffic flow at the intersections. In particular:

- **Individual state** $S^{(1)}$:
  - Current velocity of the ego vehicle (**vel**)
  - Distance of the ego vehicle from the next visiting intersection's center (**inter_dist**)
  - Velocity of the front vehicle (if any) in the same lane (**fveh_vel**)

– Distance of the ego vehicle from the front vehicle (if any) in the same lane (**fveh_dist**)

- **Predictive state** $S^{(2)}$: a predictive matrix that displays the estimated traffic flow in the next intersection that the ego vehicle will pass though.

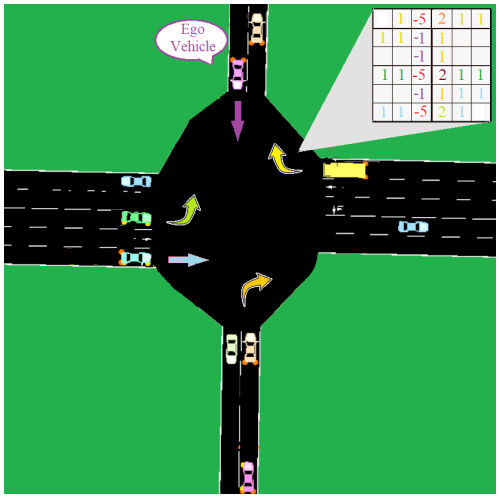


Figure 3.2: An example of the collision matrix generation mechanism.

The *collision matrix* is constructed by mapping the intersection's traffic to a matrix of size determined by the intersection's geometry (i.e., number of lanes of the crossing roads). Each vehicle can occupy a single cell within the matrix, and assuming a constant velocity for all nearby vehicles (equivalent to their current velocity), the matrix cells are populated with relevant values derived from the trajectories of these vehicles. Moreover, it is assumed by the ego vehicle that every other vehicle moves in a straight line at a constant velocity, thereby it possesses limited knowledge about them. All the cells within the matrix that encompass the intended path of the ego vehicle inside the intersection are assigned a value of minus one (-1). In the event of a potential collision with another vehicle (i.e., where their trajectories are anticipated to intersect), the corresponding cells in the matrix are assigned a value of minus five (-5) to differentiate it from the non-collision scenario (-1). The remaining cells in the matrix are assigned a value that quantifies the frequency of visits by other vehicles during the time it takes for the ego vehicle to traverse the intersection.

Figure 3.2 presents an example of the creation of the collision matrix. In this case, the collision matrix is structured as a 6×6 grid, where each row and column represents a lane traversing the intersection. Note that for the vertical lanes there are only two

possible columns that correspond to valid positions. The ego vehicle, depicted in purple color, follows a straight-line trajectory while crossing the intersection. Thus, all the cells corresponding to the third column of the matrix will be set to minus one (-1). However, as it is estimated, the ego vehicle will collide with three vehicles: the yellow track on the right, the middle lane green car on the left and the left down lane cyan car. Therefore, the 1st, 4th and 6th position of the 3rd column will take the value minus five (-5), so as to denote undesired situations.

In an attempt to emulate realistic scenarios, the ego vehicle possesses incomplete information regarding the behavior of other vehicles. Specifically, it simplistically assumes that the other vehicles will travel across the intersection in a straight line, disregarding any potential turns (either right or left) as suggested by their respective routes (unknown to the ego vehicle). In the above example, this situation arises with the yellow track on the right and the green car in the left middle lane. Based on this assumption, both vehicles will cross the intersection without turning and hence will fill the rows (1st and 6th, respectively) of the matrix horizontally. On the other hand, when a vehicle passes through a position-cell of the matrix that is different from the ego vehicle's path, then a frequency counter is increased by one in that position.

3.4.2 Action space

The choice of the action space is a crucial aspect of an RL agent. In this study, the agent governs the velocity of the vehicle, which serves as the action to be controlled. Four (4) possible levels of velocity are chosen as actions, denoted as $A = \{15, 20, 25, 30\}$ (measured in m/sec).

Initially, when a vehicle enters from an "entrance point" within the traffic network, it adopts the lowest velocity value of $15\ m/sec$ as the initial action. Subsequently, at each time step, the agent selects an action (velocity) based on the policy it adheres to. Afterwards, at every time step, the agent takes an action (velocity) according to the policy it follows, and then a low-level controller handles the transition from the current velocity to the desired velocity.

It must be noted that the low-level controller might take some time to reach the desired velocity of the agent. This "delay" in action is correlated with the acceleration of each vehicle type, which is set according to the SUMO simulator's standards.

3.4.3 Reward function

The proposed reward function is designed to eliminate the collisions and minimize the traveling time among vehicles inside the urban road network. It is formulated as follows:

$$R(s,a) = \begin{cases} +L & , \text{ if it reaches goal} \\ -L & , \text{ if it collides} \\ r(s,a) & , \text{ otherwise} \end{cases} \tag{3.1}$$

Initially, it is evaluated whether the ego vehicle has reached a terminal state. If it has, then the agent gets a positive reward of L when it successfully reaches the destination or a penalty of $-L$ in the event of a collision. In any other scenario, the following are considered: a) whether or not the distance from the front vehicle ($fveh\_dist$) is lower than a threshold value $safe\_dist$ (set to $50\ m$), and b) whether ($C = -1$) or not ($C = 1$) the vehicle is estimated to participate in a collision while traversing the next intersection. There are two possible circumstances:

- The front vehicle is far enough i.e. $fveh\_dist > safe\_dist$. Then, the (positive or negative) size of the received reward depends on the vehicle's velocity in a way of promoting higher velocities:

$$r(s,a) = C * \beta_1 * \frac{vel}{vel\_max} \tag{3.2}$$

  where $vel\_max = 30\ m/s$ denotes the maximum allowed velocity and β_1 is a coefficient set to three (3).

- In the opposite case, where the front vehicle is near, i.e. $fveh\_dist < safe\_dist$, a negative reward is always received that depends both on the vehicle's velocity and the magnitude of the distance with the front vehicle:

$$r(s,a) = -(\beta_2 + \frac{vel}{vel_{max}}) * (safe\_dist - fveh\_dist) \tag{3.3}$$

  where the value of β_2 depends on the collision situation: $\beta_2 = 1$ if $C = 1$ and $\beta_2 = 2$ if $C = -1$.

3.4.4 Algorithmic description

For approximating the Q-value function a weighted scheme of two neural networks is introduced:

- A simple DQN that captures the ego vehicle's individual state information $S^{(1)}$.
- A combined architecture of a CNN with a DDQN responsible for handling the produced *collision matrix* of the predictive state $S^{(2)}$. The purpose of the CNN is to extract features from the *collision matrix* that will be passed to the DDQN in order to select the appropriate action.

Both networks output an action among the four (4) possible levels of velocity. Then, the Q-value function is calculated as:

$$Q(s,a) = \mu\ Q_{DQN}(s^{(1)},a) + (1-\mu)Q_{DDQN}(s^{(2)},a) \tag{3.4}$$

where $Q_{DQN}(.)$ and $Q_{DDQN}(.)$ denotes the outputs of the two neural networks, while $\mu \in [0,1]$ is a positive parameter that shows their level of contribution to the decision making mechanism of the system.

Algorithm 5 describes the learning process of the proposed method.

Algorithm 5: Combinatorial DRL architecture for autonomous navigation

Input: number of iterations N, learning rate η, discount factor γ, episodes until update of target network N_u, scenario's duration T
Create the DQN, and CNN/DDQN networks for every *route-agent*
Initialize two replay buffers for every *route-agent* (one for each network)
for $i \leftarrow 1$ *to* N **do**
  while $t < T$ **do**
    for *every vehicle* i *found currently in the traffic zone* **do**
      Identify the corresponding route-agent j
      Obtain the individual state $s_{i,t}^{(1)}$
      Construct the collision matrix to get the predictive state $s_{i,t}^{(2)}$
      Obtain the Q-values from the DQN with input $s_{i,t}^{(1)}$
      Obtain the Q-values from the CNN/DDQN with input $s_{i,t}^{(2)}$
      Calculate the final Q-values according to eq. 3.4 and select an action based on ϵ-greedy strategy
      Move to next state and receive a reward (Eq. 3.1)
      Save the experiences into the replay buffers of the corresponding DQN and CNN/DDQN route-agent
    Update the main networks with batches sampled from the replay buffers
  Update the target networks every N_u episodes
Store the learned route-agents' policies

3.5 Simulation results

3.5.1 SUMO simulation environment

SUMO (Simulation of Urban MObility) is an open-source microscopic traffic simulation software developed by the German Aerospace Center (DLR) and the Institute of Transportation Systems at the University of Karlsruhe. It provides a platform for simulating and analyzing traffic flow in urban environments. SUMO is widely used in the field of transportation research, urban planning, and traffic engineering.

SUMO allows users to model complex urban traffic scenarios and simulate various aspects of traffic behavior, such as vehicle movement, lane changing, and traffic signal control. The simulation can be customized to reflect real-world conditions, including road networks, traffic flows, and various vehicle types. It supports a range of traffic simulation models, including car-following models, lane-changing models, and microscopic traffic flow models.

SUMO provides the option to create artificial scenarios via an editor, or select areas from Google Maps and transform it to SUMO scenario. Through the SUMO editor package there can be modified a variety of options about the traffic network, such as the number of intersections, number of roads, roads' length, maximum allowed velocity, and several vehicles' characteristics (type, vehicle, maximum velocity, maximum acceleration and arrival rate).

3.5.2 Data description

(a) Artificial map (b) Real-world map

Figure 3.3: Snapshots of the artificial scenario and the real-world map of Devonshire street, Florida.

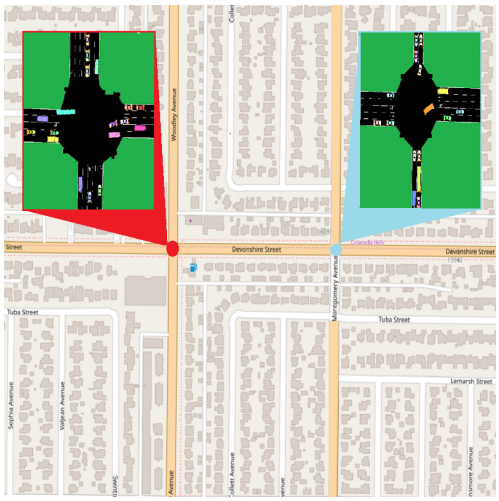


Figure 3.4: The non-symmetrical design of intersections in the real-world scenario.

The proposed deep multi-agent reinforcement learning framework was evaluated on three (3) different traffic scenarios (Table 3.1):

- an artificial scenario (Fig. 3.3(a)) with two (2) intersections, where each road is two-way and has a single lane,
- two (2) real-world scenarios (Fig. 3.3(b)) created by choosing a specific road map in California (Devonshire Street) with two intersections and road segments with multiple lanes and non-symmetrically design (see Fig. 3.4). These scenarios differ on traffic volume they cover (ie., duration and number of vehicles).

Table 3.1: Description of the scenarios used in the experimental study.

Scenario	**# vehicles**	**Duration** (min)	**#routes** (agents)	**Distance** (km)	**Longest Route** (km)
$Artificial$	300 (228 cars, 48 buses, 24 trucks)	20	18	2.8	0.600
$Real\_15$	225 (171 cars, 36 buses, 18 trucks)	15	30	7.2	1.725
$Real\_30$	450 (342 cars, 72 buses, 36 trucks)	30	30	7.2	1.725

Table 3.1 provides a detailed description of these scenarios. To be as close as possible to real traffic scenarios, all cases included several types of common vehicles, such

as passenger cars, buses and trucks, with a frequency of $75/15/10$, correspondingly. However, it must be indicated that all these types are treated with the same manner, since the *route-agents* are not specialized in any of them.

In all cases the safety checks performed by SUMO are turned off, while the traffic lights and the priority rules have been removed from the intersections. Furthermore, we devised a deterministic rule for the vehicles reaching the last road segment by setting the maximum allowed velocity ($30\ m/s$) until finishing their route and reaching their destination.

3.5.3 Implementation details

This section provides: (a) implementation details about the NNs and their training, and (b) metrics for evaluating the accuracy of the methods.

The simple DQN, which is responsible for the ego vehicle's individual state, consists of one hidden layer with 100 ReLU hidden units and an output layer with four (4) neurons equal to the possible actions. On the other hand, the second neural network employs a CNN to deal with the collision matrix. The input matrix passes through eight (8) convolutional filters of size 3×3, then down-sampled using 2×2 max-pooling and flattened into a vector. This vector is then passed through a DDQN network with a hidden layer of 200 neurons and an output layer of four (4) neurons that correspond to the available actions.

Regarding the selection of the action from the agent, a weighted combination of the outputs of the two networks is employed as described in Eq. 3.4. Several values were applied to the parameter μ in the range $[0.3, 0.7]$ during the experimentation, but without observing any significant effect on the performance. Hence this parameter is set to $\mu = 0.5$ in order to provide equality between the two networks.

Two copies of the original DQN and DDQN networks are used as target networks to improve training stability. During the learning process, these twin nets update their weights every five (5) episodes with the current weights of the original networks (hard update). Furthermore, both networks use batches of size 2500 while two experience replay buffers are maintained, one for every network, of size 15000 each. The training is performed at the end of each episode and the optimization is being conducted by Adam optimizer.

Additionally, an ϵ-greedy exploration-exploitation scheme is used, which consists

of 15000 episodes for training in the case of *Artificial* and *Real_15* scenarios, composed of 13000 episodes for exploration and another 2000 episodes for exploitation. Due to the higher complexity of the *Real_30* scenario, more exploration is needed, hence 20000 learning episodes are considered (18000 for exploration and 2000 for exploitation). The probability of ϵ is initially set to 0.9 and is diminished by 0.01 every 150 steps until it reaches a threshold where it is set to zero (0) thereafter. The learning rate was set to $\eta = 0.001$ and the discount factor $\gamma = 0.99$.

The episodes are considered terminated when all vehicles have left the traffic map. It must be noted that during training, when two or more vehicles collide, the episode does not terminate, but rather the involved vehicles are removed from the road network. This allows the method to gather more experiences for the replay buffers and reduce the number of training episodes.

The following three (3) metrics are used for measuring the performance of the proposed method:

- **Average traveling time**: mean time (in s) for the vehicles to reach their destinations
- **Average velocity**: mean velocity (in m/s) of all vehicles
- **Collisions**: total number of collisions during the episode

3.5.4 Results and robustness analysis

A comparison between the proposed method and two (2) car-following models of SUMO is provided in this section. Specifically, the proposed method's performance is evaluated against the default Krauss model and the well-known Intelligent Driver Model (IDM). The Krauss model is a spatial continuous car-following model, where the examined vehicle keeps a certain distance with the leading vehicle by computing a safe velocity between them, while the IDM aims at reaching the desired velocity by accelerating or braking depending on the current vehicle's velocity, as well as the position and velocity of the front vehicle.

Table 3.2 presents the comparative results between the three (3) methods applied to three (3) scenarios in terms of the statistics (mean value and standard deviation) of the evaluation metrics: *average traveling time*, *average velocity* and *number of collisions*. The proposed multi-agent approach outperforms significantly the other two models

Table 3.2: Comparative statistical results of the proposed method against two SUMO's default models in terms of three criteria.

Scenarios	**Our method**	**Krauss**	**IDM**
Average traveling time (sec)			
Artificial	**15.87 ± 0.13**	57.83 ± 0.78	66.92 ± 0.65
Real_15	**34.40 ± 0.56**	63.61 ± 2.61	101.08 ± 0.75
Real_30	**35.15 ± 0.90**	66.68 ± 0.32	110.33 ± 0.85
Average velocity (m/sec)			
Artificial	**26.61 ± 0.24**	13.40 ± 0.99	9.62 ± 1.97
Real_15	**25.82 ± 0.53**	20.34 ± 1.26	13.02 ± 0.30
Real_30	**25.18 ± 0.80**	19.18 ± 0.34	11.97 ± 0.28
Number of collisions			
Artificial	**0.00 ± 0.00**	0.00 ± 0.00	0.00 ± 0.00
Real_15	**0.00 ± 0.00**	3.00 ± 1.23	1.00 ± 0.71
Real_30	**0.00 ± 0.00**	6.60 ± 0.54	1.40 ± 0.55

in all cases. It always manages to solve successfully the scenarios obtaining solutions with no collisions, in contrast to the rest two methods which fail in both real-world scenarios. Moreover, it provides qualitatively better solutions and improved policies. The reported results on average *traveling time* and *velocity* show the efficiency of the method to offer much faster navigation and boost the highest velocity. This also demonstrates the effectiveness of the proposed reward function and the compact Q-function approximation scheme.

Moreover, in Fig. 3.5 the learning curves of the proposed method are provided, as received during training in terms of the *average traveling time*. The linearity of these curves proves the effectiveness of the learning process maintaining enhanced convergence capabilities without the existence of plateau phenomena.

Additional experiments were made in an attempt to further study the perspective of the proposed method to successfully offer knowledge reuse for autonomous navigating tasks, and transfer of the learned policies in unknown scenarios that share the same properties as the original trained ones.

The proposed reuse of knowledge is found on the learned policies of the route-

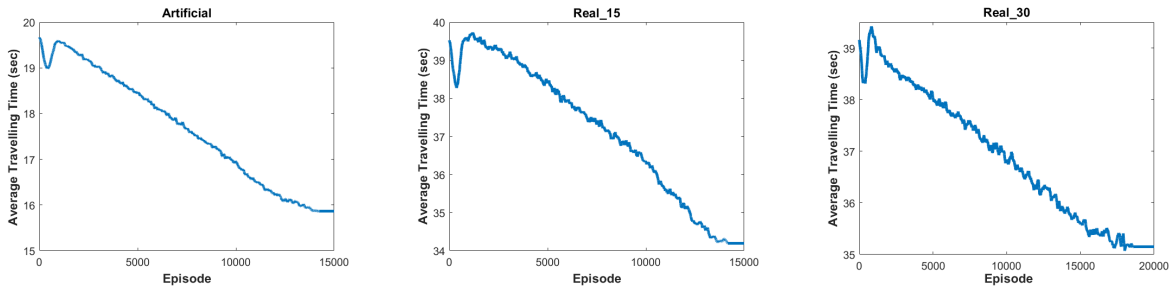


Figure 3.5: The obtained learning curves of the proposed method during training in terms of mean values of the average traveling time.

Table 3.3: Description of the unknown scenarios used for knowledge reusing.

Scenario	**# vehicles**	**Duration** *(hours)*
$Test1$	900 (684 cars, 144 buses, 72 trucks)	1
$Test2$	1800 (1368 cars, 288 buses, 144 trucks)	2
$Test3$	4500 (3420 cars, 720 buses, 360 trucks)	5

agents. In total, fifteen (15) copies of the real scenario (Fig. 3.3) with three (3) degrees of complexity (i.e., number of vehicles and duration) are generated. The goal is to evaluate the effectiveness of the learned policies of route-agents to more demanding and realistic cases and to measure their generalization capabilities. Table 3.3 gives the detailed characteristics of these generated scenarios.

A final set of experiments was conducted on the learned policies that have been discovered in the $Real\_15$ and $Real\_30$ scenarios. Note that there are available ten (10) different sets of policies for each training scenario, since five (5) simulated copies of them were used during the training of the multi-agent system. Thus, every test scenario is evaluated using $5(copies) \times 10(policies) = 50$ reuses of the discovered driving policies. Again, the same results were also obtained using the car-following models provided by the SUMO environment for comparative purposes. The statistical results are summarized in Table 3.4.

In all cases, the knowledge reuse of the trained policies was very successful since: (a) no collision occurred, and (b) the average *traveling time* and *velocity* obtained on the training scenarios (Table 3.2) are maintained. This is a very important result

Table 3.4: Driving behavior evaluation of the learned policies to unknown scenarios.

Scenario	$Real\_15$ **policies**	$Real\_30$ **policies**	**Krauss**	**IDM**
Average traveling time (sec)				
Test1	$\mathbf{34.81 \pm 0.44}$	$\mathbf{34.49 \pm 0.14}$	68.24 ± 0.22	113.47 ± 0.33
Test2	$\mathbf{36.12 \pm 0.11}$	$\mathbf{36.01 \pm 0.08}$	71.53 ± 0.96	117.62 ± 0.81
Test3	$\mathbf{37.22 \pm 0.92}$	$\mathbf{37.02 \pm 0.84}$	76.19 ± 0.87	123.44 ± 0.90
Average velocity (m/sec)				
Test1	$\mathbf{25.66 \pm 0.27}$	$\mathbf{26.00 \pm 0.41}$	18.14 ± 0.39	11.26 ± 0.74
Test2	$\mathbf{25.23 \pm 0.28}$	$\mathbf{25.31 \pm 0.31}$	17.52 ± 0.55	10.47 ± 0.48
Test3	$\mathbf{25.02 \pm 0.73}$	$\mathbf{25.12 \pm 0.69}$	16.94 ± 0.97	9.69 ± 0.73
Number of collisions				
Test1	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	9.20 ± 1.42	5.03 ± 1.18
Test2	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	10.04 ± 1.03	5.11 ± 1.44
Test3	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	11.73 ± 1.98	7.41 ± 1.17

that shows the level of stability and the generalization capabilities that the proposed deep multi-agent framework offers. According to the results, both SUMO embedded methods were unable to successfully treat such complex scenarios as they produced solutions with collisions.

As it was expected, the policies provided by the $Real\_30$ ($30\ min$ duration) scenario gave slightly better results since they were built using more training examples and traffic flow cases. However, the performance measured by the $Real\_15$ policies which is a training scenario with only 15 minutes of traffic, does not lag behind its competitor, and seems adequate to be successfully transferred and reused.

3.6 Summary

This chapter addressed the problem of autonomous navigation of vehicles in traffic networks with road segments and unsignalized intersections using a deep MARL approach. A combinatorial scheme was implemented for the approximation of the Q-function, consisting of two neural networks structures for efficiently treating the

intelligent agents' state space. Among the novelties of this study is the introduction of the *route-agents* and the creation of a predictive *collision matrix* on the traffic flow of intersections that allows agents to cooperate and learn joint policies to avoid collisions. Initial simulation results are encouraging, showing great performance even in real-world urban road networks. Finally, successful knowledge reuse and transfer of learned policies determines the ability of the proposed method to easily adapt to unknown traffic environments and to offer flexible solutions with increased generalization.

Chapter 4

Robust Traffic Management in Complex Urban Road Networks via Multi-Agent Reinforcement Learning

In this chapter, an extension of the methodology introduced in Chapter 3 is presented, addressing the problem of autonomously navigating vehicles in large-scale urban road networks with unsignalized intersections. Building upon the foundations laid out in the previous Chapter, this chapter presents novel advancements and explores additional aspects of the problem, expanding the scope and depth of the analysis.

In this study, the exploration continues introducing a richer state space, a diverse set of actions and an improved reward function. This chapter expands the boundaries of applicability by introducing an updated deep MARL method for handling large-scale road networks and extends the transfer learning framework in unknown stochastic scenarios.

Moreover, it aims to address certain limitations that were identified in the previous chapter, such as the creation of the *collision matrix* which becomes infeasible in large-scale road networks. By establishing a new source of predictive information for the unsignalized intersections, the effectiveness and efficiency of the proposed approach are enhanced, improving its practicality and robustness.

4.1 Overview

One of the most challenging problems in autonomous driving in urban environments is to handle intersections and traffic congestion problems as more and more vehicles appear to the transportation system [68]. The traffic density especially in large cities is increased and becomes a confusing problem. Consequently it becomes imperative to exert every possible effort to efficiently regulate traffic flow by developing *traffic zones* where vehicles can autonomously navigate through intersections and roundabouts to reach their destinations. Intersection management, as highlighted in [69, 70], is an emerging research field and one of the most pressing and complex problems faced by modern society.

Vehicles are expected to be able to drive autonomously to their destination in urban areas holding an intelligent behavior for situational awareness, optimal path planning and control. They will be connected to traffic zones under a centralized guidance designed in a coordinated fashion [36] in order to allow safety and efficiency while *passing-through intersections* [71]. Artificial intelligence constitutes a framework with tools as leverage for constructing intelligent, autonomous control and decision-making algorithms in an attempt to provide a more efficient, comfortable and accident-free traffic system.

The importance of traffic control at road intersections in urban environments can be reflected by the fact that they account for 40% of driving accidents in the USA according to the Fatality Analysis Reporting System (FARS) and the National Automotive Sampling System-General Estimates System (NASS-GES) data [72, 73, 74]. This study focuses on unsignalized (or uncontrolled) intersections, which refer to traffic intersections lacking traffic signals or signs, where all directions are given equal priority. Within these areas, vehicles must effectively manage their acceleration and determine how to navigate the intersections without creating collisions or deadlocks.

A key challenge lies in implementing cooperative driving strategies to address traffic conflicts and ensure smooth traffic flow [75, 76].

Many methods for controlling traffic intersections have been proposed over the last years that utilize traffic lights. It is believed that "*without traffic lights dense traffic would come from all sites of the road and would block the intersection without letting any vehicle pass through*" [77]. However, effectively managing traffic flow in the absence of traffic lights or other traffic signs poses a significant challenge, and it serves as the primary objective of this study. Within this research, each vehicle is assumed to follow a pre-defined *path* or *route*, which is part of a traffic zone comprising interconnected roads and intersections. The main goal is to develop efficient *driving profiles* for the vehicles by controlling their acceleration along their designated routes, ensuring a successful and safe arrival at their respective destinations.

Thus, the problem of autonomous navigation is decomposed into a problem of coordinating vehicles on pre-defined paths where variable levels of acceleration are imposed as actions along each path. The main challenge here is to construct traffic control zone policies that provide autonomous navigation behaviors in a way of enabling easy adaptability and transferability.

One of the key challenges when applying MAS to a task is to alleviate the burden of learning and allow the exploitation, sharing and *reusing of the knowledge* generated throughout decision-making process [78, 79]. Transfer learning focuses on storing obtained knowledge from the solution of one problem and applying it to a different but related problem. It can significantly reduce learning time and create more solid intelligent agents. Knowledge reuse becomes more and more a core technology in agent-based learning systems where *an agent can establish relationships with other agents that allow implicit or explicit knowledge sharing, and integrate the received information with its previous experience to improve learning* [80].

In this study, the problem of autonomous navigation of multiple vehicles in unsignalized large-scale urban road networks is formulated as a MAMDP based on a collaborative RL framework for learning agents' policies and supporting agents to reconcile conflicting decisions [17, 81]. The main contributions of this study can be summarized as follows:

- It considers as agents the possible routes that the vehicles can follow within a controlled traffic zone. The definition of *route-agents* (introduced in Chapter 3) has a twofold advantage: Firstly, it provides an efficient solution in MARL

schemes by mitigating the complexity and constraints associated with treating each vehicle as an independent agent. As a consequence, it positively impacts the convergence and quality of the agents' learning process, since multiple copies of the same agent (representing vehicles following the same route) can be addressed simultaneously and in a parallel fashion at each time step. Secondly, it allows transfer learning and reuse of agents' policies in handling unknown scenarios with multiple vehicles and increased stochasticity. As experimental results have shown, the proposed method manages to establish efficient knowledge reusing and to produce robust agent policies with generalization capabilities over unknown scenarios.

- It introduces a *collision term* that allows the cooperation between vehicles. This term serves as a prediction of whether or not a vehicle will potentially collide with other vehicles at the next intersection it visits. All vehicles that are expected to participate in the same collision need to coordinate their strategies and to execute their tasks jointly and cooperatively. As experiments have shown this information plays a crucial role to the performance of the proposed method and increases the quality of the learned policies.

- An efficient reward function is proposed that aims at constructing agents' policies with optimal autonomous driving behaviors. Experiments have shown that the proposed method achieves significantly less traveling time in comparison with the standard methodology which is accompanied with the SUMO simulation environment and an independent MARL framework that does not consider the *collision term*.

- It evaluates the proposed method on various simulated traffic scenarios considering complex urban road networks with unsignalized intersections and diversity in the characteristics of the vehicles. According to the results, the proposed method has the capability of successfully resolving traffic conflicts and congestion problems.

- Finally, it provides a way to reuse the learned *route-agents*' policies in unknown, stochastic environments. The results indicate that the learned policies can resolve the congestion problems in the unknown scenarios, and produce solutions comparable to the training cases.

This chapter continues with the proposed multi-agent deep reinforcement learning scheme which is described in Section 4.2, while in Section 4.3 the evaluation cases and the simulated results for measuring the efficiency of our method are presented, considering both artificial and real scenarios. Finally, in Section 4.5 the conclusions are drawn. It must be noted that the related work and the problem formulation remain the same as in Chapter 3.

4.2 Multi-route-agent deep reinforcement learning for Traffic Congestion Management

The aim of this study is to efficiently control the vehicles' acceleration so as to move them safely (avoiding collisions) and efficiently (minimizing traveling time with high velocity and fuel economy) to their destination. This task is formulated as a MAMDP that constitutes an efficient mathematical framework for sequential decision making problems in a MAS. The full description of the MAMDP framework has been presented in Chapter 2.2.5.

It must be noticed that handling simultaneously multiple copies of the same route-agent (i.e. vehicles that follow the same path) offers more flexibility to the learning process without affecting the reasoning abilities and the decision making of each agent. Note also that the route-agents are vehicle-type-specific, meaning that they are specialized to the vehicle's physical properties. Specifically, two types of vehicles are considered: (a) passenger cars and (b) buses and trucks, thus two (2) kind of *route-agents* are created for each route.

In the subsequent sections, we describe the state space, action space and the proposed reward function for the aforementioned problem.

4.2.1 Partially observable state space

Following the proposed route-agent profiling scheme, every vehicle before being served performs an *agent identification* process, where a specific route-agent undertakes its driving behavior and guides it to its destination.

At each time step, several vehicle sensing information can be extracted such as position, velocity, distance from the next visited intersection's center ($inter\_dist$), and

so on. Based on the previous information, the time that it would take for a vehicle to reach the center of the next visited intersection ($inter\_time$) can be calculated according to the following rule:

$$inter\_time = \frac{inter\_dist}{current\_velocity} \tag{4.1}$$

The latter quantity allows to predict whether or not two or more vehicles will collide while crossing the same intersection. It must be noted that all processing is handled in a centralized fashion, where each intersection keeps a dedicated controller that collects and distributes these predictions to all participants.

In this study, a continuous state space for describing vehicles is designed. It consists of the following quantities:

- **Velocity** of the *ego vehicle* (continuous value) that has a maximum allowed value.
- **Position** of the *ego vehicle* along the lane it traverses.
- **Velocity of the front vehicle**. If there isn't any front vehicle, then the default value of minus one (-1) is given.
- **Distance from the front vehicle**, measured as the Euclidean distance between the two vehicles. Again, if there isn't any front vehicle, then the default value of minus one (-1) is given.
- **Collision term**. This is a binary quantity that provides a prediction whether or not the *ego vehicle* is going to collide with other vehicles in the next visited intersection. It has a key role in the proposed approach as it allows the cooperation among vehicles. This is calculated by comparing the $inter\_time$ quantity of Eq. 4.1 between all vehicles that are moving towards the same intersection. If there are vehicles with $inter\_time$ values that differ from the $inter\_time$ of the *ego vehicle* by a small threshold, the collision term of the *ego vehicle* will be set to one (1). This means that it is expected to participate in a collision. In this experimental study the above threshold value is set to $1.5\ s$. Note that either greater, or lower value makes this feature of less importance since it corresponds to either frequent or rare collisions at intersections, respectively.
- **Density term** that shows the number of vehicles, that the *ego vehicle* is predicted to collide with at the next intersection. It is another predictive quantity that

helps at the coordination of the vehicles approaching an intersection, and it is normalized by dividing with the number of incoming lanes of the specific intersection.

Figure 4.1: An example of how the collision term is calculated.

Figure 4.1 gives an example of the collision term's computation. In particular it presents a snapshot of the SUMO simulator where four vehicles move towards the same intersection. According to their current velocities and distances from the intersection's center, it is estimated that both the pink (up right) and the green (down right) vehicles will collide since their $inter\_time$ quantity differs less than $1.5\ s$. Thus, their collision term will become one (1). On the other hand the other two (2) vehicles, blue (down left) and yellow (up left), are not expected to interfere with any vehicle and therefore their collision term will be set to zero (0).

To sum up, the agent obtains two kind of information for the corresponding vehicle: (a) its current velocity and position, distance from the front vehicle and velocity of the front vehicle (*individual information*), as well as (b) a prediction of the collision status in the upcoming intersection (*predicitve information*):

$$s = [vel, pos, fveh\_dist, fveh\_vel, col\_term, density] \tag{4.2}$$

Table 4.1: Typical levels of acceleration for various vehicle types.

Vehicle type	$Decel_L$ (m/s^2)	$Decel_M$ (m/s^2)	$Accel_M$ (m/s^2)	$Accel_L$ (m/s^2)
passenger cars	-5	-2.5	1.5	3
buses & trucks	-4	-2	0.75	1.5

4.2.2 Action space

When a vehicle follows a pre-defined path, the corresponding route-agent must control its acceleration so as to reach its destination safely. Thus, the vehicle's acceleration plays the role of the action and the task of autonomous navigation is to perform a sequence of actions in order to reach its destination in an optimal way. In the present work, five (5) levels of acceleration are considered (two values for deceleration and another two for acceleration) depending on the type of vehicle:

$$A = \{Decel_L, Decel_M, Zero, Accel_M, Accel_L\}$$

Table 4.1 presents the values of acceleration compatible per vehicle type as provided by the SUMO simulator.

4.2.3 Instantaneous and future reward function

The reward function provides an evaluation of how good is the selected action based on the current vehicle's state. In this study an efficient reward function is designed that aims at eliminating the collisions and minimizing the traveling time among vehicles.

The computation of the reward function initially examines whether or not the vehicle meets a terminal state, i.e. either reaches its destination or collides with other vehicles. In this case the agent receives a positive (success for finding goal) or negative (punishment for collision) constant reward L. It must be noted that our method did not show strong sensitivity to the choice of the reward parameter L during our experimental study. The following equation describes the main body of the reward function:

$$R(s,a) = \begin{cases} +L & , \text{ if it reaches goal} \\ -L & , \text{ if it collides} \\ r(s,a) & , \text{ otherwise} \end{cases} \tag{4.3}$$

In case of no terminal states, the reward function $r(s,a)$ depends mainly on two environmental features that can be observed: the collision term (col_term) and the

actual distance from the front vehicle ($fveh\_dist$). More specifically, there are two major cases:

- It is estimated that the vehicle will not participate in any collision at the next visited intersection ($col\_term = 0$), according to the collision term. Then,

  – if the distance from the front vehicle is greater than a *safety distance* threshold value ($fveh\_dist > safe\_dist$), the vehicle will receive a positive reward in a manner analogous to its velocity so as to promote the higher velocities. This is described by the following rule:

  $$r(s,a) = f(v) = \frac{vel}{vel_{max}} \ , \tag{4.4}$$

  that normalizes the velocity to $[0, 1]$, where vel_{max} indicates the maximum allowed velocity. Note that the value of $safe\_dist$ threshold varies according to the road length

  – if the front vehicle is nearby, $fveh\_dist < safe\_dist$, a negative reward is given that depends on both the normalized distance from the front vehicle ($f(d)$) and its current normalized velocity ($f(v)$). This is formulated as follows:

  $$\begin{aligned} r(s,a) &= -(c_1 + f(v)) * f(d) \\ &= -(c_1 + \frac{vel}{vel_{max}}) * \frac{safe\_dist}{\min(fveh\_dist, safe\_dist)} \end{aligned} \tag{4.5}$$

  where c_1 is a positive constant that takes a small value (it was set to $c_1 = 0.2$ in all experiments).

- It is estimated that the vehicle is going to participate in a collision at the next visited intersection ($col\_term = 1$), based on the collision term. Then, a negative reward will be received depending on the distance from the front vehicle:

  – if no other vehicle is in front ($fveh\_dist > safe\_dist$), then the received reward depends only on its velocity in a way of discouraging it from reaching the intersection quickly. This can be formulated as:

  $$r(s,a) = -c_2 - f(v) = -(c_2 + \frac{vel}{vel_{max}}) \tag{4.6}$$

  where c_2 is a positive constant equal to 5

– if there is another vehicle in front at a close distance ($fveh\_dist < safe\_dist$), the reward depends on both its velocity and the distance from the front vehicle, as given by:

$$\begin{aligned} r(s,a) &= -(c_3 + f(v)) * f(d) \\ &= -(c_3 + \frac{vel}{vel_{max}}) * \frac{safe\_dist}{\min(fveh\_dist, safe\_dist)} \end{aligned} \tag{4.7}$$

where c_3 is a positive constant equal to 0.4.

A last issue that must be noted is about the role of the three constants (c_1, c_2, c_3) in the above three equations (Eqs. 4.5-4.7) of the reward function. Obviously both equations 4.5, 4.7 have the same formulation since they refer to the case where there is another vehicle in close distance. They differ only on the value of their constant parameter: it is more important ($c_3 > c_1$) when the vehicle is also predicted to participate in a collision. On the other hand, the prediction of a collision, but without a vehicle in front, leads to a negative reward (Eq. 4.6) with an extra penalty denoted by the constant parameter c_2.

4.2.4 Algorithmic description

The proposed MAS introduces an efficient collaborative framework with two (2) main focuses. Firstly, it considers *route-agents*, enabling vehicles that follow the same route to cooperate in constructing an optimal shared policy. Secondly, it incorporates the *collision term* feature in the state, fostering cooperation among vehicles from different route-agents to avoid collisions and safely reach their destinations.

In summary, the proposed MARL scheme provides multiple advantages. It enables efficient management of large scale scenarios involving diverse types of vehicles, allowing for cooperation among them to prevent collisions at intersections. The scheme also enhances learning efficiency by leveraging the simultaneous use of multiple vehicles following the same route to train the corresponding agent. Finally, the constructed policies demonstrate robustness, and the scheme achieves faster convergence.

The overall scheme can be summarized in Algorithm 6.

Algorithm 6: Multi-agent DDQN for autonomous navigation in large-scale urban road networks

Input: number of iterations N, learning rate η, discount factor γ, episodes until update of target network N_u, scenario duration T

Create two DDQN networks (evaluation and target) for every *route-agent*

Initialize a replay buffer for every *route-agent*

for $i \leftarrow 1$ *to* N **do**

  while $t < T$ **do**

    for *every vehicle* i *found currently in the traffic zone* **do**

      Identify the corresponding route-agent j

      Obtain its state $s_{i,t}$ (Eq. 4.2)

      Choose an action $a_{i,t}$ following an ϵ-greedy strategy

      Move into the next state $s_{i,t+1}$ and receive a reward (Eq. 4.3)

      Save the experience into the replay buffer of the corresponding DDQN route-agent

    Update every route-agent's DDQN evaluation main using batches sampled from the replay buffer

  Every N_u episodes update the target network

Store the learned route-agents' policies

4.3 Simulation results

4.3.1 Implementation details

The proposed method, that will be mentioned from now on as *col-MARL*, was evaluated on several traffic scenarios of varying difficulty that they all concern road networks with unsignalized intersections. For each experimental case, multiple copies with random traffic conditions are created using the SUMO tools for obtaining the routes of the vehicles. Every copy has the same number of vehicles, but differs on the route plans. Moreover, the arrival rate of the vehicles was considered to be constant and equal to four (4) seconds in the case of the artificial scenarios, however, in the real scenarios variable arrival rates were used to test the robustness of the proposed method in more complex environments. Finally, the maximum velocity is set $40\ m/s$ and an initial velocity of $10\ m/s$ is applied to all vehicles entering the map.

For the Q-function approximation a DRL network is employed using the DDQN

algorithm, which consists of a main and a target network. The input state is represented as a vector of size six (6), capturing the individual and predictive information about the ego vehicle, and is connected to a single hidden layer with 128 nodes with hyperbolic tangent (tanh) activation function. This layer served as an intermediate representation, allowing the network to learn complex relationships between the input state and the desired actions. The output layer of the network consisted of five (5) nodes, corresponding to the available levels of acceleration.

During the learning process, the weights of the target network are updated every five (5) episodes with the current weights of the main network. Furthermore, the batch size was set to 2500 samples drawn from an replay buffer containing 25000 samples. The training is performed at the end of each episode with learning rate equal to 0.001 by using the Adam optimizer.

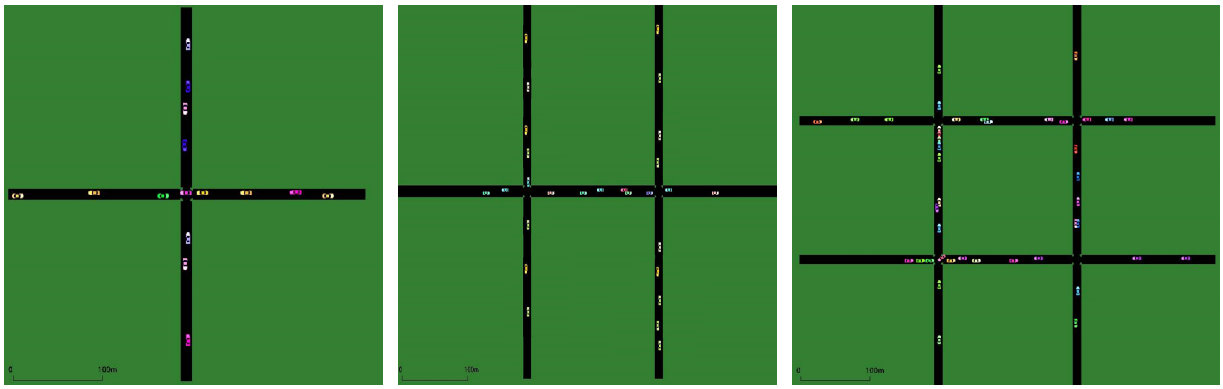
An ϵ-greedy exploration-exploitation strategy is proposed, where the probability is initialized to $\epsilon = 0.9$ and is gradually reduced at constant number of episodes. When ϵ becomes zero the exploitation phase begins. Moreover, in case of collision during the training process, the participating vehicles are removed from the traffic zone and the learning continues with the remaining vehicles. As experiments have shown this is beneficial for the optimization process.

The proposed col-MARL is compared against two methods:

- An independent MARL (i-MARL) framework where each agent learns its own policy independently. This approach considers only the individual information without taking into account the centralized terms, *collision and density*, as part of the state space. Furthermore, the reward function has been properly modified and considers only the distance from the front vehicle (Eqs. 3.2, 3.3).

- The *Intelligent Driver Model* (IDM) [82] which provides a strategy used in SUMO environment for intelligent-vehicle simulations and is considered to be one of the simplest and accident-free models producing realistic acceleration profiles. IDM's policy aims to reach the desired velocity by accelerating or braking depending on: (a) the current vehicle's velocity, and (b) the position and velocity of the leading vehicle immediately ahead. It must be noted that in order to make a fair comparison, the IDM shares the same safety settings as the proposed method (i.e., usnignalized intersections, no right-way rule, and so on).

Additionally, in all experiments a deterministic rule is utilized for all vehicles



(a) one-intersection (b) two-intersections (c) four-intersections

Figure 4.2: Snapshots of the SUMO simulator of three examples of traffic networks including different number of unsignalized intersections.

crossing their last road segment before exiting the traffic zone, where there is no longer any intersection in front of them. In that case, the maximum acceleration is applied until they reach their maximum velocity. As the experiments have shown, this is beneficial for the learning process and the quality of the produced driving policies.

At the end of the episode several useful statistics related to the vehicles' performance can be calculated, such as:

- **Average traveling time**: total time (in s) it takes for the vehicles to reach their destinations on average.
- **Average velocity**: mean velocity (in m/s) of all vehicles.
- **Average fuel consumption**: fuels (in ml/s) that were consumed by each vehicle on average.
- **Collisions**: number of collisions during the episode.

These quantities will be used next as evaluation metrics during the experimental study for measuring the performance of the proposed method.

4.3.2 Experiments with artificial road networks

The first series of experiments was conducted on three (3) artificial scenarios shown in Fig. 4.2 that consist of one (Sc_1), two (Sc_2) and four (Sc_4) intersections, respectively.

Table 4.2: Description of three artificial scenarios used in the experimental study

Scenario	**# vehicles**	**Duration** (sec)	**# roads**	**# routes (agents)**
Sc_1	150	600	8	12
Sc_2	300	1200	14	24
Sc_4	1000	4000	24	32

Table 4.3: Comparative results in three artificial scenarios.

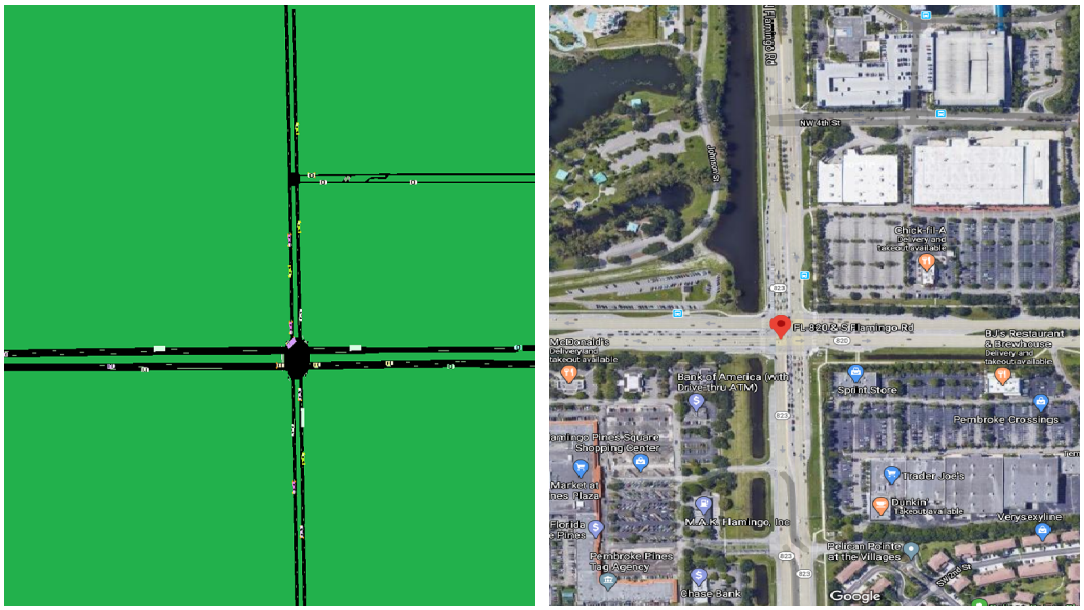
Scenario	**Traveling time** (sec)			**Velocity** (m/s)			**Fuel consumption** (ml/s)		
	col-MARL	**i-MARL**	**IDM**	**col-MARL**	**i-MARL**	**IDM**	**col-MARL**	**i-MARL**	**IDM**
Sc_1	13.9 ± 0.1	15.1 ± 0.2 *success* 100%	17.8 ± 0.1	25.0 ± 0.1	23.6 ± 0.1	16.8 ± 0.2	125.0 ± 1.7	126.7 ± 2.5	129.4 ± 1.2
Sc_2	17.8 ± 0.2	18.9 ± 0.4 *success* 70%	21.8 ± 0.3	29.4 ± 0.3	28.1 ± 0.1	23.6 ± 0.1	143.0 ± 0.5	143.7 ± 4.8	149.9 ± 3.9
Sc_4	20.9 ± 0.2	22.1 ± 0.6 *success* 40%	24.2 ± 0.5	30.8 ± 0.4	28.7 ± 0.6	24.2 ± 0.4	155.7 ± 1.7	159.1 ± 3.3	168.8 ± 3.9

Their characteristics are shown in Table 4.2, and the results are presented in Table 4.3, in terms of the statistics (mean value and standard deviation) of three (3) evaluation metrics that have been calculated by 20 individual experiments per type of scenario.

According to the results, one can observe the capability of the proposed *col-MARL* method to construct efficient policies and resolve the scenarios with higher mean velocities and less traveling time compared to the other methods. This also demonstrates the effectiveness of the proposed reward to boost higher velocities (whenever safe). Specifically, in the difficult scenario case consisting of four (4) intersections (Sc_4), the mean value of velocity of all vehicles is higher ($30.8\ m/s$) in comparison to the one achieved by the *i-MARL's* policies ($28.7\ m/s$) and the *SUMO's IDM* ($24.2\ m/s$). As a result, the average traveling time obtained by the proposed method is quite improved.

The weakness of the *i-MARL* approach (non-collaborative version) is its inability to generate successful policies and its failure to resolve all collisions, especially in complex scenarios. It must be noted that Table 4.3 reports only the statistics for the successfully resolved cases for the *i-MARL* method. The results are in agreement with our initial belief about the significant role that the *collision term* plays to the performance of the proposed method and to the development of a collaborative multi-agent environment. Finally, as it can be observed, the proposed method is able to achieve improved average fuel consumption in comparison to the SUMO's IDM.

4.3.3 Experiments with real urban road networks



(a) SUMO Map (b) Original Map from Google Maps

Figure 4.3: Snapshots from the SUMO simulator and Google Maps of the real scenario in Florida.

Further evaluations of the proposed method were conducted using real-world scenarios. For this purpose, it was selected a traffic domain located in Pembroke Pines, Florida, and specifically the intersection of Pines Boulevard and South Flamingo Road, as shown in Figure 4.3. This road network consists of a main intersection along with a smaller one in the northern part of the map. There are a total of 12 roads and 20 *route-agents*. Each road is equipped with a minimum of three (3) lanes, with the main intersection having a maximum of seven (7) lanes. This traffic domain was specifically chosen for its level of difficulty, as it is recognized as one of the most dangerous intersections in the United States [1].

The original map was used to create three (3) different types of noisy scenarios that differ on the level of variability over the arrival rate of the incoming vehicles. More specifically, by assuming an additive zero-mean Gaussian noise on the arrival rate of individual vehicles, ten (10) copies of each one of the following types of scenario were generated:

- CR scenario with constant arrival rate (1 vehicle / $4s$)

[1] https://www.chaliklaw.com/news/most-dangerous-intersections-in-broward-county/

- MR scenario with medium variable arrival rate using a Gaussian noise of standard deviation equal to 0.5
- LR scenario with large variable arrival rate using a Gaussian noise of standard deviation equal to 1.0

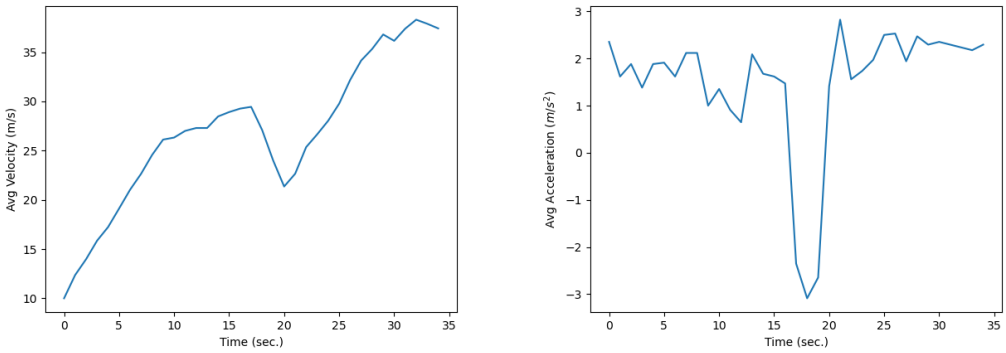
In this series of experiments three different types of vehicles are considered: "passenger cars", "buses" and "trucks", with a ratio of 75:15:10, that have different characteristics (length, acceleration, deceleration, etc.). During training, the duration of the road traffic scenarios was 20 minutes and approximately 300 vehicles were used at every scenario case.

Table 4.4: Comparative results of the proposed method and the SUMO's IDM on three noisy versions of the real scenario shown in Fig. 4.3.

	Traveling time (sec)		**Velocity** (m/s)	
Scenario	**col-MARL**	**IDM**	**col-MARL**	**IDM**
CR	51.5 ± 0.4	54.1 ± 0.8 *success* 80%	30.2 ± 0.2	26.4 ± 0.9
MR	50.8 ± 0.3	53.8 ± 0.7 *success* 60%	31.0 ± 0.3	26.5 ± 0.2
LR	49.2 ± 0.3	54.0 ± 0.0 *success* 10%	31.4 ± 0.2	26.2 ± 0.0

The results using the evaluation metrics of *average traveling time*, *average velocity* and *success rate* are presented in Table 4.4 for the proposed *col-MARL* method and the *SUMO's IDM*. No results from the *i-MARL* method are shown since it was unable to successfully resolve any scenario case. It is interesting to observe the capability of the *col-MARL* method to successfully resolve all types of scenarios without being affected by the level of noise. On the other hand, *IDM* seems to reveal weak performance in environments with variable arrival time of vehicles. Moreover, the existence of more diversity in the characteristic of vehicles (different types, variable number of lanes) seems to not affect the proposed framework to produce efficient policies and flexible driving behaviors.

In Fig. 4.4, two diagrams are provided that illustrate the average velocity and acceleration (agent's action) of vehicles following a specific route with one intersection, based on the learned policy of the corresponding route-agent in Florida's map



(a) Avg. velocity

Avg. acceleration (b)

Figure 4.4: Average velocity and acceleration (action) of all vehicles that follow a specific route as obtained from the learned policy. The valleys shows the driving behavior before and after the intersection.

(Fig. 4.3). The observed pattern in both signals shows a valley, indicating the vehicles' arrival at the intersection. At this point, the agent selects appropriate levels of deceleration as actions according to its learned policy. As a result, the vehicles' velocity decreases as they approach the intersection, as depicted in the left diagram of Fig. 4.4(a). This strategy ensures safe traversal of vehicles by avoiding collisions and significantly enhances the traffic throughput.

4.3.4 Experiments with complex urban road networks

Additional experiments were made by evaluating the performance and the scalability of the proposed method to large scale scenarios. For that purpose, one of the busiest traffic areas in Athens, Greece (Omonoia square - city center), was chosen. Figure 4.5 illustrates a snapshot of the SUMO simulator along with the original map taken by the Google maps. The road network consists of over 60 intersections of various types (2-way, 3-way and 4-way), 190 roads and nearly 150 *route-agents*. Each road is of variable length and contains from one (1) to four (4) lanes.

An important feature of the SUMO simulator that adds extra stochasticity to the scenarios is the hard-coded lane-change controller. According to this, the vehicles are allowed to change lanes when SUMO considers it appropriate, without affecting their pre-defined path. Finally, the vehicles can not exceed a maximum velocity of 30 m/s

(a) SUMO Map

(b) Original Map from Google Maps

Figure 4.5: Snapshots from the SUMO simulator and Google maps of the real scenario around the area of Omonoia Square, Athens.

that obliges with the city rules.

Two versions of the scenario were designed in terms of the arrival rate of vehicles variability:

- LS_{const} scenario with constant arrival rate of vehicles, where the"passenger cars", "buses" and "trucks" appear every 4/10/15 seconds, respectively.
- LS_{noisy} scenario using a zero-mean Gaussian noise with standard deviation equal to 0.2 on the arrival rate of all vehicles.

Both road traffic scenarios had a duration of 30 minutes, involving a total of 750 vehicles. The distribution of vehicle types was as follows: 75% passenger cars, 15% buses, and 10% trucks. To ensure robust evaluation, ten (10) different copies of each scenario were generated and assessed.

Table 4.5 presents the obtained statistical results according to three (3) evaluation metrics: *average traveling time*, *average velocity* and *success rate*. SUMO's IDM was unable to resolve both scenarios in almost all cases. Also, results from the *i-MARL* method are not reported since it was unable to provide solutions without collisions. On the other hand, the proposed method showed very promising performance in terms of scalability in networks of high complexity, as it was able to successfully resolve all test scenarios (100% success), even in situations with variable arrival rate. Furthermore, the effectiveness of the reward function can be detected on the *average velocity* column,

Table 4.5: Comparative results on the large scale scenario of Fig. 4.5

Scenario	**Traveling time** (sec)		**Velocity** (m/s)	
	col-MARL	**IDM**	**col-MARL**	**IDM**
LS_{const}	42.6 ± 0.4	48.1 ± 0.0 *success* 10%	24.9 ± 0.5	19.7 ± 0.0
LS_{noisy}	42.3 ± 0.3	$-$ *success* 0%	25.1 ± 0.4	$-$

where the results reveal the tendency of the method to serve the vehicles with high velocities.

4.4 Knowledge reuse effect of the proposed multi-agent reinforcement learning system

This section presents the capability of the proposed method to successfully offer knowledge reuse for autonomous navigation tasks in environments with unsignalized intersections. In this case, knowledge is represented by the learned policies of the route-agents which can be (re)used on vehicles in unknown environments. Several copies of the original scenarios were considered that follow the same generation mechanism used in the training phase. These copies serve as the unknown scenarios and are accompanied with increased "number of vehicles" and "duration", in an attempt to test the effectiveness of the learned agents' policies to successfully transferring knowledge, and to measure their generalization capabilities.

Every test case produced was then evaluated by each one of the learned policies of route-agents that were produced during the training phase, according to the following procedure: Before entering the (unknown) traffic zone, each vehicle chooses the appropriate route-agent to follow based on its destination path. The selected agents are then responsible for navigating safely the "acquired" vehicles and appropriately controlling their acceleration. This is made by employing sequentially the established agent's policy to choose the actions that yield the highest expected return given the current vehicle's state.

At first, the knowledge reuse was studied in unknown traffic domains using the

Table 4.6: Description of the evaluation cases utilized for knowledge reusing.

Evaluation case	**Description**	**#vehicles**	**Duration** (sec)
$Sc_1(500)$	One Intersection	500	2000
$Sc_1(1000)$		1000	4000
$Sc_2(1000)$	Two Intersections	1000	4000
$Sc_2(2000)$		2000	8000
$Sc_4(2000)$	Four Intersections	2000	8000
$Sc_4(5000)$		5000	20000

artificial scenarios of Fig. 4.2. In particular, 20 copies for every one of the three (3) original scenario profiles were generated with variable degree of difficulty: ten (10) cases with a medium number of vehicles and another ten (10) cases with much larger number of vehicles (and duration). Their specific characteristics are presented in Table 4.6.

Table 4.7: Comparative statistical results over various evaluation cases using three evaluation criteria.

Evaluation case	**Traveling time** (sec)		**Velocity** (m/s)		**Fuel consumption** (ml/s)	
	col-MARL	**IDM**	**col-MARL**	**IDM**	**col-MARL**	**IDM**
$Sc_1(500)$	13.8 ± 0.3	18.1 ± 0.1	25.0 ± 0.1	23.4 ± 0.2	125.4 ± 2.3	130.0 ± 1.9
$Sc_1(1000)$	13.7 ± 0.4	17.8 ± 0.3	25.2 ± 0.4	23.6 ± 0.3	125.9 ± 2.6	129.8 ± 1.2
$Sc_2(1000)$	18.0 ± 0.2	22.5 ± 0.4	29.3 ± 0.3	27.2 ± 0.2	142.1 ± 1.2	149.8 ± 3.6
$Sc_2(2000)$	17.7 ± 0.1	22.9 ± 0.2	29.7 ± 0.2	26.9 ± 0.0	143.0 ± 1.8	150.6 ± 1.3
$Sc_4(2000)$	20.6 ± 0.3	24.3 ± 0.2	31.6 ± 0.4	28.2 ± 0.3	155.3 ± 1.9	169.2 ± 1.1
$Sc_4(5000)$	20.7 ± 0.4	24.8 ± 0.1	31.5 ± 0.5	27.6 ± 0.5	154.9 ± 2.1	171.4 ± 3.2

Comparative results are shown in Table 4.7 in terms of the statistics (mean value and standard deviation) of the three (3) evaluation metrics calculated from $10(copies) \times 20(learned\ policies) = 200$ trials per evaluation case (test environment).

According to the results, in all cases the episodes were terminated without any collision showing the capability of the proposed method to efficiently offer knowledge reuse. It successfully performs transfer learning and maintains its decision-making policies to unknown environments. This can be seen by comparing the results of Tables 4.3 and 4.7 concerning the training and the evaluation phases, respectively, where all measurements have almost the same values. An interesting observation

Table 4.8: Contingency table analysis of the three types of learned policies found by the proposed method in the real scenario of Fig. 4.3. The statistical results are taken using evaluation cases of various level of noise.

Evaluation case	*policies of* CR			*policies of* MR			*policies of* LR			*IDM*
	Traveling time (sec)	**Velocity** (m/s)	**Succ.** (%)	**Traveling time** (sec)	**Velocity** (m/s)	**Succ.** (%)	**Traveling time** (sec)	**Velocity** (m/s)	**Succ.** (%)	**Succ.** (%)
$CR(1h)$	51.6 ± 0.1	30.4 ± 0.3	100	50.4 ± 0.3	30.9 ± 0.2	100	49.6 ± 0.3	31.1 ± 0.4	100	62
$MR(1h)$	51.4 ± 0.2	30.0 ± 0.4	90	50.8 ± 0.1	30.4 ± 0.2	100	49.3 ± 0.1	31.0 ± 0.1	100	7
$LR(1h)$	51.3 ± 0.2	30.0 ± 0.5	74	50.1 ± 0.4	30.9 ± 0.4	92	49.1 ± 0.3	31.3 ± 0.2	100	0

concerns its ability to maintain its performance when using scenarios with significantly larger number of vehicles and duration. As an example, in the obtained results from the case of traffic networks with four intersections (SC_4), both training (1000 vehicles) and evaluation cases (2000 or 5000 vehicles), support the previous claim.

In comparison with the *SUMO's IDM* driving model, the proposed *col-MARL* method achieves solutions with significantly increased *average velocity* and thus lower *traveling time*. Notice that the performance of the *i-MARL* method is not presented, due to its inability to resolve the scenarios without producing collisions.

Further experiments on knowledge reuse were made using the real scenario of Fig. 4.3. Following the same procedure as in the case of artificial scenarios, ten (10) simulated copies of the following types of scenarios were created:

- $CR(1h)$: one (1) hour traffic with constant arrival rate (1 vehicle $/4s$)
- $MR(1h)$: one (1) hour traffic with medium variable arrival rate of vehicles using a zero-mean Gaussian additive noise of standard deviation equal to 0.5
- $LR(1h)$: one (1) hour traffic with large variable arrival rate of vehicles using a zero-mean Gaussian additive noise of standard deviation equal to 1.0

In all cases more than 900 vehicles were generated. Again, different types of vehicles were considered with the same analogy as in training: 75% "passenger cars", 15% "buses", and 10% "trucks".

The obtained results are presented in a contingency table design, in Table 4.8. This shows the performance of the learned policies of the proposed method (10 policies per case) to the above evaluation cases (10 copies per case). The reason of adopting this contingency table analysis is to study the sensitivity of the learned policies to the noise of the arrival time of vehicles in the traffic domain.

According to the results, the learned policies of the LR scenarios (with large noise on the arrival rate) showed the best performance, since they managed to successfully handle all evaluation cases, independently of the level of noise they contained. Comparing the results of both Tables 4.4 and 4.8, it can be observed that the values of the *average traveling time* and *velocity* of vehicles were maintained in the same (high) level as those reached during the training procedure. This is of great importance since it shows the ability of the proposed MARL scheme to avoid overfitting and to yield a superior level of generalization to environments with or without the presence of noise.

On the other hand, the learned policies of medium noise arrival time scenarios (MR) had only a small failure rate of 10% in tested scenarios of higher noise ($LR(1h)$), while all other scenarios of equal or smaller noise were resolved. As it was expected, the worst performance is being observed on the constant rate learned policies (CR) as they are unable to generalize well to unknown scenarios with noise. Moreover, *IDM's* performance deteriorated in these experiments as it resolved 62% of the constant rate cases, while resolving any other type of scenario with additive noise was almost impossible (7% and 0% success rate, respectively).

Table 4.9: Contingency table analysis of the two types of learned policies found by the proposed method in the large scale real scenario of Fig. 4.5

Evaluation case	*policies of* LS_{const}			*policies of* LS_{noisy}			*IDM*
	Traveling time (sec)	**Velocity** (m/s)	**Succ.** $(\%)$	**Traveling time** (sec)	**Velocity** (m/s)	**Succ.** $(\%)$	**Succ.** $(\%)$
$LS_{const}(1h)$	43.0 ± 0.3	24.3 ± 0.1	100	42.5 ± 0.2	24.9 ± 0.3	100	0
$LS_{noisy}(1h)$	43.5 ± 0.5	24.1 ± 0.3	80	42.6 ± 0.3	24.9 ± 0.2	100	0
$LS_{const}(2h)$	43.1 ± 0.2	24.2 ± 0.2	100	42.5 ± 0.3	25.0 ± 0.3	100	0
$LS_{noisy}(2h)$	43.3 ± 0.4	24.1 ± 0.2	60	42.4 ± 0.2	25.1 ± 0.1	100	0

A final set of experiments on knowledge reuse was made using the large scale real scenario of Fig. 4.5. Ten (10) simulated copies of the following types of scenarios were created:

- $LS_{const}(1h)$: one (1) hour traffic with constant arrival rate: "passenger cars", "buses" and "trucks" appear every 4/10/15 secs. In total 1500 vehicles were used.

- $LS_{noisy}(1h)$: one (1) hour traffic with variable arrival rate of vehicles using a

zero-mean Gaussian noise with standard deviation equal to 0.2. In total 1500 vehicles were used.

- $LS_{const}(2h)$: two (2) hour traffic with constant arrival rate (4/10/15) in all vehicles. In total 3000 vehicles were used.

- $LS_{noisy}(2h)$: two (2) hour traffic with variable arrival rate of vehicles using a zero-mean Gaussian noise with standard deviation equal to 0.2. In total 3000 vehicles were used.

The performance of the learned policies of the proposed method (10 policies per case) to the unknown scenarios is presented in Table 4.9. Based on the statistical results, the learned "noisy" policies had more generalization properties as they successfully resolved all cases without collisions. As it was expected, the learned "constant" policies failed in some of the noisy cases since training scenarios did not cover noisy examples. On the other hand, *IDM* completely failed in all evaluation cases. It is interesting to observe, once again, that even in such demanding evaluation cases the reused policies did not decline much from the training solutions in terms of *average traveling time* and *average velocity*.

Supplementary results of the proposed method can be found in a related google drive address [2], where several simulation runs on both real-word scenarios are presented under various traffic cases using SUMO GUI.

4.5 Summary

This chapter proposes a deep MARL approach for addressing autonomous vehicles' navigation in traffic environments with unsignalized intersections. A novel perspective to the problem is presented by treating the routes as the agents, while the effectiveness of the proposed method is assessed in several large-scale real-world traffic scenarios of varying difficulty containing large number of vehicles and variable level of noise. The results were very promising as the proposed method achieved both its objectives: (a) navigate safely all the vehicles to their destinations avoiding collisions, and (b) minimize the traveling time.

[2] https://drive.google.com/drive/folders/1qJ2H1QFfuF9QtHrPMJty1yof97H1kVjY

The proposed approach is compared against SUMO's IDM car-following model and an i-MARL framework, where the experimental analysis demonstrated that it out-performs both since it manages to drastically reduce the traveling time while ensuring safety.

Ultimately, a way of transferring the obtained agents' knowledge to new agents in unknown traffic environments is displayed. The learned policies are tested to more demanding cases of traffic, and it is confirmed that they are able to resolve them without the need of additional training. Moreover, the results from knowledge reuse indicate that the solutions it provides are of high quality and comparable to the results obtained during training.

Chapter 5

Hierarchical Multi-Agent Reinforcement Learning for Managing Air Traffic Congestion

This chapter studies the resolution of imbalances between demand and capacity in the air traffic management (ATM) domain. The problem is initially formulated as collaborative MAMDP, where several MARL methods are provided. Then, an extension through the prism of hierarchical reinforcement learning (HRL) is presented, where a method leveraging state abstraction is proposed.

The proposed approach is evaluated on a real-world scenario containing flights from Barcelona to Madrid and is compared against alternative hierarchical schemes.

The obtained results show the robustness of the hierarchical framework providing solutions that eliminate the congestion problems in the air traffic domain.

5.1 Overview

In a multi-agent environment, congestion problems arise when limited resources need to be shared among multiple agents simultaneously. Such problems are prevalent in various domains of our modern world and greatly affect our businesses, daily activities, and overall lives. In the domain of ATM, congestion problems occur when the demand for airspace use exceeds its capacity, resulting in overcrowded areas known as *hotspots*. This situation is referred to as **Demand and Capacity Balance (DCB)** problem.

The DCB issues are typically addressed through airspace and flow management solutions, which include regulatory measures that introduce delays to the involved flights. However, these delays can propagate throughout the system, introducing uncertainty and increased operational costs. One of the biggest challenges is to handle demand-capacity imbalances during the pre-tactical phase (i.e., before flights' take-off), due to the limited available operational information at that time.

In such cases, MARL has emerged as a promising framework to address a wide range of problems, allowing multiple autonomous agents to learn in a decentralized manner, while interacting within a shared environment. In this chapter, it is discussed the task of resolving *demand-capacity imbalances* in the ATM domain at the pre-tactical stage of operation. Under this setup, the flights are considered to be the agents of the MAS, and their goal is to coordinate their joint actions (i.e., ground delays) in order to resolve congestions in which they participate.

To tackle this challenge, the DCB problem at pre-tactical stage of ATM operations is formulated as a collaborative MAMDP. In this context, the airspace is divided into *air sectors*, and the limited *capacity* of these sectors accounts for the necessity of operational constrains. Hence, the goal is to minimize the scheduled flight delays and associated delay costs. This is achieved by introducing a collaborative MARL (CMARL) algorithm to solve the aforementioned MAMDP.

However, the large number of flights per day in European airspace, along with the numerous delay options per flight available to resolve DCB problems, results in

an exponentially increasing state-action space. To address this issue and improve computational efficiency, the use of *abstraction* or *aggregation* techniques is commonly employed in the field of machine learning.

Abstraction can be applied in both the state and the action space. In the state space, decision-makers can find solutions more quickly by operating in an abstract state space, where groups of states are treated as a single unit, and ignoring unnecessary state information. Similarly, in the action space, also known as *temporal abstraction*, decision-makers consider high-level actions composed of multiple lower-level actions. Thus, abstraction can appear simultaneously in the state and action spaces, or in only one of them.

Delving more into the structure of the DCB issues and considering a hierarchical decomposition of the problem, a general hierarchical collaborative framework is proposed that allows several hierarchical schemes to be introduced. This leads to the incorporation of abstraction schemes in state and/or action spaces, facilitating the creation of multiple policies at different levels of abstraction. Also, this study introduces a hierarchical method that extends CMARL and employs *state abstraction*. By leveraging the capabilities of state abstraction, agents can effectively explore the original (ground) space more efficiently, leading to the discovery of higher quality refined solutions.

To evaluate the effectiveness of the proposed method, multiple experiments are conducted on real-world cases involving thousands of agents, representing aircraft following specific trajectories. The results demonstrate the efficacy of the hierarchical framework in effectively addressing congestion problems in the ATM domain.

The contributions can be summarized as follows:

- The DCB problem is initially formulated as a collaborative MAMDP. Subsequently, it is extended to the hierarchical case, incorporating multiple levels of abstraction.
- A generic hierarchical MARL framework capable of functioning at multiple levels of abstraction is presented.
- The proposed hierarchical framework can be instantiated to various hierarchical schemes, exploiting multiple levels of abstraction, both at the action and the state space.

- The performance of the proposed HCMARL is evaluated using real-world scenarios that involve a significant number of flights against other hierarchical methods.

This chapter is structured as follows. It begins with a brief review of previous works in Section 5.2, and in Section 5.3 the DCB problem in the ATM domain is presented. Furthermore, Section 5.4 discusses MARL approaches for ATM, while Section5.5 proposes a hierarchical framework with multiple levels of abstraction. Finally, Section 5.6 presents the evaluation cases and experimental results, and Section 5.7 concludes the chapter with a small discussion.

5.2 Related work

Congestion problems have been extensively investigated in various domains including game theoretic models [83, 84, 85, 86], optimization, transportation, automatic control [87], and autonomous agents [88, 89]. These interdisciplinary efforts have contributed to a deeper understanding and effective solutions for managing congestion issues. Moreover, many MARL approaches [90, 91, 92, 93] have been proposed over the years for the resolution of congestion problems.

Hierarchical RL has been the subject of extensive research for a considerable period of time. Within the literature, several early works have laid the foundation to this field, including hierarchies of abstract machines (HAM) [94], the "option" framework [95], the feudal networks [96] and the MAX-Q learning method[97]. Specifically, in [96], the authors propose a hierarchical approach where high-level managers assign tasks to sub-managers, who then learn to fulfill those tasks. Following, in [95] the "options" framework is introduced, which leverages temporal abstractions in the action space and extends the standard Markov Decision Process formulation to a semi-Markov Decision Process (SMDP). In this framework, agents have the ability to select either primitive actions or multi-step actions, known as "options", where each option is defined by a policy over actions and a termination function. The MAX-Q algorithm is utilized in [97], where a hierarchical reinforcement learning method is developed based on decomposing the value function into combinations of value functions.

The authors in [98] extend the "options" framework to address partially observable markov decision processes (POMDP). To solve it, they leverage state abstraction

and develop a hierarchical Monte Carlo tree search (MCTS) algorithm. Building upon the "options" framework, in [99] the authors propose the option-critic approach, which uses a policy gradient method so as the agent to learn options autonomously. Moreover, in [100] the option-critic framework is further expanded, by introducing a generalized reinforcement learning architecture capable of learning options with more than two levels of abstraction. In [101], the authors propose a framework that incorporates intrinsic behaviors and introduces a meta-controller responsible for determining sub-goals.

In the work of [102], the authors focus on unmanned vehicle swarms with multiple objectives and introduce a hierarchical reinforcement learning algorithm called dynamic domain reduction for multi-agent planning that simultaneously explores sub-environments and generates action sequences that maximize expected rewards. Finally, more recent works try to learn the temporal abstraction with deep learning [103, 104, 105].

The objective of the present study is to bridge the previously described approaches by developing a general framework for hierarchical MARL. While the focus is on the ATM domain, the framework is designed to be applicable to multiple objective domains. The main goals of this study are twofold: (a) create a generic multi-agent framework that supports abstractions at multiple levels using different abstraction methods, and (b) allow agents to engage in coordination to resolve common challenges. To evaluate these objectives, multiple experiments are conducted to assess their potential in providing effective solutions to congestion problems in the ATM domain.

5.3 The demand and capacity balance problem

Nowadays, the nature of ATM domain presents challenges with demand-capacity imbalances, which are typically addressed through delays before take-off and increased costs for all parties involved. The objective is to design optimal traffic flows that align with air traffic control (ATC) capacity, while ensuring the safe and efficient operation of flights for airlines.

The task of addressing demand-capacity imbalances in ATM has been extensively discussed and defined in [89, 106]. Following is a brief overview of the DCB problem

as defined in the aforementioned papers. Two key components generally define the problem: *airspace sectors* and *aircraft trajectories*.

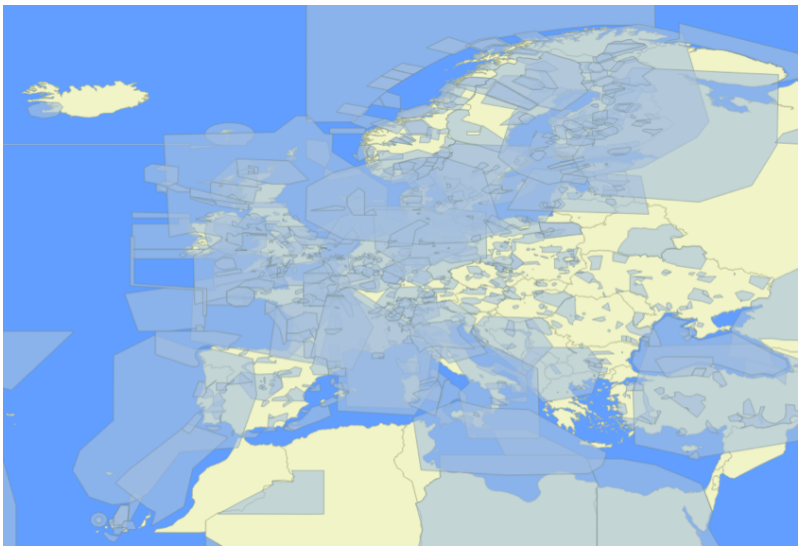


Figure 5.1: An example of airspace sectors in 2D.

Sectors play a crucial role in dividing the airspace into distinct volumes, effectively segregating the airspace. They can be understood as groups of blocks in the airspace, which are defined by their geometry. The airspace sectorization can vary depending on the sector configuration and the number of active sectors. Throughout a single day, the sectorization of airspace can change frequently to accommodate different operational conditions and requirements, but only one sector configuration can be active at any time. The most crucial aspect of a sector is its *capacity*, which represents the maximum number of flights that can pass through a sector within a specific time period. It is important to ensure that the *demand* for each sector, which refers to the number of flights that want to pass-through the sector, should not exceed its *capacity* at any given time.

To calculate the demand for each sector, various measures can be utilized, but this study focuses on the *entry count*, which quantifies the number of flights entering the sector during a particular time period and is used by the network managers at pre-tactical stage. In the experiments, the time period for measuring the *entry count* is set to one (1) hour.

Typically, *aircraft trajectories* are described as sequences of spatio-temporal tuples containing longitude, latitude, altitude, and timestamp information. In the context of congestion problems, though, trajectories can be also represented as time series

of events that indicate the utilization of shared resources such as sectors. This may include a description of the entry and exit locations of the aircraft (e.g. coordinates and flight levels), entry and exit times, or anticipated time at which the aircraft will pass through a specific sector.

In this chapter, the focus is on congestion problems that arise from the capacity limitations of sectors (resources) and the imbalances between demand and capacity for these shared resources. The objective is to address situations where the demand exceeds the maximum allowed capacity, leading to violations which are known as *hotspots*.

In order to address DCB issues, agents (flights) have to adjust their resource utilization schedule by imposing *delays* before the execution of their trajectories. This means that agents can shift the entire timetable for utilizing the required resources by a specific time duration. Consequently, the agents must cooperate and coordinate in order to learn the appropriate joint delays that they have to apply to their trajectories, while considering the operational constraints related to the capacity of the required resources.

Hence, the objective can be re-formulated as to effectively resolving all *hotspots*, by finding optimal solutions to the DCB problem while minimizing the total delay imposed on flights (i.e., the cumulative delay across all flights) as well as the average delay (i.e., the ratio of total delay to the number of flights) in relation to the number of delayed flights.

Nevertheless, by imposing delays on trajectories, it is possible for congestion problems to be shifted in different time periods and sectors, and generate new congested situations. Agents that partecipate in the same hotspot can be regarded as *peers*, since they need to collaboratively form a strategy that resolves the congestion. This suggests that agents form *neighborhoods* consisting of interacting peers. Obviously, the composition of interacting trajectories within a neighborhood may change as congestion problems propagate and get resolved. Therefore, it becomes necessary to dynamically update the neighborhoods of agents that execute interacting trajectories, as agents make decisions regarding different delays. All the above contribute to the utilization of a graphical representation in order to model societies of agents $\mathcal{S} = \{\mathcal{T}, \mathcal{AG}, \mathcal{E}\}$. Under this representation, every node of the graph corresponds to an agent $A_i \in \mathcal{AG}$ executing a trajectory $T_i \in \mathcal{T}$ and any edge $(A_i, A_j) \in \mathcal{E}$ signifies that those agents are *peers* and must coordinate their actions to resolve the common *hotspot* in which they

participate. All agents connected to an agent A_i through an edge in the graph belong to the *neighborhood* of A_i (denoted as $N(A_i)$).

Moreover, for each agent A_i, there exist a maximum allowed value for delay (in minutes), denoted as $MaxDelay_i$. This range can take values in $D_i = \{0, \ldots, MaxDelay_i\}$. It is important to note that the maximum preferred delay may differ for each flight. However, in this study, it is assumed that all agents share the same $MaxDelay$ value and do not have any specific preferences regarding delays, apart from wanting to decrease their own delay and the total delay of the multi-agent system.

5.4 Multi-agent reinforcement learning structures for ATM

The DCB process in ATM is formulated as a MARL problem, where the flight-agents operate in the same environment and share common resources (sectors). To address this problem, the MAMDP is considered as the underlying model, which has been thoroughly described in Chapter 2.2.5.

The *local state* of agent A_i is denoted as s_i and includes two major quantities:

- the delay imposed on trajectory T_i executed by flight-agent A_i, denoted as d_i. This value ranges within the set of possible delay actions in D_i,
- the number of hotspots in which A_i is involved, denoted as h_i.

$$s_i = \{d_i, h_i\} \tag{5.1}$$

Moreover, two agents, A_i and A_j, that belong to the same *neighborhood* share a *joint state*, s_{ij}, that consists of the concatenated local state variables of both agents.

The *local action*, a_i, of each flight-agent A_i decides about the *delay*. It is a binary decision concerning whether the agent should add or not one (1) more unit of time (i.e., minute) in its total delay. The agent is responsible for distributing delay units to the corresponding flight until it departs from the airport. Afterwards, the agent flies in its pre-defined trajectory without the possibility to receive additional delay or to "communicate" with other agents to resolve imbalances.

The most difficult choice in many real-world RL scenarios concerns designing an effective reward function that promotes desirable behavior. In the case of the DCB problem, it has been developed an individual delay reward, denoted as r_i, for each flight-agent A_i. This reward is based on the agent's involvement in hotspots while

executing its trajectory, taking also into consideration the agent's chosen delay. The formulation of this reward function is given by the following equation, which was based on our previous studies [89, 106]:

$$r_i(s_i, a_i) = C(s_i, a_i) - \lambda \times DC(s_i, a_i), \tag{5.2}$$

where C captures the agent's participation in hotspots, and DC relates to the cost of the delay for a given flight agent A_i. The parameter λ plays a crucial role in balancing the trade-off between the cost of participating in hotspots and the cost of imposing ground delays.

Specifically, in the proposed formulation, the function C measures the total duration that the agent is projected to stay inside the congested sectors. The exact relationship between the function C and this duration is defined as follows:

$$C(s_i, a_i) = \begin{cases} 81 \times TDC, & \text{if } TDC > 0 \\ C_+, & \text{if } TDC = 0 \end{cases}, \tag{5.3}$$

where TDC represents the total duration of congestion (hotspots) experienced by agent A_i. When the agent is not predicted to participate in any hotspots, then the value of TDC is equal to 0, and the agent receives a large positive constant C_+ as a reward. The coefficient 81 corresponds to the average strategic delay cost per minute in Europe (measured in Euros) when 92% of the flights do not experience any delays [107].

On the other hand, the DC function represents the delay cost incurred when flights are delayed at the gate. This cost is determined solely based on the number of minutes of delay given so far (d_i) and the type of the aircraft. The following formulation has been utilized to calculate the delay cost:

$$DC(s_i, a_i) = SDC(d_i, at_i), \tag{5.4}$$

where SDC is a function that calculates the strategic delay cost based on the specific aircraft type of agent A_i, denoted as at_i. It is important to note that in the general case, DC could incorporate additional airline strategic policies and considerations regarding flight delays.

5.4.1 Multi-agent independent reinforcement learning

The simplest multi-agent approach to solve the MAMDP is called *multi-agent independent reinforcement learning (MIRL)*. In this framework each agent learns its policy

independently, and considers all other agents part of the environment. The variant of Q-learning algorithm, as described in [108], assumes that the (global) Q-function is a linear combination of (local) Q-functions:

$$Q(s,a) = \sum_{i=1}^{|\mathcal{AG}|} Q_i(s_i, a_i), \tag{5.5}$$

where local Q-values are adjusted according to the basic Q-learning update rule:

$$Q_i(s_i, a_i) = (1-\eta)Q_i(s_i, a_i) + \eta \left[r_i + \gamma \max_{a^*} Q_i(s_i', a^*) \right] \tag{5.6}$$

Following [108], the above equation uses the *global reward* which depends on the global state and global action of the agents. However, a simplified version can be adopted by considering the *local reward*, r_i, in the update rule.

5.4.2 Collaborative multi-agent reinforcement learning

The proposed *collaborative multi-agent reinforcement learning (CMARL)* approach capitalizes on the problem's structure, and specifically on the interactions between flights. This approach considers that the agents do not possess knowledge of the transition model and interact concurrently with all their *peers*.

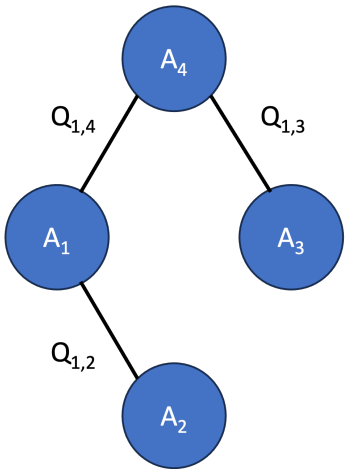


Figure 5.2: An example of a coordination graph between 4 agents.

Using the notion of *coordination graphs* [109], one can model the interactions between agents in a MAS. In such graphs, the agents are represented as nodes and neighboring agents are connected through edges. The concept of coordination graphs offers a method to decompose a complex multi-agent Q-function. Instead of relying

on a single joint Q-function that is dependent on the joint action of all agents, coordination graphs utilize hypergraphs to break down this Q-function into a collection of lower-dimensional Q-functions associated with the edges of the hypergraph. Each edge represents a distinct Q-function and by passing messages along the edges of the coordination hypergraph, the process of finding the optimal joint action can be achieved. This decomposition approach allows for more efficient computation and coordination among agents in complex MAS. Figure 5.2 illustrates an example of a coordination graph composed of four (4) agents. As an instance, for the DCB problem, agent A_1 must coordinate its actions with agents A_2 and A_4 in order to resolve their common imbalances. Notice that agents A_2 and A_4 are not connected with an edge in the coordination graph, which implies that agent A_1 participates in two different congested situations (hotspots).

The proposed method is a variation of the sparse cooperative Q-learning method introduced in [46] that exploits the structure of coordination graphs. In particular, if two peer agents, A_i and A_j, are connected through an edge in the coordination graph, the joint Q-function for these agents is denoted as $Q_{ij}(s_{ij}, a_{ij})$, and is updated using the following rule:

$$Q_{ij}(s_{ij}, a_{ij}) = (1-\eta)Q_{ij}(s_{ij}, a_{ij}) + \eta\left[\frac{r_i}{|N(A_i)|} + \frac{r_j}{|N(A_j)|} + \gamma Q_{ij}(s'_{ij}, a^*_{ij})\right], \tag{5.7}$$

where η is the learning rate and $|N(A_i)|, |N(A_j)|$ are the number of *peers* (neighbors) of agents A_i and A_j, respectively.

It is important to highlight that in the previous equation, a^*_{ij} represents the optimal joint action for both agents A_i and A_j given the joint state s'_{ij}. The joint action refers to the set of actions a_i and a_j that form a common strategy for agents A_i and A_j. In the literature, this optimal strategy is typically estimated using the *max-plus* message-passing algorithm [46]. However, in this case, in order to reduce computational complexity, a simplified approach has been adopted. Specifically, the best joint action is directly obtained from each agent's (e.g. A_i) Q-function $Q_i(s_i, a_i)$, which is calculated as the summation of local Q_{ij} values within its neighborhood:

$$a^*_i = \arg\max_{a_i} Q_i(s_i, a_i) \tag{5.8}$$

$$Q_i(s_i, a_i) = \frac{1}{2} \sum_{j \in N(A_i)} Q_{ij}(s_{ij}, a_{ij}) \tag{5.9}$$

The global Q-function can then be calculated as follows:

$$Q(s, a) = \frac{1}{2} \sum_{i,j \in \mathcal{E}} Q_{ij}(s_{ij}, a_{ij}) \tag{5.10}$$

According to this approach, the agents update their Q-values by propagating edge-specific temporal differences to their neighboring agents, sharing their local rewards with them.

5.5 Hierarchical multi-agent reinforcement learning for ATM

State abstraction (or state aggregation) has been widely studied in artificial intelligence (e.g., [110]) and operations research [111] as a technique for accelerating decision making. Abstraction can be thought of as a process that maps the ground representation, the original description of a problem, to an abstract representation, a much more compact and easier one to work with [110]. It denotes a relation of the form:

$$\phi : \mathcal{S} \to \mathcal{S}_\phi$$

that maps each environmental state $s \in \mathcal{S}$ to an abstract state $\phi(s) \in \mathcal{S}_\phi$, where typically $|\mathcal{S}| \leq |\mathcal{S}_\phi|$.

5.5.1 State abstraction for reinforcement learning

State abstraction is a commonly used technique in RL that simplifies the representation of the *ground* state space. In many real-world problems, the *ground* state space can be large and complex, which can make learning and decision-making computationally expensive and time-consuming. Rather than operating within the *ground* state space, decision makers often achieve faster solutions by operating within an *abstract* state space, grouping similar states together. Thus, the dimensionality of the original state space is reduced and the learning process becomes more efficient.

The goal of state abstraction is to create a coarser, more general representation of the state space, where states that share similar characteristics are clustered together

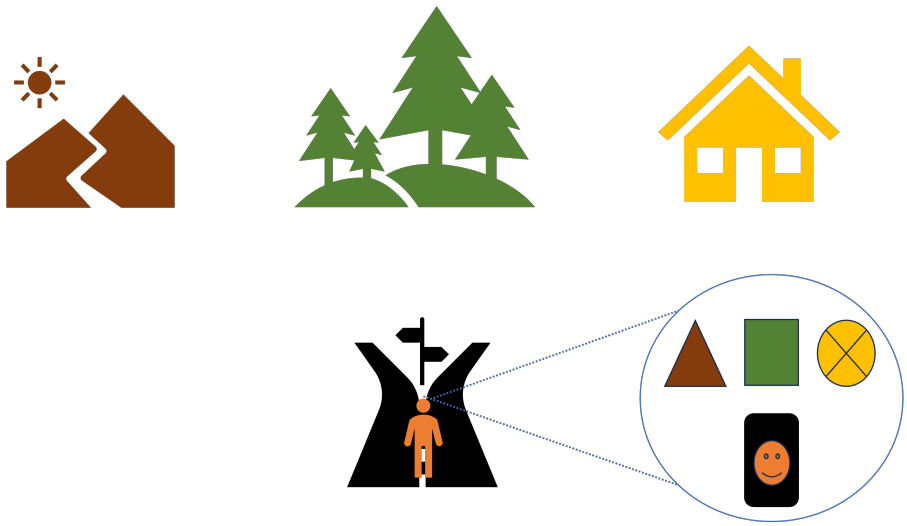


Figure 5.3: Abstract representation of the original state space.

(see Fig. 5.3). This clustering is typically performed based on some predefined criteria, such as proximity or similarity of state features. By grouping states together, the number of distinct states is reduced, resulting in a smaller, more manageable state space.

State abstraction can be applied in different ways depending on the specific problem and domain. One common approach is to partition the *ground* state space into a set of macro-states, where each macro-state represents a cluster of similar states. Each macro-state then serves as a representative of the group it belongs to, encapsulating the relevant information of the original states within that group.

Once the state space has been abstracted, the RL agent operates on the *abstract* state space rather than the original high-dimensional state space. Moreover, state abstraction can help in generalization, as the agent learns to make decisions based on the abstracted states, which may capture the important underlying patterns and structure of the environment.

After learning a policy in the *abstract* state space, the next step is to transfer the solution back to the *ground* state space. To achieve this, the agent needs a mapping function that relates the abstract states to their corresponding ground states. This mapping function establishes a connection between the abstract and ground state spaces. Ultimately, this allows the agent to make decisions and take actions in the *ground* state space based on its learned policy in the *abstract* space, and refine it to obtain a better solutions.

5.5.2 The proposed scheme

The DCB problem in ATM can be formulated as a hierarchical MARL framework, where the flight-agents operate in the same environment and share common resources at multiple levels of abstraction. According to the problem specification, the proposed scheme extends the typical hierarchical MARL (presented in Chapter 2.2.4) containing the following features:

- A set of *abstraction levels* $L \in \{1, \ldots, h\}$.
- A *ground / abstract local state*, s_i^L, per agent A_i, comprising state variables that correspond to (a) the delay imposed to the trajectory T_i denoted as d_i^L, and (b) the number of hotspots in which A_i is involved denoted as h_i. The *ground / abstract joint state* s_{ij}^L of agents A_i and A_j is the tuple of the ground /abstract state variables for both agents.
- A set of *actions* $\mathcal{A}_i^L$ per agent A_i that ranges in the set of delay options assumed by A_i in D_i, given the abstraction step at level L (defined below).
- A *state abstraction function* at every level L:

$$\phi_L : s_i \rightarrow s_i^L \tag{5.11}$$

  that maps ground local states of agent A_i (or ground joint states of agents A_i and A_j) to abstract local states at level L, s_i^L (respectively, s_{ij}^L). Specifically, the abstraction function ϕ_L maps ground states with respect to the abstraction step (specified below) applied on delays, to the corresponding abstract states, given that ground and abstract states have equal number of hotspots.
- The *abstraction step* at level L, denoted as M_L, defining the amount of time instants that correspond to the same abstract time point, used in the state abstraction function. For example, when $M_L = 10$, then time instants $1-10$ belong to the first abstract time point at level L, $11-20$ to the second, and so on. This is further discussed below. The value of M_L decreases as the abstraction proceeds from level h and moving towards level 1, where $M_1 = M_{ground} = 1$.
- The *local reward* of an agent A_i, denoted r_i, is the reward that the agent gets by executing its local action in a local state at the ground level. In this study, the reward function is independent of the hierarchy level L.

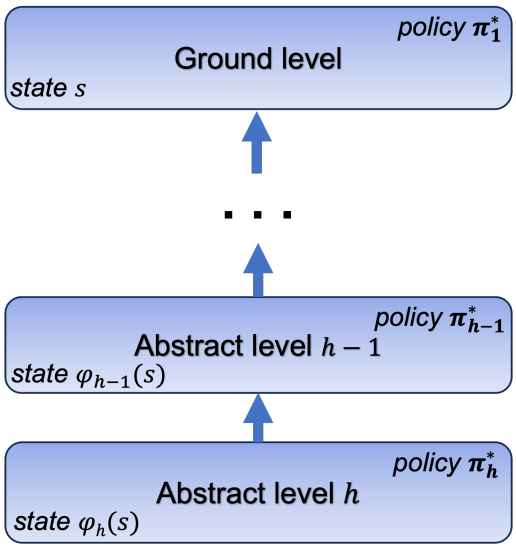


Figure 5.4: Multi-level abstraction

It must be noted that the proposed hierarchical framework is general and can support many levels of abstraction. In Fig. 5.4, there are $h-1$ abstract levels and a ground level. The higher the level, the more abstract the state representation becomes. The solution of a higher abstract level is transferred to the next lower abstract level by propagating the Q-values. In this context, the agent carries over the knowledge and policy learned at a higher level to initialize the policy of the lower level. The process continues until the Q-values reach the ground level, which contains the most detailed and specific states of the environment. At the ground level, the final solution is achieved by refining the policy obtained from the abstract levels.

However, the proposed method which constitutes a hierarchical extension of CMARL (called *HCMARL*), considers two (2) levels of state abstraction: the *ground level* and the *abstract level*. Specifically, the focus is on the *delay* feature of the state, which is the only variable subject to abstraction.

In HCMARL, the state space of the ground level is initially mapped to an abstract state space by dividing the delay interval (of length $MaxDelay$) into a number of K equidistant intervals of length L, where $K = MaxDelay/L$. As shown in Fig. 5.5, the same state in the abstract space corresponds to all states of the ground level with delays between consecutive time points t and $t+I$, where $I = 5$ in this example. At the abstract level, the agent has the option to increase its delay by a unit of size equal to I time instants, until reaching the next time point $t+I$, as long as it does not exceed the $MaxDelay$ value. In contrast, at the original state level, only one time

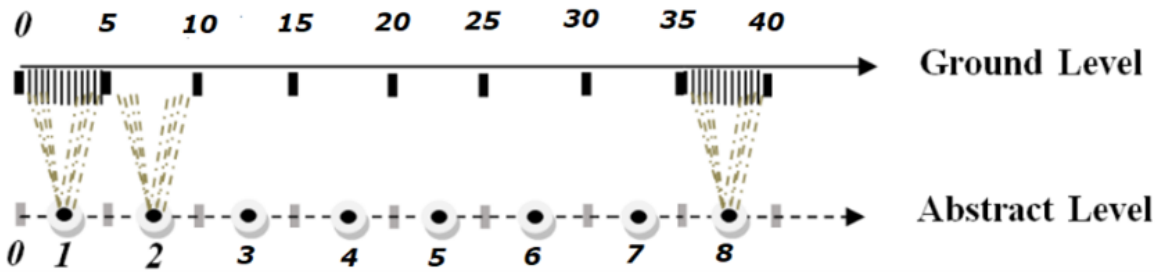


Figure 5.5: An example of the construction of the abstract level. Delay ($MaxDelay = 40$ in this case) is partitioned into a number of $K = 8$ equidistant intervals of 5 minutes and delays between consecutive time points are mapped to the same state in the abstract level.

instant can be added as in CMARL.

The learning process begins by training a policy at the *abstract level* in order to provide an initial solution to the DCB problem. Afterwards, this abstract policy can be transferred to the *ground level* in order to further refine the solution. During this transfer, we assume that the original states corresponding to the same abstract level state have the same optimal Q^* values (policy) from the Q-values that have already been computed at the abstract level.

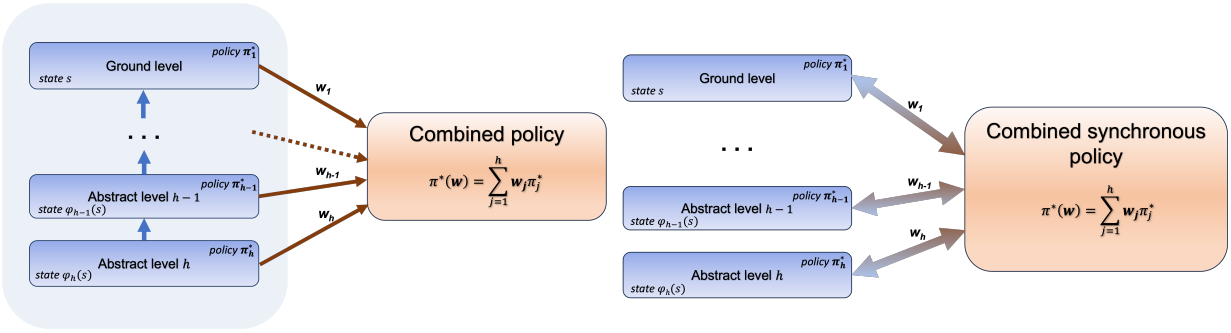
The proposed hierarchical scheme offers two advantages. Firstly, it provides an efficient initialization strategy for Q-learning in the original state space. Secondly, it yields a final solution that is superior to the abstract level's solution for the DCB problem. In the experiments conducted, the interval length was set to $I = 10$ minutes for the abstract level. However, as already pointed out, the proposed scheme can handle multiple abstract levels of variable interval lengths.

The local Q-function Q_{ij}^L at the abstract level L for the agents A_i and A_j that are connected through an edge in the coordination graph, is calculated according to the joint state at abstraction level L, as determined by the mapping function ϕ_L and the joint action a_{ij}. According to the update rule in Eq. 5.7, the hierarchical case becomes:

$$Q_{ij}^L(\phi_L(s_{ij}), a_{ij}) = (1-\eta)Q_{ij}^L(\phi_L(s_{ij}), a_{ij}) + \eta\left[\frac{r_i}{|N(A_i)|} + \frac{r_j}{|N(A_j)|} + \gamma Q_{ij}^L(\phi_L(s'_{ij}), a^*_{ij})\right], \tag{5.12}$$

where $\phi_L(s_{ij})$ is the state abstraction function that maps every joint ground state of agents A_i and A_j to an abstract local state at level L.

Again, just as in the non-hierarchical CMARL method, a^*_{ij} denotes the best joint



(a) Asynchronous multi-level combined policy

(b) Synchronous multi-level combined policy

Figure 5.6: Two alternative hierarchical reinforcement learning schemes that combine multi-level policies obtained from different abstract levels.

action of both agents A_i and A_j for the joint state s'_{ij}. Instead of using the *max-plus* algorithm, the best actions can be directly obtained from each peer agent's Q-function:

$$a_i^* = \arg\max_{a_i} Q_i(\phi_L(s_i), a_i) \tag{5.13}$$

$$a_j^* = \arg\max_{a_j} Q_j(\phi_L(s_j), a_j) \tag{5.14}$$

5.5.3 Extensions

Some early studies have been conducted using two alternative schemes based on the hierarchical multi-agent reinforcement learning framework discussed previously.

Assuming the linear model, the first scheme (Fig. 5.6(a)) works in two (2) phases. The first phase is the same as the one explored previously and involves transferring the learned policy from a higher abstract level to the next lower abstract level, until the ground level. After training the policy in the ground level, the second phase initiates, where the scheme aims to create a combined policy that integrates the policies from all levels. This is achieved by assigning weights to each level's policy, determining their influence on the final combined policy. The goal is to find the optimal weight vector that balances the contribution of each level's policy. By leveraging the strengths of each level's policy, the combined policy aims to achieve better performance in the environment.

The second scheme (Fig. 5.6(b)) consists, again, of multiple abstract levels and a ground level. Contrary to the first scheme, each level learns a policy independently of the other levels, without helping in the initialization of the policy of the next level. Instead, the policies at each level communicate with a combined synchronous policy and exchange Q-values. The synchronous combined policy integrates the policies of all levels through a weighted combination scheme. As the training progresses, the level-specific policies continue to learn and refine their policies independently, but also with the help of the combined synchronous policy. The goal is to find an optimal weight vector for the combined policy, which provides the final solution.

In both schemes, the weight optimization process can be carried out with various methods, such as gradient-based optimization or reinforcement learning methods, such as DQN including regularization techniques for linear weights (e.g. l_2 or l_1 penalty).

5.6 Experimental results

5.6.1 Data description

The construction of the evaluation cases can be achieved by exploiting the *flight plans*, which give an image of the ATM before the flights depart from the origin airport. In this experimental study, the flight plans are obtained by the Spanish Operational Data Center, and are created based on the guidance of domain experts, so as to ensure that the solutions provided by the proposed methods can be compared to the delays imposed by the NM. Along with the flight plans, an evaluation case is accompanied with a list of all active sectors and their capacities for the specific day.

In this context, the effectiveness of the proposed methods is assessed in multiple real-world evaluation scenarios with varying levels of difficulty. Specifically, each evaluation case represents a particular day over Spain in 2016. The difficulty of each case can be determined based on the following criteria, obtained by the real solutions provided by the network managers (NM):

- **Number of flights**: The total number of flights for that particular day above Spain.
- **Average traffic density**: The number of interacting flights in average.

- **Maximum delay**: The maximum delay imposed that particular day to any flight.
- **Average delay**: The average delay per flight ignoring all delays with less than 4 minutes (according to experts' advise).
- **Number of flights with delay**: The number of flights that delays were imposed.
- **Maximum number of hotspots (number of flights)**: The initial number of hotspots together with the number of flights that participate in those hotspots (each flight may participate in more than one hotspots).

Table 5.1: Description of the evaluation cases

Scenario	**Evaluation cases**			**NM reported results**		
	# flights	***avg traffic density***	***max delay***	***avg delay***	***regulated flights***	***# hotspots (# flights)***
Jul2	5572	6.39	80	1.663	498	29 (778)
Jul12	5408	5.84	95	0.95	254	28 (820)
Aug4	5544	6.41	66	0.383	146	33 (853)
Aug13	6000	10.89	147	1.152	415	53 (1460)
Sep3	5788	5.24	61	0.732	280	26 (783)

Table 5.1 contains the criteria values for each evaluation case. In general, it is important to highlight that while the NM impose delays on flights to address demand-capacity imbalances, this alone does not resolve the DCB problem. Even when NM impose delays during the pre-tactical stage, hotspots can still occur. This highlights the tolerance of the system and its reliance on resolving imbalances during the tactical phase of operations, as opposed to the pre-tactical phase, aligning with the objective of this study. As a result, the delays imposed by the NM cannot be directly compared to the solutions provided by the proposed methods. The low predictability in the pre-tactical phase currently limits the NM from effectively resolving the DCB problem, leaving decisions to be made during the tactical phase. However, the comparison does demonstrate the potential of HRL in effectively addressing such problems.

5.6.2 Implementation details

During the training of CMARL, 15.000 training episodes are considered following an ϵ-greedy exploration-exploitation strategy. In particular, the probability is set to $\epsilon = 0.9$ and every 120 rounds it is diminished by the value of 0.01. To enhance the performance of the proposed methodology, flights that do not participate in any hotspot automatically are given delay equal to zero (0) as a deterministic decision rule. However, it must be noted that any of these flights may participate in hotspots in the future, due to the dynamic delay scheduling that occurs in the multi-agent environment.

On the other hand, the HCMARL method contains two stages (corresponding to the abstract and the ground level) of learning, both consisting of 15.000 episodes following an ϵ-greedy exploration-exploitation strategy. It must be noted that the exploration of the ground level does not need to be as extensive as in the case of the abstract level, hence it begins with probability $\epsilon = 0.7$. Finally, the learning rate α is set to 0.01, the value of discount factor γ is set to 0.99, and the reward parameter λ (Eq. 5.2) is experimentally set to 20.

5.6.3 Results

The proposed methods (CMARL and HCMARL) are compared to three (3) HMIRL approaches. Specifically, these approaches considered state–temporal abstraction (stHMIRL), state only abstraction (sHMIRL) and temporal only abstraction (tHMIRL). Specific details about the implementation of these algorithms are given in [112].

Tables 5.2 and 5.3 present detailed comparative results between the HCMARL, the non-hierarchical CMARL, and the three (3) HMIRL approaches, where several statistical measurements were considered that have been calculated after executing ten (10) independent experiments for every evaluation case. In particular, the mean value, the standard deviation (std), the median and the interquartile range (IQR) are provided for the *average delay per flight* and the *number of regulated flights* (i.e. flights with delay), respectively. The best mean value in each Table and case is indicated in bold, while the second best is underlined.

According to the obtained results, it is obvious that *HCMARL* is among the most dominant methods (along with *sHMIRL*) and constantly provides exemplary results. Only in the case of Aug13 the *tHMIRL* presents the best average delay per flight, but

Table 5.2: Statistical measurements of the average delay per flight, as calculated by 10 independent experiments. Best mean value is indicated in bold, and the second best is underlined.

Scenario	Method	Mean	std	median	IRQ
Jul2	HCMARL	<u>1.590</u>	0.049	1.590	0.080
	CMARL	1.728	0.045	1.725	0.075
	stHMIRL	1.742	0.046	1.735	0.061
	sHMIRL	**1.358**	0.054	1.350	0.005
	tHMIRL	1.637	0.060	1.650	0.068
Jul12	HCMARL	**0.103**	0.009	0.100	0.010
	CMARL	<u>0.115</u>	0.013	0.110	0.020
	stHMIRL	0.410	0.028	0.408	0.033
	sHMIRL	0.187	0.015	0.185	0.028
	tHMIRL	0.205	0.016	0.205	0.020
Aug4	HCMARL	<u>0.783</u>	0.056	0.780	0.090
	CMARL	0.835	0.059	0.820	0.080
	stHMIRL	1.129	0.056	1.130	0.055
	sHMIRL	**0.731**	0.028	0.730	0.035
	tHMIRL	0.846	0.042	0.855	0.065
Aug13	HCMARL	1.115	0.053	1.110	0.035
	CMARL	1.383	0.048	1.380	0.050
	stHMIRL	1.168	0.041	1.159	0.053
	sHMIRL	<u>0.996</u>	0.044	0.990	0.075
	tHMIRL	**0.975**	0.042	0.975	0.048
Sep3	HCMARL	<u>0.790</u>	0.041	0.780	0.065
	CMARL	0.861	0.052	0.845	0.050
	stHMIRL	0.855	0.064	0.867	0.104
	sHMIRL	**0.578**	0.038	0.570	0.038
	tHMIRL	0.896	0.045	0.895	0.028

Table 5.3: Statistical measurements of the regulated-flights, as calculated by 10 independent experiments. Best mean value is indicated in bold, and the second best is underlined.

Scenario	Method	Mean	std	median	IRQ
Jul2	HCMARL	<u>337.00</u>	8.56	334.00	12.50
	CMARL	441.20	5.93	441.00	8.00
	stHMIRL	448.70	13.75	448.50	14.25
	sHMIRL	**331.90**	14.74	329.50	16.25
	tHMIRL	361.20	10.40	362.00	15.00
Jul12	HCMARL	**61.20**	1.61	61.00	2.00
	CMARL	<u>66.00</u>	4.27	66.00	4.00
	stHMIRL	228.65	17.66	225.50	19.75
	sHMIRL	127.90	9.45	128.50	13.25
	tHMIRL	135.00	9.37	131.00	13.75
Aug4	HCMARL	**216.40**	8.30	214.00	8.50
	CMARL	243.40	7.63	234.50	7.00
	stHMIRL	385.75	20.19	391.00	30.75
	sHMIRL	<u>243.00</u>	12.11	242.00	21.75
	tHMIRL	309.00	9.07	309.00	8.00
Aug13	HCMARL	**402.81**	3.71	402.00	5.00
	CMARL	456.20	11.91	452.50	9.00
	stHMIRL	566.50	12.97	568.00	19.50
	sHMIRL	<u>421.70</u>	17.34	419.00	14.75
	tHMIRL	443.30	18.87	449.50	31.50
Sep3	HCMARL	<u>216.20</u>	5.10	215.00	9.00
	CMARL	263.50	11.36	264.00	18.00
	stHMIRL	359.45	17.31	360.00	25.25
	sHMIRL	**207.90**	9.02	206.50	13.00
	tHMIRL	296.90	11.11	298.00	13.25

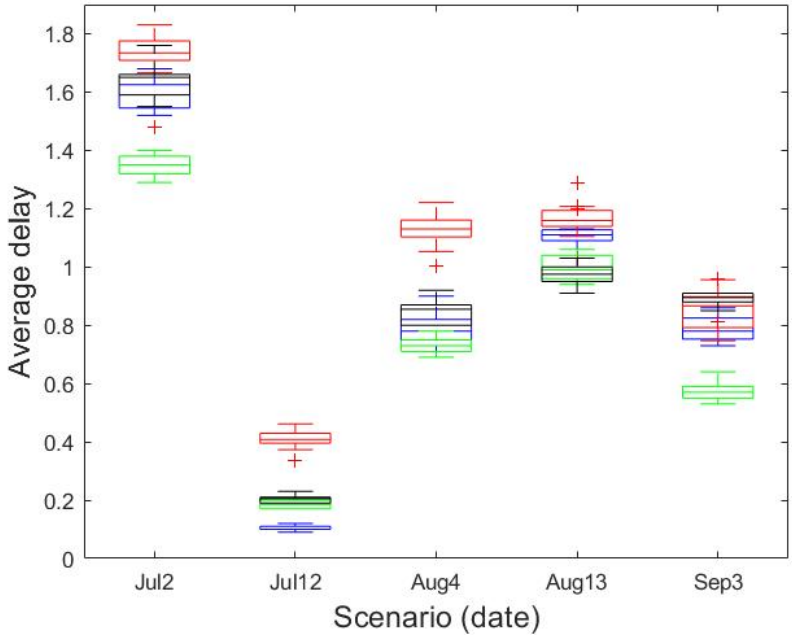
for the same case *HCMARL* has the lowest number of regulated flights. Comparing the proposed methods with the results provided by the NM (see Table 5.1), it can be noticed that the methods consistently provide better results, except in the case of $Aug4$, where all methods are unable to outperform the NM. However, as already pointed out, delays imposed by the NM leave unresolved many imbalances, while the solutions of the proposed methods always eliminate the hotspots. As an example, the delays of NM resolve only 2 hotspot occurrences out of the 33 in Aug4 scenario (see Table 5.1). The reported results of the proposed methods in conjunction with the delays imposed by the NM, show the effectiveness of the hierarchical framework and its ability to provide qualitative solutions to real-world complex problems.

The advantages of the *HCMARL* can be summarized as follows:

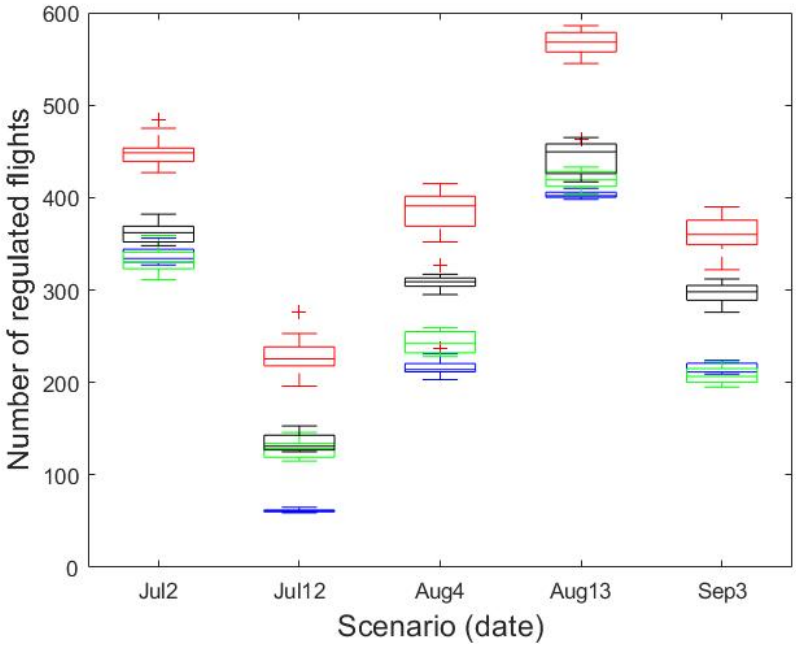
- It effectively explores the state space at different levels of abstraction. This is due to a more informative initialization of the original (ground) space by the abstract layer, that allows to enhance the learning procedure and discover more optimal solutions.
- It enables the exploitation and combination of multiple policies through transfer learning mechanisms

Furthermore, comparing the results of HCMARL to those reported by the non-hierarchical CMARL method for the same evaluation cases, it is clear that the hierarchical method has improved performance and effectiveness. However, in the case of Jul12, CMARL outperforms the HMIRL methods and achieves the second highest score behind *HCMARL*.

In addition, Fig. 5.7 illustrates the same results using box plots representations in the five evaluation cases. The first diagram (Fig. 5.7(a)) shows the *average delay per flight*, while the second one (5.7(b)) presents the *number of regulated flights*. Notice that, following the standard practice in the domain, flights with delays less than 4 minutes are not considered in the results. In all cases the blue colored box plot corresponds to the *HCMARL* method, the red colored box plot corresponds to the *stHMIRL* method, the green one to the *sHMIRL* method, and the black one to the *tHMIRL* method. On the other hand, the non-hierarchical method CMARL is not included in the box plots, so as to have a more clear comparison between the hierarchical methods.



(a) Average delay per flight, and per scenario



(b) Number of regulated flights per scenario

Figure 5.7: Comparative results in terms of (a) the average delay per flight and (b) the number of regulated flights presented in box plots for the four comparative hierarchical methods per evaluation case: HCMARL (blue), HMIRL (red), sHMIRL (green), and tHMIRL (black)

The box plot diagrams provide insights into the robustness of the *HCMARL* method in both measurements (average delay and regulated flights), where this method (along with *sHMIRL*) presents the best results between the hierarchical methods. The other two methods either do not offer effective results, or contain outliers in their solutions. More specifically, *stHMIRL* presents outliers in almost all cases, while *tHMIRL* fails to provide better results compared to *HCMARL*. In contrast, the proposed *HCMARL* method does not exhibit such outliers. The standard deviation and interquartile range of this method is consistently lower across most evaluation cases, as depicted in Tables 5.2, and 5.3.

5.7 Summary

This chapter presents a hierarchical MARL framework to address the challenge of resolving DCB problems during the pre-tactical phase in the ATM domain, which involves multiple agents and complex congestion scenarios. The proposed approach supports state abstraction at various levels and offers benefits in several aspects, such as: (a) progressive refinement of solutions by initializing state-action values based on previous abstraction levels, (b) cooperation among agents through a coordination graph, and (b) the ability to accommodate different levels of abstractions.

By exploring these alternatives, the proposed hierarchical scheme operates on two (2) levels of abstraction: an abstract level and a ground level. The effectiveness of the proposed methodology has been demonstrated through evaluation on five (5) real-world cases, representing real flights above Spain. The results indicate that the proposed HCMARL method exhibits consistent behavior across different cases, showcasing its robustness and ability to handle complex congestion scenarios with a large number of agents.

Chapter 6

Inverse Reinforcement Learning for Aircraft Trajectory Prediction

The main goal of AI research is to create machines that possess human-like intelligence, and reasoning abilities. Inverse reinforcement learning, involves teaching apprentice agents by observing demonstrations given by experts. This framework allows for the development of improved solutions that could even surpass the performance of the experts themselves in some cases. In this chapter, the focus shifts on addressing the problem of predicting aircraft trajectories in the aviation domain using an IRL approach. The proposed learning scheme involves imitating demonstrated historical expert flight trajectories by utilizing raw trajectory data enriched with meteorological features. The algorithm learns an efficient reward model during the imitation process, which also possesses the ability to generalize to unknown cases.

6.1 Overview

In the aviation industry, towards implementing the trajectory based operations (TBO) paradigm, predictability of trajectories is of immense importance. This is because uncertainties that occur during flights can have significant impacts on various aspects of operations, including those related to airspace users such as airlines, air traffic controllers, ground operators, and passengers. The necessity to confront these uncertainties and adapt to them can be costly for all those involved. As an example, it may require imposing *delays* to flights or choosing alternative routes to those originally planned, which can result in increased fuel consumption, higher workloads, additional costs, and potentially challenge the capacity of the entire ATM system. Therefore, ensuring trajectory predictability is critical for minimizing disruptions and maintaining efficiency in the aviation industry.

Given that RL techniques are inherently a good candidate for dealing with trajectories, the aim here is to develop and evaluate IRL methods that are trained to imitate demonstrated flight trajectories. By doing so, the historical trajectories are treated as training data provided by an "expert" that an IRL algorithm should exploit and learn policies for generating sequences of actions for moving between positions in the 3D space through time, predicting the evolution of trajectories.

When historical demonstrations are available for a specific task, it is often beneficial to observe the expert behavior and learn a policy based on the expert's actions. Typically, this involves access to a set of expert trajectories, which represent a sequence of states and actions in an environment. The objective is to create a policy that imitates the expert's behavior and reproduces the demonstrated trajectories. One challenge in this process is the lack of a reward signal to evaluate the expert's decisions and facilitate the learning process.

IRL makes the assumption that the expert's policy is optimal with respect to an underlying unknown reward function. The goal of the *apprentice* agent is to discover the reward function using the available expert demonstrations, which can explain the optimal behavior [25, 113]. At a second phase, direct RL schemes can be employed for optimizing the control policy based on the discovered reward function and imitate the hidden decision mechanism of the expert. Thus, the training procedure iteratively updates the reward function and learns the policy.

This chapter presents an integrated solution to the problem of aircraft trajectories

prediction through the prism of IRL, and more specifically *apprenticeship learning* [25]. The main idea is to discover a policy that generates trajectories that match those demonstrated by the expert. The learning procedure is not solely focused on creating a policy as a mapping from demonstrated states to demonstrated actions (which are often unknown), but rather as a function that operates in an environment, mapping agent's states to a hypothetical action space that can lead to an optimal imitation of the expert.

This study presents several significant contributions, which can be summarized as follows:

- To begin with, this study utilizes an IRL-based *apprenticeship learning* framework to address the problem of aircraft trajectory prediction for the first time. This is strengthened by the fact that it covers a significant application, since for the experimental analysis real trajectories and real-world data in the aviation domain are used. Another interesting aspect is that apart from the observed trajectories in the spatial domain, there is another series of raw data concerning meteorological information that is also time-dependent and must be taken into account to the design of the intelligent agent. As a result, the environment becomes more complex and has a stochastic nature that may affect significantly the imitation procedure.

- Secondly, a comprehensive trajectory analysis is established by assuming an informative agent state representation considering a feature vector space of spatial and meteorological information. Also, the action space considers a discrete set of aircraft heading angles found by the historical data.

- Finally, the proposed IRL framework allows the flexibility to construct (sub)optimal policies with generalization capabilities that imitate efficiently the demonstrated trajectories. As experiments have shown, with only a few trajectories as input, the proposed method is able to reach very satisfactory solutions

The rest of this chapter is structured as follows. Section 6.2 presents a brief review of the literature, and Section 6.3 describes the problem setting. In Section 6.4 the general framework of the proposed IRL method is described, and Section 6.5 reports evaluation cases and experimental results. Finally, Section 6.6 concludes the chapter with a small discussion.

6.2 Related work

In traditional RL tasks the reward function is usually specified manually and an optimal policy is found so as to maximize the expected accumulated reward. On the other hand, IRL tries to estimate the unknown reward function that can best explain the expert behavior and induce solutions similar to that of the expert's. For this reason, a set of expert trajectories must be available as a training set which cover the sequences of states that the expert visits while performing actions in its environment [114, 115]. The expert's reward function, R^E, can be depicted in various forms such as a linear combination of weighted feature functions or as a distribution over real-valued maps from states to reward values, among others. The main problem in IRL is the *reward ambiguity*, where multiple reward functions can explain the same expert behavior, as it has been thoroughly discussed in Chapter 2.

One of the core approaches in IRL is called *margin optimization* [115, 116, 117, 118], which has the objective to acquire a reward function that explains the expert's policy better than all the other policies by a margin. The methods under this prism aim to resolve the *reward ambiguity* problem by finding solutions that maximize a certain margin, but they introduce bias to the reward function.

On the other hand, to eliminate that bias, *entropy optimization* [26, 119, 120, 121, 122] methods utilize the *maximum entropy principle* [27], which helps acquire a distribution over behaviors that is parameterized by the reward function weights. Based on this principle, the chosen distribution is the one with the maximum entropy. More details about this approach can be found in Chapter 2.3.2.

An alternative point of view for IRL is through classical ML techniques, such as classification and regression. One such solution is to treat the problem as a supervised classification task and train a classifier to act as the policy [123, 124, 125, 126], or utilize probabilistic models, such as Gaussian Mixture Models (GMM) or Hidden Markov Models (HMM), to generate similar trajectories to those demonstrated by the expert [127, 128]. Moreover, IRL can be defined as a multi-class classification problem by framing the state-action pairs of an expert trajectory as input-label pairs [129, 130].

In this study, a modified version of the *apprenticeship learning* [25] approach is proposed. This method lies in the *margin optimization* category. In short, the reward function is featured-based and the aim is to learn the feature weights by minimizing the margin between the feature expectations of the learned policy and the feature

expectations from the expert's demonstrations. More details of this methods are given in section 6.4.2.

IRL has also enjoyed diverse applications in automated control systems that try to imitate the behavior of human experts. Some characteristic examples are: learning how to drive [25], predicting trajectories of off-road vehicles [131], controlling helicopters [132], capture human navigation behaviors [133], estimating optimal neural network architecture [134], predicting mouse movements [135], generating music [136] and handling demographic data from city visitors trajectories [137].

The literature of IRL becomes rich and grows in our days. An alternative imitation learning strategy from expert trajectories is given in [138] without calculating precisely the reward signal. Finally, an interesting approach is presented in [139] that uses successful as well as failed demonstrations simultaneously, for estimating the reward signal and learning optimal policies.

In most cases, it is assumed that the expert trajectories consist of both action and state values from the expert. However, in real-world applications optimal actions are usually not available. This requires the structural exploration of the environment in order to imitate the expert behavior according to environment's dynamics (although it is not necessary to learn such a model for the environment).

6.3 Problem setting

In the aviation domain, a trajectory refers to the description of an aircraft's movement both in the air and on the ground. This information can be represented through a sequentially ordered series of aircraft states, which are described using a set of variables. The most significant variables are the longitude (lon), latitude (lat), and geodetic altitude (alt) coordinates. Trajectories that solely offer spatio-temporal details for each state (i.e., 3D positions and timestamps) can be identified by leveraging surveillance data and are referred to as *raw trajectories*.

More formally, a *raw trajectory* T of an aircraft is defined to be a sequence of $|T|$ pairs s_i=$<p_i, t_i>$, $i \in [1, \ldots, |T|]$, where p_i is a point (lon, lat, alt) in the 3D space and t_i is a timestamp. An *enriched trajectory state* corresponds to a raw trajectory point which is defined to be a triplet s_i=$<p_i, t_i, v_i>$, where v_i is a vector consisting of categorical and/or numerical variables annotating the raw trajectory state. Hence, an *enriched*

trajectory T is defined to be a sequence of enriched states s_i=<p_i, t_i, v_i>, $i \in [1, \ldots, |T|]$.

By adopting a data-driven approach, the objective is to utilize historical aircraft trajectories that comprise 3D aircraft positions, timestamps, and additional meteorological variables. Within this context, a *predicted trajectory* can be characterized as the anticipated progression of the aircraft's state based on its present flight conditions. This entails using an initial state and specifying how the aircraft will transition from one state to another in a sequential manner.

Casting the trajectory prediction to a data-driven problem, and assuming a set $T_E = \{T_{E,i} | i = 1, 2, 3, ...\}$ of historical, demonstrated enriched trajectories, the *trajectory prediction problem* can be defined as follows: Given T_E and a "reward" function r, the objective is to learn a policy for predicting any trajectory T_π, such that

$$T_\pi \sim \max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{H} \gamma^t r(\langle p, t, v \rangle, a_t) \right] , \tag{6.1}$$

where $\mathbb{E}[.]$ denotes the expected cumulative discounted rewards - given the discount factor $\gamma \in (0, 1)$ - for all enriched states $s_t = \langle p, t, v \rangle$ generated along any trajectory, up to a time horizon H, by following a policy $\pi(a_t|s_t)$ prescribing the probability of applying an action a_t at a state s_t. Actually, according to equation 6.1, the ultimate objective is to find the policy π that determines the generation of a maximal expected cumulative reward predicted trajectory T_π. The time horizon H may depend on the average flight duration, extracted from historical data in T_E.

The form of the reward function can vary depending on the approach taken to address the problem. In the context of a data-driven trajectory prediction process, the reward function typically measures the extent to which predictions align with patterns, constraints, policies, and demonstrated trajectories extracted from historical data. This topic will be explored in more details in the following sections.

6.4 Flight trajectory modeling with IRL

In this study, a set of expert trajectories that correspond to different flights from the same origin-destination airport pair is considered. These trajectories are assumed to be executed within a defined time period (e.g. a month). Although it is possible to extend the method to cover longer time periods, doing so would necessitate more extensive exploration within a significantly larger state-space due to larger weather

fluctuations between months. Additionally, it is essential for the trajectories under consideration to be uni-modal, meaning they should adhere to a single behavioral pattern. Methods addressing challenges with multi-modal expert behaviors will be introduced in the subsequent Chapter.

Thus, the goal is to imitate the demonstrated trajectories in T_E and acquire a (sub)optimal policy that can generate accurate predictions for new trajectories executed under similar weather conditions. Furthermore, it is essential for the learning method to possess generalization capabilities, enabling it to learn policies that predict trajectories, even when starting from previously unseen initial positions.

6.4.1 The state and action spaces

The enriched trajectory of an aircraft considered in this study is a time-series that contains two types of input variables. The first type consists of four (4) variables that provide the spatio-temporal features of longitude, latitude, altitude, and timestamp. The second type of information describes the weather conditions at that specific spatio-temporal state. These weather conditions are characterized by various meteorological features, which are extracted from the National Oceanic and Atmospheric Administration (NOAA) database [1]. Specifically, the meteorological features include the pressure surface, relative humidity isobaric, temperature isobaric, wind speed gust surface, u-component of wind isobaric, and v-component of wind isobaric.

Rather than working with the original state space, a more informative feature space is constructed in order to improve the discrimination capabilities among states. In the case of spatial information, a set of L_{pos} equidistant points in time along the centroid trajectory of the training set is considered: $\{\mu_1^{pos}, \ldots, \mu_{L_{pos}}^{pos}\}$, where L_{pos} is set manually. Using these points, a feature space can be constructed according to L_{pos} RBF basis functions:

$$\phi_{pos}(s) = (\phi_1(s), \ldots, \phi_{L_{pos}}(s)) \text{ , where } \phi_k(s) = e^{-\beta_{pos}\|p-\mu_k^{pos}\|^2} \tag{6.2}$$

In the above equation β_{pos} is a positive scalar parameter (inverse variance) that is common to any basis function and p is the 2D spatial state (lon, lat) of the aircraft. It is important to highlight that in the construction of spatial features, only the longitude and latitude variables are taken into account. The geodetic altitude is obtained using

[1] https://www.ncei.noaa.gov/cdo-web/

a straightforward linear regression NN that has been trained beforehand. Although this may appear as a simplification, it proves to be highly beneficial in generating accurate features and serves as a valuable tool for determining the action space.

In the case of meteorological features, a clustering procedure is initially performed by using the *K-means* algorithm over the dataset of demonstrated trajectories considering the six (6) meteorological variables. This procedure returns a number of L_{met} clusters, with L_{met} cluster's centers μ_k^{met}. Then, similar to the case of spatial information, these are used for the construction of a feature space for the meteorological variables based on L_{meteo} RBF basis functions:

$$\phi_{meteo}(s) = (\varphi_1(s), \ldots, \varphi_{L_{meteo}}(s)) \text{ , where } \varphi_k(s) = e^{-\beta_{meteo}\|v-\mu_k^{meteo}\|^2}, \tag{6.3}$$

In the previous equation, β_{meteo} is a scalar parameter (inverse variance) and v is the meteorological state of the aircraft.

Combining the previous two (2) feature vectors, the proposed state space can be described as a vector space of $L_{pos}+L_{meteo}$ dimensions with two sources of information corresponding to spatio-temporal and meteorological variables:

$$\begin{aligned} s &\longrightarrow \phi(s) \\ (p, v) &\longrightarrow (\phi_{pos}(s), \phi_{meteo}(s)) \end{aligned}$$

In the experimental study, 20 features per information type were used, i.e. $L_{pos} = L_{meteo} = 20$. Thus the initial state space has been transformed to a space in 40 dimensions.

In order to control the flight trajectory and provide predictions for the next state, the RL agent faces the challenge that optimal actions are usually not available in real-world applications. As a result, it must explore the environment and learn the dynamics in order to successfully imitate the expert behavior and execute optimal actions.

One possible strategy to overcome this challenge is to use a regression model that directly predicts the next state without considering possible actions. This approach can be thought of as a form of behavior cloning. However, in this study, the problem is formulated within a "classical" MDP framework by constructing an action space related to the problem. This allows the creation of a RL agent to control the flight trajectory and make predictions for the next state.

Specifically, since it is not possible to control the weather conditions, the actions can only be applied on the transition of the aircraft. One possible way to do this

- which is also the case in this study - is through a set of heading angles that the aircraft can be directed towards in the 2D space, defined by the longitude and latitude coordinates. Specifically, the agent can select an action from the set of L_a discrete angles ($\theta = \{\theta_1, \ldots, \theta_{L_a}\}$), causing the aircraft to move to a new position in the 2D space according to the following transition functions:

$$lon_{t+1} = lon_t + cos(\theta_k) \cdot \Delta lon \quad \text{and} \quad lat_{t+1} = lat_t + sin(\theta_k) \cdot \Delta lat, \tag{6.4}$$

where the Δlon and Δlat are the step sizes of longitude (lon) and latitude (lat), respectively, at each position. These have been calculated as the mean value of the difference of these two measurements during successive positions in all training trajectories. It should be noted that the desired magnitude of the aircrafts' velocity is constant and not affected by the RL agent.

Four (4) possible actions were used ($L_a = 4$) that correspond to four (4) heading angles: $A = \{100, 200, 230, 260\}$. It must be noted that the determination of the proper angles can be (automatically) derived from the directions of the expert trajectories.

At the end of this process, the geodetic altitude (alt) is indirectly provided by a linear regression NN given the new (lon, lat) coordinates. As a result, the aircraft moves in a new position in the 3D space, where new meteorological conditions are met.

6.4.2 Apprenticeship learning for trajectory modeling

As it has been thoroughly described in Chapter 2.3.2, IRL aims at discovering a reward function through expert demonstrations that explains the expert policy π_E. A common strategy is to assume a linear model for the reward function over the feature vector of state, $\phi(s)$:

$$R_{\mathbf{w}}(s) = \mathbf{w}^T \phi(s) \tag{6.5}$$

where $\mathbf{w}$ is a vector of linear coefficients of size equal to the dimensions of the feature space, i.e. $L_{pos} + L_{meteo}$. The apprenticeship learning process is focused on determining the proper weight vector in order to shape the expert's behavior.

In the case of *apprenticeship learning*, the linear weights $\mathbf{w}$ are obtained by minimizing the distance between the vectors of policy's feature expectations, $\mu(\pi)$, and expert's feature expectations, $\mu(\pi_E)$. Feature expectations refer to the expected values of features extracted from states under a given policy. The IRL process followed

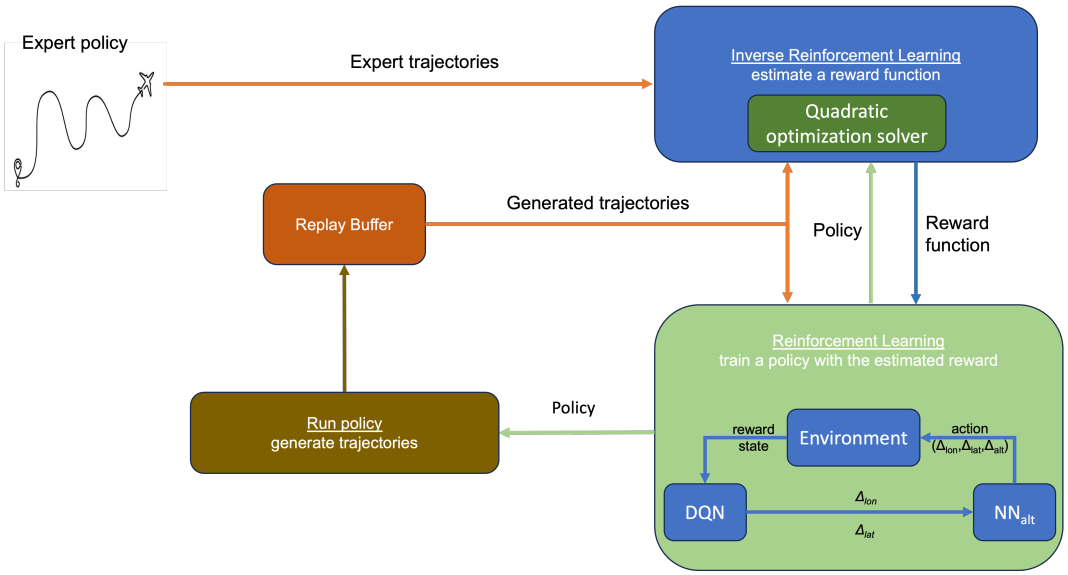


Figure 6.1: Overview of the proposed IRL structure.

in this study, involves applying an optimization strategy that is based on solving a quadratic programming marginal - maximization problem by adding an extra constraint on these weights, i.e. $\|\mathbf{w}\|_2 \leq 1$, that introduces regularization properties on the solution.

Once the linear weights, $\mathbf{w}$, are estimated, then an approximation of the expert's reward function can be calculated. Subsequently, a DQN framework can be employed to learn the apprentice policy by taking into account the estimated reward function. This allows the agent to train effectively and facilitates its ability to imitate the demonstrated trajectories.

An overall illustration of the proposed *apprenticeship learning* method is depicted in Fig. 6.1. The major building blocks are the following:

- An IRL method, which involves estimating the reward function by leveraging both the demonstrated expert trajectories and sample trajectories generated by the agent's policy. To obtain the linear weights required for the calculation of the reward function, it solves a quadratic programming optimization problem for Eq. 6.5.

- A RL method, which employs a DQN framework to address the MDP using the estimated reward function. It involves a feed-forward NN that computes the Q-function in the RL context, which is responsible for making decisions regarding the aircraft's actions. The NN receives the enriched trajectory state as

input, which comprises of the transformed spatial and meteorological features. As explained, the feature space is of dimension $L_{pos} + L_{meteo}$.

- A regression NN, denoted as $NN_{altitude}$, specifically designed to predict the geodetic altitude (alt) of the aircraft at any given point. This prediction is based on the longitude and latitude coordinates (lon, lat), which are provided as inputs to the network.

The proposed methodology begins by creating the feature expectation vector of the experts ($\mu(\pi_E)$), calculated as:

$$\mu(\pi_E) = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^i, a_t^i) \tag{6.6}$$

Subsequently, a random initialization of the weights **w** is conducted, and the first policy π^0 is generated. Next, the feature expectation vector of this policy $\mu(\pi^0)$ is computed according to:

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t, a_t)\right] \tag{6.7}$$

The initial set of weights is then determined through an IRL step, wherein the feature-weights are computed by employing the generated feature expectation and the expert feature expectation vectors as inputs to the quadratic solver. This particular step is considered the core of the IRL algorithm, as it aims to infer the underlying reward function that the expert trajectory seeks to optimize.

Having obtained an estimate of the reward function, the method proceeds by using the DQN framework for solving the RL trajectory prediction problem. During the first step of the RL procedure, a replay buffer is created to store the agent's experiences, and the weights of the NN are initialized randomly. In each iteration, the agent selects an action based on an exploration policy (such as ϵ-greedy) and observes the next state and reward from the environment. The experience tuple $(state, action, reward, next\ \ state)$ is stored in the replay memory, and when a sufficient number of experiences has been collected, mini-batches are sampled in order to update the DQN by minimizing the mean squared error between the predicted Q-values and the target Q-values. The target Q-values are calculated using a separate target network, which is a copy of the main DQN with frozen parameters. This helps stabilize the training process.

After the update of the DQN, the IRL process restarts in order to find a better estimate of the reward function and the whole methodology repeats until it reaches a point where the feature expectations of the expert and the feature expectation of the policy are nearly equal bounded by some constant threshold.

The overall structure of the proposed method can be summarized in Algorithm 7.

Algorithm 7: Apprenticeship learning for trajectory prediction

Input: Expert trajectories T^E, number of number of iterations N

Train the regression network $NN_{altitude}$ for predicting the altitude given (lon, lat) pairs of expert trajectories

Initialize an initial policy $\pi(0)$ and obtain an initial estimation of the reward function, receiving the linear weights **w**

for $i \leftarrow 1$ *to* N **do**

- Pick a random initial state from T^E
- **while** *termination condition is not met* **do**
  - Obtain the feature vector $\phi(s_t)$ consisting of spatial and meteorological features (section 6.4.1)
  - Take an action a_t according to DQN model and move into the next state s_{t+1} (Eq. 6.4)
  - Make a prediction of the altitude alt_{t+1} using the $NN_{altitude}$ model
  - Receive a reward r_t based on the model of the IRL task (Eq. 6.5)
  - Store samples $(s_t, a_t, s_{t+1}, r_t, done)$ in the replay buffer
  - Update the DQN model (every 50 steps)
- Update the target network (every 5 episodes)
- Perform the IRL task and receive a new estimation of the linear weights by solving the constrained quadratic programming problem (every 100 episodes)

Once the learning procedure is complete, the proposed methodology can be utilized for trajectory modeling. The only thing missing is the weather conditions, which can be obtained from a weather forecasting module. The trajectory prediction procedure can be outlined as follows:

1. An initial position represented as $p_0 = (lon_0, lat_0, alt_0)$ is drawn from the set of expert initial positions. From there, the spatial feature vector $\phi_{pos}(s_0)$ is obtained

along with the weather conditions at this specific state and the corresponding weather feature vector $\phi_{meteo}(s_0)$ is derived. Combining the spatial and meteorological features, the feature vector $\phi(s_0)$ is constructed, which serves as the state vector for the agent's initial state.

2. The next step is to select an action a_0 based on the learned policy obtained from the trained DQN network. This action selection process yields the next longitude and latitude values. Subsequently, using the $NN_{altitude}$ regression model, the next altitude value is predicted. By combining the updated longitude, latitude, and altitude values, the method performs a policy-based transition to the next position, denoted as p_1, within the 3D space. This transition allows the progression to a subsequent point along the trajectory.

3. The aforementioned two-step procedure is repeated until the episode either successfully concludes, meaning the target airport is reached, or terminates due to failure.

6.5 Experimental results

6.5.1 Data description

The effectiveness of the proposed method is evaluated in real-world applications using actual flights between Barcelona (BCN) and Madrid (MAD). The dataset consists of radar tracks, which represent raw trajectories, for a total of 528 different flights. Additionally, the weather data for those flights are obtained from the NOAA database. The trajectories were recorded during April 2016 and were divided into two separate clusters of 250 and 278 members, respectively. Each cluster was treated as a distinct set of trajectories. Figure 6.2 illustrates a map of the experimental dataset.

The proposed method, named as *AppLearn*, is evaluated on these two datasets (clusters) according to the following four (4) scenarios in terms of the ratio of cases in the training-test sets:

- P_1: Only the centroid trajectory of each dataset was used as training set ($M = 1$), while all other trajectories were treated as test examples
- P_2: 10% training - 90% test

Figure 6.2: A view of the experimental dataset that consists of expert trajectories flown from Barcelona (BCN) to Madrid (MAD). They are partitioned into 2 clusters of trajectories which are treated separately as two different datasets.

- P_3: training and test sets of equal size ($50\% - 50\%$ partition)
- P_4: 90% training - 10% test

6.5.2 Implementation details

The proposed framework is trained for 15.000 episodes using the ϵ-greedy exploration-exploitation scheme for the DQN. Initially, the exploration parameter is set to $\epsilon = 0.9$, and it is gradually reduced every 100 episodes until it reaches zero (0). This marks the end of the exploration phase and initiates the exploitation phase, allowing the learning process to converge towards a (near) optimal policy. The DQN network is updated every 50 time steps using mini-batches of 5000 experiences randomly sampled from a replay buffer containing 200.000 samples. Furthermore, every five (5) episodes, the target network is updated with the weights from the main network (i.e., hard update) to enhance the stability and the performance.

The DQN network has one hidden layer consisting of 150 neurons with ReLU activation functions, which in turn is connected to an output layer with neurons equal to the number of available actions. For the optimization process, the Adam optimizer

is utilized with a learning rate equal to $a = 0.001$ in the exploration phase, which is gradually diminished in the exploitation phase until it reaches a small threshold value. This procedure is important for the stabilization and the convergence of the method.

Regarding the $NN_{altitude}$ network, it is a feed-forward NN designed for regression tasks. Its architecture consists of a hidden layer with 50 ReLU activation units and an output layer that produces a single value, i.e. the altitude. To train this network, the expert (lon, lat) pairs are used as input, and the alt dimension as label.

Another crucial implementation detail involves the termination conditions of the trajectories. Specifically, the generation process is over when one of the following conditions is met:

1. The aircraft's position lies within a 5km radius from the destination airport. This indicates a *successful episode*.

2. The aircraft's position is outside of a predefined bounding box that geographically surrounds the space between the departure and the destination airports.

3. The number of steps exceeds a maximum threshold value, which in this case is set to 2000 time steps.

Note that the violation of one of the last two conditions results to a *failed episode*.

A last remark is that for aiding the optimization procedure, a positive high reward is given to the agent in case of successful termination of episode, and a negative high reward as penalty in failure cases. This technique is referred to as *reward augmentation* [140], and can serve as a general framework for incorporating prior knowledge into imitation learning by offering additional incentives to the agent without interfering with the learning process.

To compare the proposed method, a BC scheme is used as a baseline. This scheme is implemented by training an SVM classifier that takes the four (4) spatio-temporal and the six (6) meteorological features per state as inputs, and the best action (i.e., heading angle) among the set of possible actions as the label. Specifically, the SVM classifier used RBF kernels, and was trained on the historical data.

The proposed approach is evaluated on its ability to predict the flight trajectories according to the following two (2) measures:

Table 6.1: Comparative results of the proposed $AppLearn$ and the BC approaches in terms of the success rate and the RMSE metric.

Dataset	**Partition**	**% success rate**		**RMSE (km.)**	
		AppLearn	*BC*	*AppLearn*	*BC*
Cluster 1 (250 trajectories)	P_1	61	0	26.88	-
	P_2	96	38	22.45	30.54
	P_3	96	50	17.54	28.42
	P_4	98	95	14.43	25.91
Cluster 2 (278 trajectories)	P_1	72	0	20.61	-
	P_2	94	40	20.53	27.90
	P_3	97	90	17.42	24.55
	P_4	99	97	13.81	23.38

- **Success rate (%)**: the percentage of the successful episodes, i.e. predicted trajectories reaching the destination airport within a 5km radius
- **RMSE (km.)**: Root mean squared error between the predicted and the true trajectory. This is found by initially calculating the *Dynamic Time Warping (DTW)* distances of them, then detecting their matching points, and finally computing their root mean squared error (RMSE) concerning only the 3 spatial features (longitude, latitude, altitude)

The reported results are the mean values obtained from ten (10) independent experiments.

6.5.3 Results

The results obtained from applying both methods, *AppLearn* and *BC*, to the two (2) experimental datasets corresponding to the two (2) clusters are presented in Table 6.1. The results indicate that the *AppLearn* method is capable of learning efficient policies that produce trajectories with high success rates, which are close to those of the experts measured by the RMSE. Additionally, compared to the *BC* method, it is evident that the *AppLearn* method consistently produces better solutions for every partition scheme in both clusters, while also having a higher percentage of successfully terminated trajectories.

Specifically, the *AppLearn* method provides the best results for both clusters in the P_4 partition (90% of expert trajectories used for training), where it manages to successfully produce 98% and 99% successfully predicted trajectories with an average RMSE of $14.43\ km$ and $13.81\ km$ for cluster 1 and cluster 2, respectively. Moreover, the results on P_2 and P_3 partitions also yield high success rates, but the average RMSE is also increased in both cases. On the other hand, the *BC* method cannot generalize well when the training set is reduced, resulting to inferior performance.

It is interesting to note that the proposed *AppLearn* method is able to create a satisfactory policy even in the most difficult and least informative partition scenario (P_1), where it is trained on only one expert trajectory, i.e. the centroid of the cluster. The resulting trajectories showed a high success rate, with 61% and 72% of all produced trajectories being successful, and acceptable average RMSE values. This is a promising result and demonstrates the effectiveness of the proposed method in predicting trajectories even in the absence of large datasets or multiple demonstrated trajectories. In contrast, the *BC* method failed to produce any successful trajectories, as shown in Table 6.1.

Figure 6.3 shows an example of the *AppLearn* learning progress in terms of *average RMSE* and the calculated *average reward* on all training trajectories. These learning curves are the averages of ten (10) independent runs in the case of partition scenario P_3. Note that they include measurements from all the tested trajectories, successful or not. The latter explains the bell shaped curve of the *average RMSE* evaluation metric, since in the beginning of the training the number of successful trajectories reaching the destination is very low.

It is noteworthy to observe the convergence of the proposed method once it finishes the exploration phase and enters the exploitation phase. During the exploration phase (until episode 11.200), a considerable amount of randomness is introduced due to the $\epsilon - greedy$ approach, and thus the produced RMSE from the roll-outs is noisy. In the exploitation phase, learning gradually stabilizes until it reaches convergence. Additionally, the *average reward* remains negative for most of the exploration phase, but constantly increases until it finally converges in the late episodes of the exploitation phase. This highlights a significant advantage of the proposed method, demonstrating its ability to construct a high-valued reward scheme and fit well to the expert's behavior simultaneously.

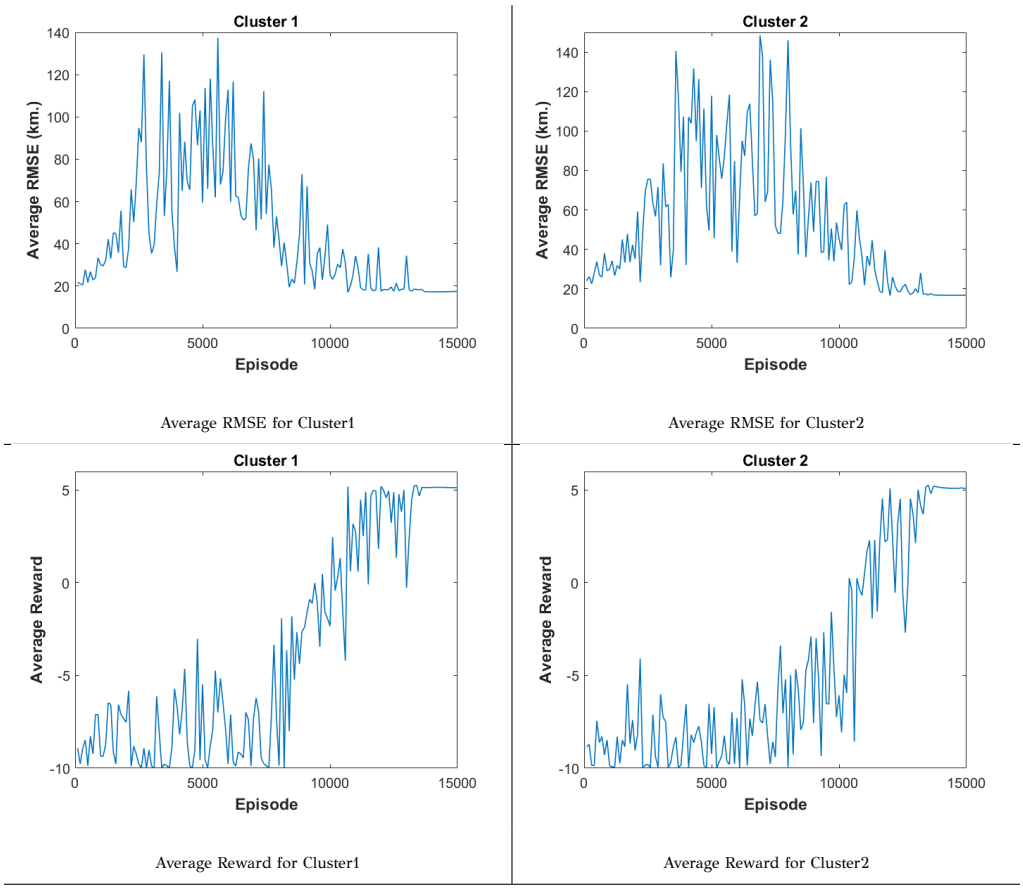


Figure 6.3: Learning curves for the Average RMSE and the Average Reward for both clusters.

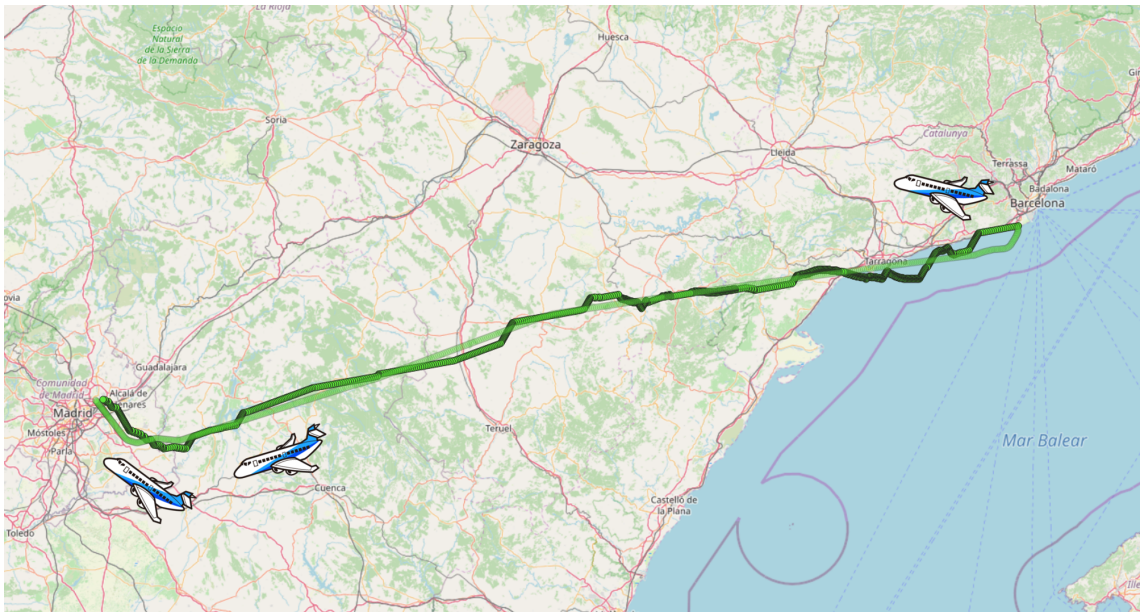


Figure 6.4: An example of a generated trajectory (opaque green line) versus the corresponding expert trajectory (transparent green line).

Finally, Figure 6.4 presents an example of a predicted trajectory provided by *AppLearn* after finishing the learning process. An interesting observation is that the learned policy manages to steer the plane before landing to Madrid and to imitate the local pattern of the expert trajectory, where the actual trajectory veered towards the North.

6.6 Summary

In this chapter, an IRL approach is presented for trajectory modeling in the aviation domain, called *AppLearn*. The method leverages historical expert trajectories as training examples in order to learn an efficient model for the reward function and imitate the expert's policy that generated the demonstrated trajectories.

The state space is constructed using feature vectors that encode spatio-temporal and meteorological information, while the action space consists of discrete aircraft heading angles. To train the Q-value function approximation scheme and enable decision-making, a DQN-based neural structure is employed. Finally, the results are highly promising, demonstrating the ability of the proposed *AppLearn* method to accurately predict trajectories and exhibit generalization capabilities even when using few demonstrations.

Chapter 7

Modular and Multi-modal Generative Adversarial Imitation Learning for modeling Flight Trajectories

This chapter addresses the modularity of trajectories in conjunction to multi-modality (i.e., patterns of behavior), towards imitating the execution of historical aircraft trajectories. Furthermore, it proposes an imitation learning framework that segments demonstrated aircraft trajectories into sub-trajectories corresponding to different flight phases. This segmentation helps in separating different modalities and learning a mixture of policies for each flight phase. Domain-specific rules are used in order to segment the trajectories, and generative multi-modal imitation learning methods are employed to learn the mixture of policies. This modular approach

enables accurate prediction of modalities and sub-trajectories, resulting in the prediction of the aircraft state evolution throughout the entire trajectory in a compositional manner.

7.1 Overview

Aircraft trajectories, similar to other types of trajectories, possess unique characteristics, imposing challenges and opportunities for trajectory modeling methods. One of the main challenges is that these trajectories may exhibit diverse patterns of behavior or *modalities*, even for the same pair of origin and destination airports. This variability is due to a multitude of factors, including flight parameters, which airlines consider sensitive business information, route charges, weather conditions, air traffic, delays, air traffic manager instructions, and network manager regulations. Secondly, these trajectories comprise distinct phases that can be determined by specific rules for each origin-destination airport pair.

Data-driven trajectory modeling methods either aim at disentangling existing modalities (patterns of behavior), which are mostly represented by means of latent variables, or at segmenting trajectories to sub-trajectories, either through hierarchical task decomposition depending on the needs and objectives. Additionally, ML methods for modeling and predicting trajectories, in general, fail to address the existence of multiple modalities and trajectories' modules (i.e., sub-trajectories for sub-tasks) in a conjunctive manner. Specifically for aircraft trajectories, existing trajectory modeling and prediction methods are modality-agnostic, or do not exploit information about flight phases during the prediction process.

Motivated by these challenging issues, this study aims to build models that allow the incorporation of information about trajectory modules, and modalities. Our early experiments with flight trajectories, showed that due to the existence of various mixtures of modalities in distinct flight phases, modeling trajectory modules separately and disentangling their existing modalities, can boost the performance by approximately 20% in spatial 3D prediction, and approximately 55% in predicting estimated time of aircraft arrival, compared to module and modality-agnostic approaches. This improvement is due to the profound impact of modeling individual trajectory modules and their modalities, explicitly. Indeed, modalities exhibit striking variations in

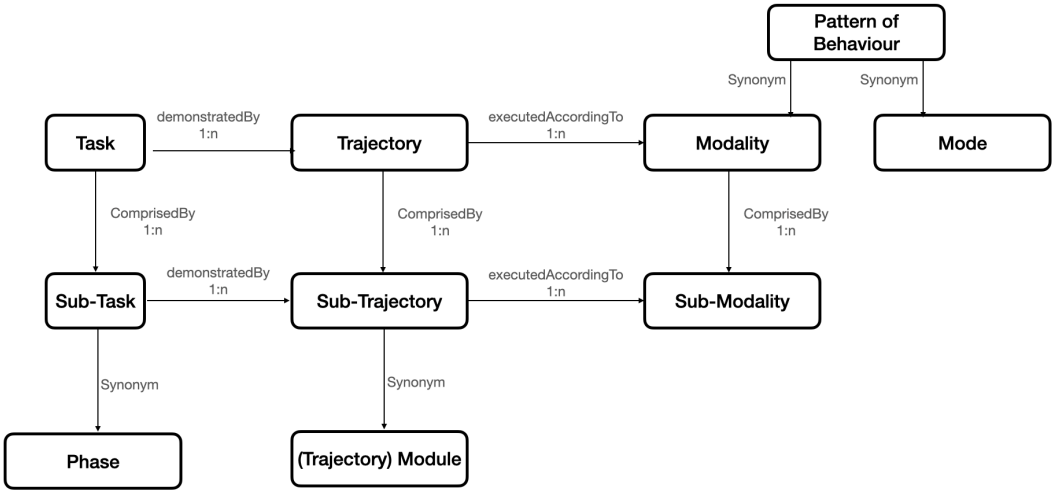


Figure 7.1: Terms used throughout the chapter and the relations between the corresponding concepts.

executing trajectories, with challenging transitions between modalities moving from flight phase to flight phase. This observation presents a substantial challenge to the conventional assumption of a consistent behavioral mode throughout the entire trajectory. Therefore, it is crucial to develop modular, multi-modal approaches that can effectively capture and adapt to the unique characteristics of each phase, enhancing modeling robustness and prediction precision: Addressing modularity and multi-modality conjunctively, is highly challenging.

In this chapter, the trajectory modeling problem for the prediction of trajectories is formulated as an IL problem, exploiting unsegmented historical trajectories. Specifically, it proposes an IL framework which segments aircraft trajectories into sub-trajectories corresponding to flight phases. According to the proposed approach, this facilitates disentangling modalities per flight phase and supports learning a mixture of policies per flight phase, towards predicting the evolution of spatio-temporal aircraft states per flight phase. In the end, the whole trajectory is formed by composing sub-trajectories corresponding to distinct phases.

To clarify the terminology, Fig. 7.1 shows the terms used throughout the chapter to describe the modular, multi-modal aircraft trajectories IL problem, with relations between the concepts that these terms denote.

The contributions of this study are as follows:

- It formulates the problem of imitating modular and multi-modal aircraft trajectories. This entails breaking down the trajectory imitation problem into multi-

ple sub-problems, each focusing on imitating a specific trajectory module (flight phase). The objectives of these sub-problems are twofold: firstly, to predict the modality and execution of each sub-trajectory, and secondly, to predict the progression of the aircraft's states throughout the entire trajectory by combining the predicted sub-trajectories.

- It proposes an IL framework for the aircraft trajectory prediction problem exploiting the modularity of aircraft trajectories. In so doing, it proposes: (a) identifying trajectory modules by decomposing the trajectories into meaningful sub-trajectories corresponding to distinct flight phases, (b) clustering sub-trajectories to detect modalities (patterns of behavior) of flight phase execution, and (c) training a mixture of policies per flight phase, where each policy corresponds to a mode of phase execution.

- It investigates the use of state-of-the-art multi-modal IL algorithms, following a supervised way of disentangling trajectory modes, enhanced with advanced adversarial loss schemes.

- The effectiveness of the proposed framework is assessed by applying it to model 4D aircraft trajectories using a real dataset comprising long flight trajectories between Paris and Istanbul. This evaluation serves to demonstrate the practicality and advantages of the proposed method in modeling extended trajectories and accurately predicting aircraft behavior over significant prediction horizons. The results showcase the capability of the framework to achieve high levels of prediction accuracy in long-range trajectory forecasting.

The structure of this chapter is as follows. Section 7.2 reviews the literature on the research topic of aircraft trajectory prediction and also on multi-modal IL. Then, Section 7.3 specifies the problem to be solved, and Section 7.4 describes the proposed data-driven IL algorithms for modeling multi-modal aircraft trajectories. Finally, Section 7.5 presents the experimental cases and the obtained results, and Section 7.6 concludes the chapter.

7.2 Related work

7.2.1 Multi-modal IL

In IL, the agent learns to perform a task by exploiting demonstrations of expert behaviors. Some of the most common approaches used in IL include BC, direct policy learning (DPL) [141], IRL, and AIL. The generative adversarial imitation learning (GAIL) [28] is an advanced technique in the field of IL, which leverages the power of GANs to train an agent in order to imitate expert behavior in a given task or environment. In comparison to the apprenticeship learning method (see Chapter 6), which tries to infer the goal of the experts by learning their reward function, GAIL tries to directly copy the experts without the need to find the reward function. A detailed description of GAIL has been given in Chapter 2.3.3.

The limitation of GAIL is its vulnerability in the face of multiple behavioral modes when performing a task, such as when the demonstrations exhibit various expert patterns of behavior. To overcome the multi-modal behavioral hurdle in IL, one may pursue either unsupervised or supervised approaches, depending on whether the behavioral modes are known in advance. An unsupervised extension of GAIL that tries to solve the multi-modal problem is Burn-InfoGAIL [142]. It uses a dynamic Bayesian network to draw latent codes (representing modes) from burn-in expert demonstrations maximizing the mutual information between demonstrations and codes. In VAE-GAIL [143], the latent code is being inferred from a VAE, which is trained to encode demonstration sequences to embedding vectors, while the decoder is responsible for reconstructing the trajectories. In the traffic management domain, some recent works leverage generative models to sample diverse trajectories of variable modalities. In their majority they employ VAEs [144, 145] or GANs [146, 147].

In [148], the latent codes are known before hand and the discriminator is conditioned on the annotated latent vectors. An extension of that work [149] uses an auxiliary classifier to reconstruct the latent vectors and aid the discriminator to set up an adversarial loss for the generator. In [150], the authors use a task variable with the role of discriminating different contexts, so as to learn diverse reward functions and policies for multiple tasks. Furthermore, in [151] the agent learns a multi-modal imitation policy by optimizing a sparse coding lifelong intention dictionary, which enables agents to imitate various behaviors. The algorithm follows the structure of vanilla GAIL, with an additional term in the objective function for rewarding state-

action pairs that help at the inference of the latent variable. In a recent work [152], the authors study the problem of learning from multi-modal demonstrations where the experts have different dynamics than the imitator, and in [153, 154] the authors propose multi-modal trajectory prediction approaches with conditional variational autoencoder (cVAE) in combination with BC and GAIL, respectively. Specifically, in [153], the whole expert trajectories are being encoded using the cVAE, and the latent variable (encode) is used to condition the policy of BC, while in [154] the cVAE extracts the latent variable using a part of the trajectory and then use it to model the GAIL policy. These methods do not align with the specific goals of this study, which are to disentangle modalities and imitate trajectories by exploiting unsegmented trajectories. For instance, VAE methods require a part of the trajectory as input to generate the latent code.

Additionally, some proposals use attention mechanisms to predict trajectories in a multi-modal manner [155, 156]. Others consider multi-modality as a distribution fitting and sampling problem, and introduce generative models to solve it [146], while some address the modality prediction problem as a classification problem [157]. All the above methods do not utilize historical trajectories in order to imitate the behavior of experts, hence they cannot be directly compared with the methods proposed in this chapter.

As far as current knowledge goes, there are no state-of-the-art ML methods available to predict modular and multi-modal aircraft trajectories. This chapter explores the use of two state-of-the-art multi-modal extensions of GAIL: (a) Info-GAIL [158], which has been modified to a supervised version here, and (b) Triple-GAIL [159]. These methods demonstrate state-of-the-art performance in disentangling multi-modal demonstrations, while identifying accurately behavioral models.

7.2.2 Modular and hierarchical RL

Modular RL approaches decompose a problem into a collection of RL modules, each of which learns a separate policy to solve a portion (corresponding to a sub-task) of the original problem [160, 161].

A family of approaches in [162, 161] employs a state predictor for each sub-task, modeling environmental dynamics. The RL controllers associated with the sub-tasks are dynamically chosen based on prediction error at any particular moment. An

alternative scheme is to introduce a responsibility signal to weight outputs of multiple models and gate their learning. However, this increases the complexity of learning and may degrade the overall performance.

Hierarchical RL breaks down a RL problem into a hierarchy of sub-problems, where high-level tasks utilize lower-level tasks as primitive actions (e.g. [95], [96],and [97] are some of the early approaches). The low-level tasks can themselves be RL problems with even lower-level tasks. Thus, task decomposition re-formulates the original task into a sequence of sub-tasks, at different levels of abstraction.

Recent works in the field of HRL utilize the power of deep NNs to determine sub-goals for task execution (e.g. in [101, 163]) or useful representations of the state space in HRL tasks [164]. Authors in [165] propose the HAC method that combines learning a multi-level hierarchy of sub-goals and a hindsight experience replay memory to learn a policy that selects the optimal sub-goals at each state.

In addition to these efforts, compositional structured RL, either assumes full knowledge of the correct partition of the tasks [166], or automatically learns the structure and the task [167, 168, 169, 170]. Moreover, the work presented in [171] proposes a HRL approach to learn both compound and composable policies within the same learning process.

The proposed methodology of the present study takes advantage of the semantics of aircraft trajectories to identify sub-tasks that correspond to distinct flight phases. In so doing, it does not follow an HRL approach, but rather a compositional approach to IL, where sub-trajectories for executing flight phases are predicted according to (sub-)trajectories' demonstrations, and these predictions are composed to produce the prediction of the whole task.

7.2.3 Aircraft trajectory prediction

Nowadays, aircraft trajectory predictors are based on mechanistic formulations of the aircraft motion problem. Predictors' outputs are generated based on a-priori knowledge of the flight plan (i.e. airline's planned and intended trajectory), the expected command and control strategies released by the pilot, or the flight management system instructions (known as aircraft intent [172]), and the aircraft performance. Aircraft intent together with very precise aircraft performance models, such as the prominent

Base of Aircraft Data (BADA) [1], has helped to improve the prediction accuracy. However, these mechanistic, model-based approaches require information that typically is not known at prediction time, given that it includes airlines' business sensitive information (i.e. values of parameters such as initial aircraft weight /mass payload, cost index, pilot flight modes, etc.). Therefore, mechanistic prediction accuracy is reduced considerably beyond a limited prediction horizon of approximately ten (10) minutes. In addition, these solutions are not able to predict how the trajectory is finally "shaped", based on evidence from past flights, showing trends, preferences and strategies of stakeholders. We conjecture that this ability to comply with the ways in which flown trajectories evolve in space and time will significantly advance the current abilities towards implementing the TBO paradigm.

Various efforts in the field of data-driven aircraft trajectory prediction have explored the application of statistical analysis and ML techniques with different objectives. Linear regression models [173, 174], and NNs [175, 176, 177], have improved the trajectory prediction accuracy for traffic flow forecasting, while generalized linear models [178] have been applied for the trajectory prediction in arrival management scenarios. Multiple linear regression methods [179, 180] have been used for predicting estimated times of arrival (ETA). These efforts include as input historical surveillance data, and they usually require additional supporting data for accurate trajectory predictions (e.g. flight plans, airspace structure, ATC procedures, airline strategy, weather forecasts, etc.), depending on their objectives.

State-of-the-art data-driven approaches in the ATM domain that are closely related to the methods proposed in this chapter are those in [177, 181, 182, 183]. Authors in [181] introduce a stochastic approach, modeling trajectories in space and time by using a set of 4D spatio-temporal data cubes, enriching them with aircraft motion parameters and weather conditions. This approach computes the most likely sequence of states derived by a HMM, which has been trained over trajectories enriched with weather variables. The algorithm computes the maximal probability of the optimal state sequence, which is best aligned with the observation sequence of the aircraft trajectory. According to this approach, the space and time are discretized, with the lateral cube resolution being 13km and with the temporal cube resolution being one (1) hour.

[1] Base of Aircraft Data, https://simulations.eurocontrol.int/solutions/bada-aircraft-performance-model/

In [177] the authors propose a tree-based matching algorithm to construct image-like feature maps from high-fidelity meteorological datasets. Then, they model the trajectory points as conditional Gaussian mixtures with parameters to be learned from the proposed deep generative model, which is an end-to-end convolutional RNN that consists of a long short-term memory (LSTM) encoder network and a mixture density LSTM decoder network. This approach requires information about flight plans, and a set of actual trajectory points, prior to prediction, which constrains the prediction task.

The approach in [182] is a "constrained" approach, learning the deviations of trajectories from flight plans and reporting deviations per waypoint. This is in contrast to the proposed unconstrained approach, which aims to predict the evolution of aircraft state without exploiting any additional information regarding the predicted trajectory. Finally, a recent work [184] studies the use of a deep encoder-decoder architecture for aircraft trajectory prediction in a specific operational phase (cruise phase) and for a limited prediction horizon.

The first work that formulates the aircraft trajectory prediction problem as an IL problem is the one described in [185]. This work provides significant results regarding uni-modal trajectory predictions, also in comparison to other state-of-the-art methods (e.g. [181, 182]). Applearn [183], an apprenticeship learning IL approach for the trajectory prediction problem, proposed as an alternative to the above approach, and has been thoroughly presented in Chapter 6.

In this chapter, to a greater extent than the aforementioned approaches, the trajectory prediction problem is formulated as a modular, multi-modal IL problem in a continuous state-action space, without considering constraints to which the trajectory must adhere. The aim is to identify trajectory modules corresponding to flight phases and disentangle different modalities with respect to these phases with the objective to learn a multi-modal policy per phase, and to predict the modality and execution of each phase. Ultimately, these predictions of trajectory modules are composed to predict the 4D evolution of the whole trajectory.

7.3 Problem setting

As already pointed out, IL methods typically assume that unsegmented expert demonstrations (i.e. trajectories in executing a task) come from a single expert. Modality-agnostic methods focus on imitating the expert behavior based on the assumption that there is a single modality. However, in real world applications, variability among demonstrations for executing the same task even under the same conditions can be explained by underlying factors, which in many cases are unknown or hidden to an external observer (e.g. due to business sensitive data).

While in Chapter 6 there is the underlying assumption that historical trajectories follow a single modality from origin to destination, usually there exist multiple modalities in executing a flight, and in addition to that, each trajectory comprises modules executing distinct flight phases: This combination of modalities with modules results into modular and multi-modal trajectories, where each of the modules may be comprised of a distinct mixture of sub-modalities.

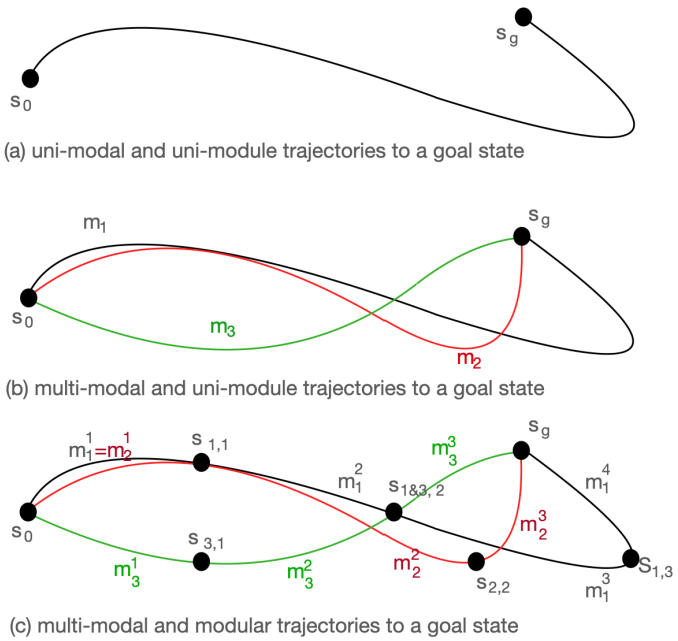


Figure 7.2: From uni-modal and uni-module trajectories, to multi-modal modular trajectories.

To better explain the problem and the proposed approach, Fig. 7.2 shows cases of trajectories in a more general setting. Each line (black, red and green) in this

figure exemplifies a distinct pattern of behavior towards a goal state s_g rather than a single trajectory. The symbols m_y and m_y^x denote the modality and sub-modality, respectively, of the x_{th} module and of the y_{th} pattern, while the symbols s_y^x denote intermediate sub-goals for the x_{th} module of the y_{th} pattern(s). In detail, the case depicted in Fig. 7.2(a) is the simplest case with unsegmented trajectories following a single modality. The case in Fig. 7.2(b) shows three modalities, m_1, m_2, m_3 towards achieving the goal state s_g, with unsegmented trajectories. No modules, neither sub-modalities (i.e., modalities for sub-trajectories) exist in this case. The case in Fig. 7.2(c) shows a more complex case where trajectories are segmented into modules achieving intermediate goals, towards the final goal s_g. It must be noticed that while there are three main modalities for the trajectories, there exist many sub-modalities for sub-trajectories, nearly one for any of the trajectory modules. Moreover, some sub-modalities may be similar, as for instance the sub-modalities for the first modules of the black and red colored modalities (i.e., m_1^1, m_2^1).

To imitate such a set of modular, multi-modal trajectories, it requires a set of expert policies' mixtures. One option is to have a mixture per modality m_1, m_2, m_3, where each mixture comprises one policy per module. This requires learning one mixture of policies per modality, which may lead to not being able to factor out and identify similar sub-modalities between modalities. Therefore, policies for similar sub-modalities may be learned independently, and they may not generalize effectively in learning policies for sub-tasks. As an alternative, one can have a set of policy mixtures per sub-task, so that each mixture comprises one policy per task sub-modality. For instance, in the case of Fig. 7.2(c), three sub-tasks may be identified: The first (let us name this "climb") corresponds to the first modules of trajectories, and can be executed by following any one of the two sub-modalities, $m_1^1 = m_2^1$ and m_3^1. The second sub-task (called "cruise") can be executed by following any of the subsequent sub-modalities m_1^2, m_2^2 and m_3^2, and the third (called "descent") by any of the sub-modalities m_1^4, m_2^3 and m_3^3 towards the final goal. It must be noted that the first modality (black) has an additional module corresponding to sub-modality m_1^3 which prolongs the execution of the second sub-task towards landing. In the worst case, one policy per sub-modality may be needed, but since sub-modalities can be shared between sub-tasks (as for instance, $m_1^1 = m_2^1$), policies can be factored out and be modeled once for sub-modalities shared between sub-tasks.

From this example it becomes clear that: (a) sub-tasks are in general associated

with mixtures of sub-modalities, which may differ between sub-tasks (for instance the first sub-task has two sub-modalities, and the third sub-task has three), but (b) one can generalize and learn policies for sub-modalities shared in executing sub-tasks. Sub-tasks that are somehow aligned are indeed the "phases" of executing the initial task. Nevertheless, there are two challenges that need to be addressed: (a) determining how to identify phases (or aligned sub-tasks) within the trajectories, and (b) finding a way to identify the sub-modalities present within these phases, which can also be shared among multiple trajectories.

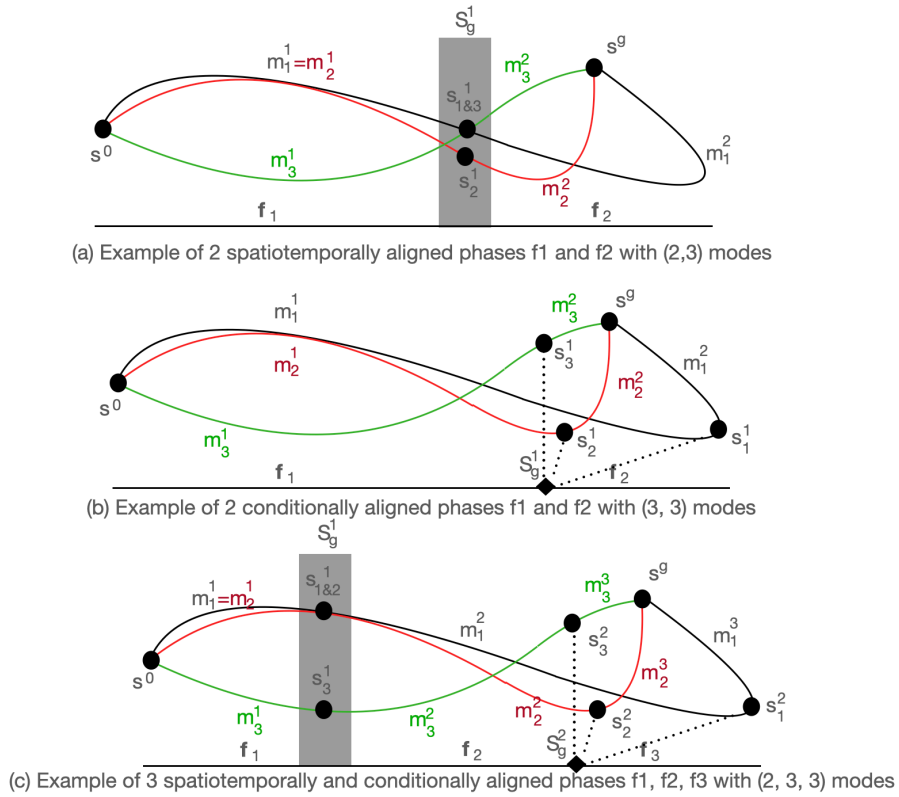


Figure 7.3: Identification of phases.

Figure 7.3 shows three possible approaches for resolving the first issue. In case (a) the goals of sub-tasks are aligned based on spatial and/or temporal criteria. For instance, the first sub-task, for any of the three modalities, should reach any state that is within a region defined by spatial and/or temporal criteria, denoted as S_g^1. This results into two modules corresponding to phases f_1 and f_2 as depicted in the example. Case (b) segregates tasks based on conditionally defined goal states. In this case, there are certain conditions to be met by goal states in S_g^1, which may involve spatial/temporal features together with other state features. These states may not be

temporally or spatially aligned. This segments trajectories into modules corresponding to the aligned phases f_1 and f_2 in the depicted example. In case (c) trajectories are segmented to modules that are spatially/temporally and conditionally aligned. In this case, trajectories are segmented in three modules corresponding to phases f_1, f_2 and f_3. Finally, in any of the cases presented, each phase comprises a mixture of sub-modalities, which may also depend on the set of phases identified.

In this study, the flight phases of aircraft trajectories are aligned by establishing specific criteria that determine the end of the "climb" phase (signifying the start of the "cruise" phase) and the end of the "cruise" phase (signifying the start of the "descent" phase). By doing so, three well-aligned phases are identified for trajectories during flights: "climb", "cruise", and "descent". However, it is important to note that this is just one way of handling it, and it is possible to specify additional phases by defining alternative criteria, which allows for a more detailed breakdown of sub-tasks. For instance, the "descent" phase could be further segmented into entering the terminal management area, executing holding patterns, and approaching the destination airport.

It is crucial to highlight that the objective is not to identify the phases with high accuracy, but rather to establish consistent identification. The key focus is on ensuring that sub-tasks are identified consistently in both the training and test datasets, even if the identification process is done roughly or approximately. The emphasis is on maintaining consistency rather than achieving precise delineation of the phases.

Once the phases are identified, the next step is to address the second challenge: recognizing the sub-modalities within each phase during execution. To achieve this, a clustering approach is employed on historical sub-trajectories that correspond to each phase. This clustering process helps identifying distinct patterns of execution of the sub-tasks within the phases. By focusing on clustering sub-trajectories, it becomes possible to recognize the shared sub-modalities among trajectories more effectively. This approach proves more advantageous than clustering entire trajectories to identify trajectory modalities and subsequently segmenting these modalities into phases.

Overall, given a set of unsegmented trajectories, the overall objectives in this study are to identify meaningful and well aligned trajectory modules specifying phases, disentangle different patterns of demonstrated behaviors per phase, and learn models that under specific circumstances imitate trajectories by choosing (a) the most appropriate modality per phase, and (b) the policy that generates each sub-trajectory. The

following paragraph formulate the modular multi-modal aircraft trajectory prediction problem addressed in this chapter in a more rigorous way.

7.3.1 Flight trajectory modeling as modular multi-modal IL problem

In the previous Chapter, a definition of a trajectory T has been given, which is a chronologically ordered sequence of aircraft states determined by latitude (lat), longitude (lon), geodetic altitude (alt) and timestamp (t). Moreover, the notion of *enriched trajectory* has been thoroughly described along with its most relevant features, such as airspeeds, bearing (c), heading (y), instantaneous aircraft mass (m), as well as, traffic, day of the week/year, cost variable and preferences of airlines. Even though most of them are very informative on the behavior of the aircraft, they are usually hidden since they reveal airlines' business strategies, and in general cannot be considered during modeling aircraft trajectories.

A *predicted trajectory* specifies the future evolution of the aircraft state as a function of: (a) the current flight conditions (e.g. a given state), (b) a forecast of contextual variables (e.g. weather conditions), and (c) a "policy" specifying how the aircraft intends to transit among subsequent states.

Casting the aircraft trajectory prediction problem as a *modular multi-modal IL problem*, then the problem is specified as follows: Given a set T_E= $\{T_{E_i}, i = 1, ...N\}$ of historical aircraft trajectories that have been executed following multiple modalities $\{m_1, ...m_M\}$, the goal is to: (a) specify flight phases $F = \{f_1, f_2, ...f_p\}$ together with the sets of initial, S_0^x, and goal states S_g^x, of each phase f_x, and (b) segment each of the trajectories into p sub-trajectories corresponding to the flight phases. This results into a set T_E= $\{T_E^x, x = 1, ...p\}$ of historical aircraft sub-trajectories, where each set $T_E^x = \{T_{E_i}^x, i = 1, ...N\}$ comprises one sub-trajectory per $T_{Ei}, i = 1, ...N$ and phase f_x. The aim is to learn a mixture of policies $\Pi^x = \{\pi_{m_1^x}^x, ..., \pi_{m_k^x}^x\}$ per phase f_x, where each policy corresponds to one of the k sub-modalities $m_i^x, i = 1, ...k$ executing f_x.

This problem specification is very generic, as it allows specifying any set of phases to segment trajectories, in conjunction to specifying phases' initial and goal states, and allows to factor sub-trajectories into sub-modalities $m_i^x, i = 1...k$ per phase f_x, so as to learn one policy per sub-modality. The execution of a flight trajectory is determined by: (a) the initial state s_0 chosen from S_0^1, (b) the set of sub-trajectory

modalities (one modality per flight phase, i.e. $\{m^1, m^2, ..., m^p\}$), and (c) the set of policies $\{\pi^1_{m^1}, ..., \pi^p_{m^p}\}$ specifying the execution of each of the phases, with respect to the determined sub-modalities.

Any such policy, given a time step Δt, and the initial state of aircraft in phase f_k, i.e. s^k_0, brings the distribution of the state-action pairs of the imitator as close as possible to the distribution of those demonstrated, at any time instant $t = t_0 + (\Delta t * i)$ during the flight phase, predicting how the expert would behave under specific conditions. The prediction concerns determining the evolution of aircraft 4D states in space and time. Following the specification in [186], the set $\mathcal{A}$ of actions contains all the possible combinations of differences in all three (3) spatial dimensions between subsequent trajectory state positions, given the constraint that each such difference is feasible within the constant Δt, and the aircraft's capabilities. This last condition ensures the "flyability" of the trajectory. Specifically, an action is the magnitude of aircraft shift in space: given a position in terms of longitude, latitude and altitude (lon, lat, alt), actions take the form of (Δlon, Δlat, Δalt), and the position in the next state given a constant Δt is:

$$lon_{t+1} = lon_t + \Delta lon \tag{7.1}$$

$$lat_{t+1} = lat_t + \Delta lat \tag{7.2}$$

$$alt_{t+1} = alt_t + \Delta alt \tag{7.3}$$

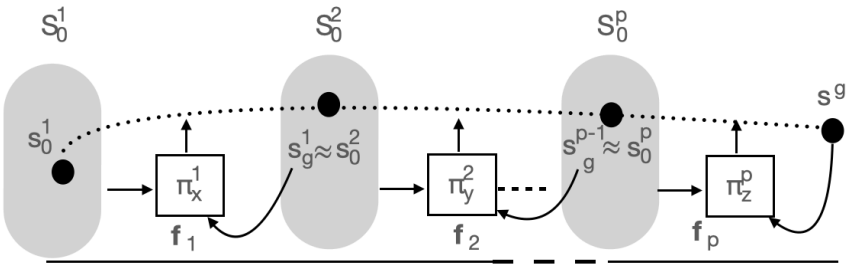


Figure 7.4: The overall process for predicting trajectories.

Figure 7.4 depicts the overall process for predicting trajectories. Given an initial state s^0_1 sampled from S^0_1, the predictor has to decide on the modality $x = m^1$ for the execution of the first phase, and thus on the policy π^1_x which will generate the sub-trajectory for phase f_1, reaching a goal state s^1_g. Then, to continue the execution of trajectory to the next phase, a new initial state s^2_0 is sampled from S^2_0, which should

be "similar" or "near" to s_g^1. The procedure follows iteratively for subsequent phases: Decide on the modality $y = m^2$ for the execution of the second phase, and thus on the policy π_y^2 which will generate the sub-trajectory for phase f_2, reaching a goal state s_g^2, and so on for subsequent phases, until executing the final phase and reaching the goal state s^g.

7.4 Modular multi-modal modeling of aircraft trajectories

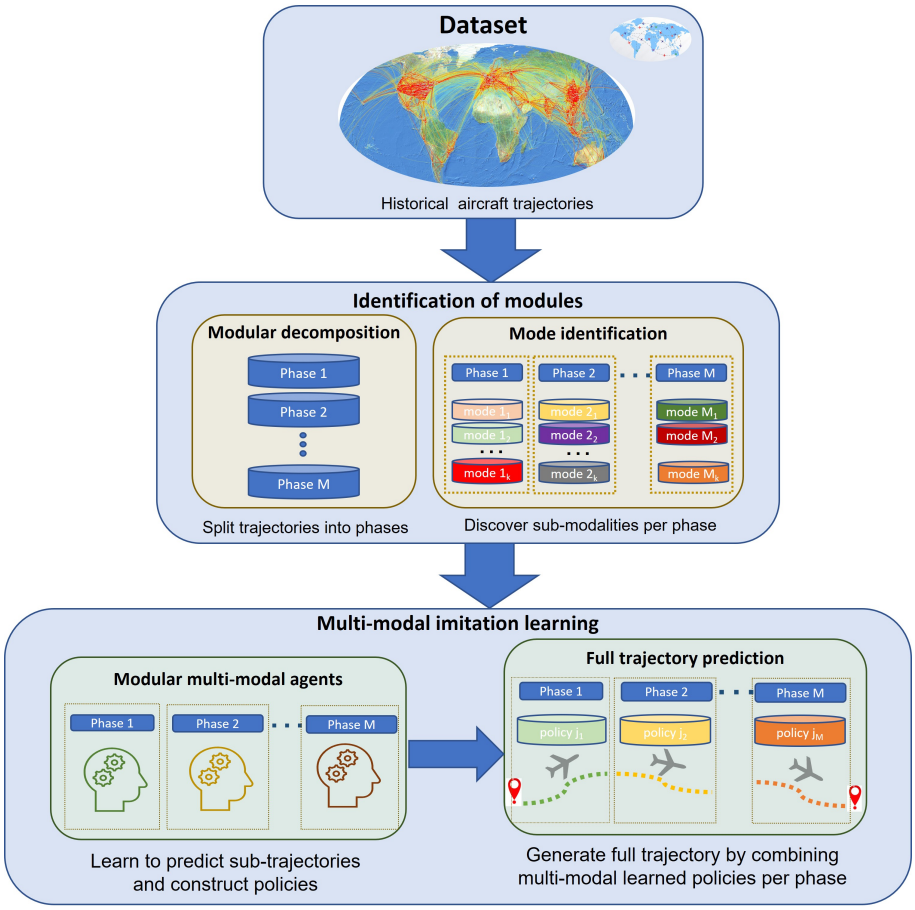


Figure 7.5: Overview of the proposed method for modular and multi-modal aircraft trajectory modeling and prediction.

Figure 7.5 depicts the overall proposed modular multi-modal method for modeling and predicting trajectories via imitation learning. Given a dataset of unsegmented historical trajectories (in our case aircraft trajectories), the trajectory modules are initially identified. This involves segmenting trajectories into phases (modular decomposition stage) and identifying modalities per module (mode identification stage)

Module	Description
Phase 1: Climb	The initial time period during which the aircraft has a zero or positive rate of climb.
Phase 2: Cruise	The time period following the initial climb during which the aircraft is in level flight. This phase also includes en-route climb and descent, i.e., any time period during cruise, where the aircraft approaches a higher or lower flight level.
Phase 3: Descent	The time period before landing, during which the aircraft has a negative rate of climb.

Table 7.1: The selected flight phases together with a brief description.

in an unsupervised manner by clustering sub-trajectories. This results into modular trajectories and multi-modal sub-trajectories corresponding to phases. Phase-specific multi-modal IL agents are trained by exploiting the identified sub-trajectories. These are trained to model trajectories with respect to the dynamics and constraints that arise within each phase. The training of each of these agents results into a mixture of policies, corresponding to phase-specific modalities. Finally, learned policies are used for the prediction of the modality and sub-trajectory per phase. Through composition of sub-trajectories, this results into predicting the spatio-temporal evolution of the whole trajectory.

Subsequent paragraphs describe the functionality and methods used per stage, also detailing how this is tailored to the modeling and prediction of aircraft trajectories.

7.4.1 Identification of modules

This stage, given a set of unsegmented trajectories, identifies sub-trajectories corresponding to flight phases, and identifies the existing sub-modalities per phase.

Long-duration flights typically follow the restrictive sequence pattern of phases "climb", "cruise", and "descent". Although multiple descents and climbs (i.e., "touch-down" maneuvers) are likely to occur in training flights, such occurrences are not typical in commercial flights. Therefore, the transition to the "descent" phase occurs exclusively from the "cruise" phase, and the transition to the "cruise" phase occurs solely from the "climb" phase. Table 7.1 provides an overview of the phases along with a brief description for each phase.

The conditions that define the termination of each phase are based on the vertical profile of the flight. Specifically, for each trajectory point the following are computed: altitude difference ΔH, time difference ΔT and difference of altitude rate ΔA with the previous point. Therefore, a boolean climbing flag is defined, which is true if $\Delta H/\Delta T$ is above a given threshold or $\Delta A/\Delta T$ is positive. For the identification of phases, one should also consider that the "cruise" phase of long-range flights has the longer duration among flight phases. An estimation of the flight duration can be made by observing the duration of the demonstrated expert flights. The "climb" phase ends when the climbing flag is not true, and the flight duration at that point is more than 10% of the estimated flight duration. Finally, the "descent" phase can start only after a "cruise" phase, where the difference of altitudes between consecutive positions is above a given threshold, and the flight duration at the starting point has exceeded 65% of the estimated total flight duration. The last criterion acts as a safety lock against transitions to lower flight levels during the "cruise" phase.

During the investigation of the available flight plans, we discovered that flight phases can be effectively aligned using specific *waypoints*. These waypoints are fixed geographical positions that aircraft pass through when departing from or arriving at an airport. They serve as reference points for transitioning between different flight phases. By leveraging these waypoints, it becomes possible to mark the transitions between phases without relying on vertical profiles, heuristics, or predefined thresholds as previously described. This approach offers a more robust and independent method for identifying phase transitions in any flight, regardless of its vertical profile.

Given that behavioral modes per phase (sub-modalities) are not provided in demonstrations, there is the need to identify them and associate each demonstrated sub-trajectory to the corresponding phase modality. This study considers the agglomerative hierarchical clustering for the identification of sub-modalities, which relies on the bottom-up generation of a tree-like structure of clusters [187]. Each sub-trajectory begins as an individual cluster, and clusters are merged iteratively based on linkage criteria.

It must be noted that the clustering process exploits only the spatial information regarding trajectories, i.e. the $d = 3$ spatial variables of longitude, latitude and altitude. This is done, since the aim is to disentangle sub-modalities in space and time, independently of the factors that determine their choice. A normalized version of the *dynamic time warping (DTW)* method [188] is used in order to calculate the distance

measure of two sub-trajectories of variable length, L_i and L_j, as it is specified in the following equation:

$$DTW_{norm}(L_i, L_j) = \frac{DTW(L_i, L_j)}{\sqrt{d * L}} \quad , \tag{7.4}$$

where $L = max(|L_i|, |L_j|)$. This provides a measure normalized to $[0, 1]$.

Also, the *Ward's* method is utilized to determine the pair of existing clusters to be merged. The outcome of agglomerative clustering is a binary tree, i.e. a *dendrogram*, where the height of the branches represents the degree of dissimilarity between groups that are being joined. Cutting the tree at a specific height is equivalent to inducing a clustering solution. Finally, the number of clusters is estimated using the silhouette criterion [189].

7.4.2 Multi-modal IL methods for trajectory modeling

The segmented trajectories in the output of the previous stage demonstrate the execution of flights' phases, following the phase-specific sub-modalities identified. At this stage, multi-modal IL algorithms exploit these sub-trajectories in a supervised way to model and predict sub-trajectories. Subsequent paragraphs present the proposed algorithms, *Triple-GAIL-GP* and *sInfo-GAIL-GP*. Both methods build upon the GAIL framework by incorporating additional information from multiple modalities, which was described in Chapter 2.3.3. Moreover, a comprehensive description of WGANs is provided in in Chapter 2.1.5, which is the specific variant of GANs that it is integrated into the proposed multi-modal methods to enhance their performance.

One of the primary challenges of multi-modal IL is the *mode collapse* problem. Similar to GANs, mode collapse occurs when the agent fails to explore the full spectrum of expert behaviors and instead converges to a limited set of modes. This lack of diversity can hinder the agent's ability to generalize and adapt to novel situations, as it may fail to reproduce the full range of expert actions. To overcome this limitation, this study proposes incorporating the Wasserstein distance into the objective function of the multi-modal IL methods. The Wasserstein distance offers a more informative and stable training signal by measuring the discrepancy between the distributions of expert demonstrations and the generated trajectories. By adding the Wasserstein distance to the objective, the agent is incentivized to explore and cover a broader range of expert behaviors.

Triple-GAIL-GP

Triple-GAIL [159] is a supervised extension of GAIL that has been proposed to address the problem of multi-modality with the aim to capture diverse behaviors from a set of experts. It uses a latent variable c that represents the behavioral mode and learns conditional policies, $\pi(a|s,c)$.

Triple-GAIL comprises three networks:(a) the generator, G_θ, with parameters θ that follows an actor-critic architecture consisting of a policy network $\pi_\theta(a|s,c)$ and a value function estimation network, (b) the discriminator, $D_w(s,a,c)$, with parameters w that functions as in GAIL, and (c) the selector, $C_\psi(c|s,a)$, with parameters ψ, which infers the latent variable given a state-action pair. The generator and selector networks are trained to produce (s,a,c) samples that are similar to the expert demonstrations, while the discriminator is trained to distinguish the real from the generated samples.

In order to enhance the Triple-GAIL's efficiency, a *gradient penalty* term is integrated in the objective function of the discriminator, as described in the WGAN-GP framework (see Chapter 2.1.5 for more details). We call this *Triple-GAIL-GP* and its objective function can be defined as:

$$\begin{aligned} &\min_{\theta,\psi}\max_{w}\mathbb{E}_{\pi_E}[log(1-D_w(s,a,c))]+\omega*\mathbb{E}_{\pi_\theta}[log(D_w(s,a,c))] \\ &+(1-\omega)*\mathbb{E}_{C_\psi}[log(D_w(s,a,c)]+\lambda_E R_E+\lambda_G R_G-\lambda_H H(\pi_\theta) \\ &+\lambda_{GP}\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(|\nabla_{\hat{x}}D_w(\hat{x})|_2-1)^2] \quad , \end{aligned} \tag{7.5}$$

where R_E and R_G are two cross-entropy terms that guarantee the convergence of the distributions. Specifically, R_E is the standard supervised loss ensuring that the selector converges to the expert distribution and R_G is the divergence between the distributions of the selector and the generator. Also, the corresponding coefficients λ_E, λ_G suggest the importance of the above entropy terms and λ_{GP} regulates the influence of gradient penalty. It must be noted that proper values of coefficients were found experimentally to be $\lambda_E = \lambda_G = 0.5$ and $\lambda_{GP} = 1.0$. The last term of Eq. 7.5 is the gradient penalty which enforces the Lipschitz continuity constraint to the discriminator. The rest terms constitute the standard Triple-GAIL loss [159].

The training procedure begins by initializing the generator with a pre-trained BC model to speed up the convergence. Afterwards, the algorithm samples an initial state from the set of demonstrated expert trajectories along with its true latent variable (mode) and the generator rolls-out the trajectory using the policy $\pi_\theta(a|s,c)$. Concurrently, the selector takes as input the generated state-action pairs and outputs

a latent variable for each one. In doing so, two diverse (s, a, c) sets are obtained: one from the generator with the true latent variable, and one from the selector with the predicted latent variable. The two sets along with the expert (s, a, c) samples, are then passed to the discriminator D_w for judgement. The optimum is met when both the generator's and the selector's (s, a, c) distributions converge to the distribution of the expert samples. Algorithm 8 specifies the training process of *Triple-GAIL-GP*.

Algorithm 8: Triple-GAIL-GP for trajectory modeling

Input: Expert trajectories T^E

Initialize the policy weights θ using BC

Create two buffers for the generator and the selector samples

for *episode = 1, 2, ...* **do**

  while *not enough samples in generator's buffer* **do**

    Sample a starting state from T^E and its true latent variable c

    Roll-out the trajectory with fixed latent variable

    Store state-action-latent tuple (s, a, c) into generator's buffer

    Estimate the latent variable, $\widehat{c}$, using the selector and store the $(s, a, \widehat{c})$ sample into selector's buffer

  Select equal number of samples from generator's buffer, x_G, selector's buffer, x_C, and expert demonstrations, x_E

  Compute the gradient penalty term

  Update w and ψ parameters by taking partial derivatives of Eq. 7.5

  Update θ using TRPO/PPO with the surrogate reward: $r = -logD_w(s, a, c)$

It must be noted that the gradient penalty requires computing the gradient of the discriminator output with respect to the noisy input $\hat{x}$ (Eq. 7.5). The noisy input comes from a distribution combining expert and policy samples. Specifically, the method samples data from these distributions and performs a weighted-average interpolation, creating a new set of samples that lie between the demonstrated and generator's distributions. Finally, the penalty term is the squared difference between the gradient norm of the output of the discriminator applied on the interpolated data and one (1).

sInfo-GAIL-GP

Info-GAIL [158] is another variation of the GAIL algorithm designed to address the issue of multi-modality, which was inspired by InfoGANs (see Chapter 2.1.5). Unlike Triple-GAIL, Info-GAIL utilizes an information-theoretic objective function to learn the policy, instead of using a triplet loss function. Additionally, in order to discover trajectories' modalities, which is its main goal, Info-GAIL uses unlabeled expert demonstrations. In contrast to Triple-GAIL, which employs a selector, Info-GAIL utilizes a posterior approximator (as InfoGAN) to select the mode based on the state-action pairs. Finally, Info-GAIL defines the policy and expert distributions as state-action distributions without conditioning on the latent variable.

In the original version of Info-GAIL, the latent variables associated with the trajectories are unknown and are discovered in an unsupervised way. In the proposed approach, a supervised version of Info-GAIL is structured, because the semantic information (i.e. modality of each sub-trajectory) is available. Hence, the choice of latent variable values is made through the clustering procedure, as described earlier.

Similar to Triple-GAIL, Info-GAIL comprises: (a) a generator G_θ that follows an actor-critic architecture modeling a policy $\pi_\theta(a|s,c)$ and a value function approximator, (b) a discriminator $D_w(s,a)$, and (c) a posterior network $Q_\psi(c|s,a)$ with parameters ψ that approximates the probability of a state-action pair to belong to a specific modality c. Both the discriminator D_w and the posterior Q_ψ networks are used to form a surrogate reward that guides the generator's policy to match the state-action distribution of demonstrated trajectories. Moreover, a gradient penalty term is added in the discriminator's objective function, as explained in the Triple-GAIL algorithm. This version of Info-GAIL is called supervised Info-GAIL, and denoted as *sInfo-GAIL-GP*.

Hence, the objective function of *sInfo-GAIL-GP* with the addition of gradient penalty term becomes:

$$\begin{aligned} \min_{\theta,\psi}\max_{w} \mathbb{E}_{\pi_E}[log(1-D_w(s,a))] + \mathbb{E}_{\pi_\theta}[logD_w(s,a)] \\ -\lambda_H H(\pi_\theta) - \lambda_I L_I(\pi_\theta, Q_\psi) + \lambda_{GP}\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(|\nabla_{\hat{x}} D_w(\hat{x})|_2 - 1)^2 \ , \end{aligned} \tag{7.6}$$

where $L_I(\pi_\theta, Q_\psi)$ is a variational lower bound of the mutual information between the state-action pairs and the latent variable, while $Q(c|s,a)$ is an approximation of the posterior probability $P(c|s,a)$ (similar to InfoGAN). Both regularization coefficients λ_H and λ_I are considered as constants, and have empirically tuned to $\lambda_H = 0.01$ and $\lambda_I = 0.5$ after experimentation. The gradient penalty term has the same form as in

Triple-GAIL-GP.

The training procedure of the *sInfo-GAIL-GP* approach is quite similar with this of *Triple-GAIL-GP*, except that the discriminator D_w and posterior Q_ψ networks use the generated state-action pairs to update their weights (without including the latent variable). The policy weights are updated according to the PPO algorithm that combines the output of the discriminator and the posterior to obtain the surrogate reward:

$$r = -logD_w(s,a) * l_D + logQ_\psi(s,a) * l_Q \ , \tag{7.7}$$

where l_D and l_Q are weight terms that determine the contribution of each network.

Algorithm 9 provides an outline of the *sInfo-GAIL-GP* method.

Algorithm 9: sInfo-GAIL-GP for trajectory modeling

Input: Expert trajectories T^E

Initialize the policy weights θ using BC

Create a buffer for storing generated state-action pairs

for *episode = 1, 2, ...* **do**

  while *not enough samples in buffer* **do**

    Sample a starting state from T^E and its true latent variable c

    Roll-out the trajectory with fixed latent variable;

    Store the generated state-action pairs (s,a) into the buffer

  Select equal number of samples from the buffer, x_G, and the expert trajectories, x_E

  Compute the gradient penalty term

  Update w and ψ parameters by taking partial derivatives of Eq.7.6

  Update θ using TRPO/PPO with the surrogate reward (Eq. 7.7)

Mode selection

The aim of the evaluation phase of the proposed scheme is twofold: Given a phase and an initial aircraft state (a) predict the optimal behavioral mode and (b) predict the subsequent aircraft states until the aircraft reaches a goal state, i.e. a waypoint that signifies the start of the next phase, or the destination airport, in the case of the last phase.

In doing so, the following procedure per phase is followed: The trained generator $G_{\widehat{\theta}}$, and either the trained selector $C_{\widehat{\psi}}$ (for *Triple-GAIL-GP*), or the posterior network

$Q_{\hat{\psi}}$ (for *sInfo-GAIL-GP*) are used to generate M distinct trajectories, equal to the number of identified phase's modalities. Each modality corresponds to a different value of the latent variable c. Then, for each predicted trajectory, the posterior probabilities of the visited state-action pairs are obtained. At the end of this process, the sub-trajectory with the maximum average posterior probability is selected for that phase. The mode of that sub-trajectory indicates the behavioral mode predicted by the method.

7.5 Experimental results

The experimental study aims to evaluate the effectiveness of the proposed modular multi-modal IL scheme, which was trained using a real-world dataset of commercial flights. The proposed scheme is compared against a uni-module multi-modal IL baseline, which does not segment trajectories into phases, but disentangles the modalities and mimics the evolution of trajectories using a mixture of policies that correspond to the identified modalities.

7.5.1 Experimental data

The trajectory dataset comprises real-world commercial flights from Paris Charles de Gaul to Istanbul airport. This set contains 181 flight trajectories, and spans over a period of four (4) months in 2018. Apart from the four (4) variables specifying 4D spatio-temporal information per trajectory point, i.e., longitude, latitude, altitude and the timestamp, trajectories are enriched with five (5) additional variables:

- three (3) weather variables obtained from the Copernicus Knowledge Base (CDS Dataset) including temperature isobaric, u-wind component and v-wind component,
- the aircraft model, represented as one of nine (9) possible integer values, given the set of aircraft types identified in historical trajectories, and
- the flight delay at gate, which is an integer value of the delay before the flight takes off (in minutes).

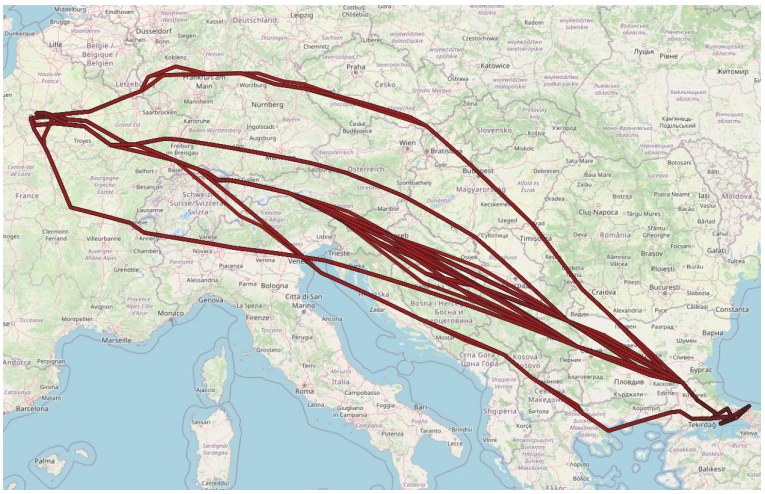


Figure 7.6: The Paris-Istanbul dataset of flight trajectories used during experiments

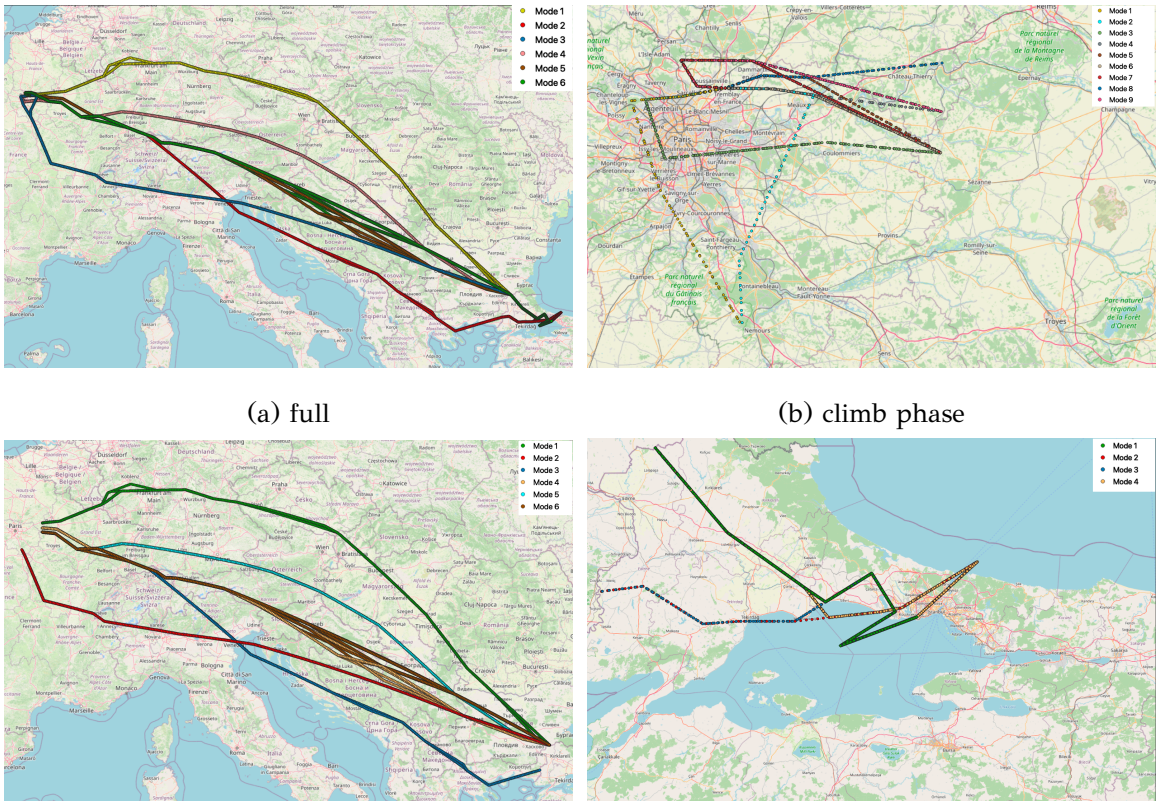
It must be noted that the trajectory dataset has been cleaned and pre-processed to ensure accuracy and reliability in describing trajectories, and in reporting the results. In particular, since the dataset provides only flight plans (i.e. trajectories indicating waypoints crossed, with no constant spatial or temporal distance between consecutive waypoints), an interpolation process between consecutive waypoints for each trajectory was necessary to obtain trajectories with points in a constant time interval, set to $\Delta t = 20$ seconds. The interpolation process assumes constant aircraft speed between consecutive waypoints, equal to the average speed reported in these points. Finally, all continuous state variables are normalized to zero mean with unit variance (z-score scaling). Figure 7.6 shows the trajectories in the dataset, as depicted by the open source visualizer tool QGIS [2].

7.5.2 Identification of modules and modes

As detailed in the previous section, the identified modules correspond to three flight phases (“climb”, “cruise”, “descent”) based on the designated waypoints associated with the departure and arrival at both airports. Figure 7.7 illustrates the result of applying the modules identification strategy to the specific dataset.

Having determined the modules, the sub-trajectories per phase are clustered to identify the sub-modalities. The result of the clustering process can be seen in Figure 7.7, where each cluster is represented with a different color. In particular, the process

[2]https://www.qgis.org/en/site/



(a) full

(b) climb phase

(c) cruise phase

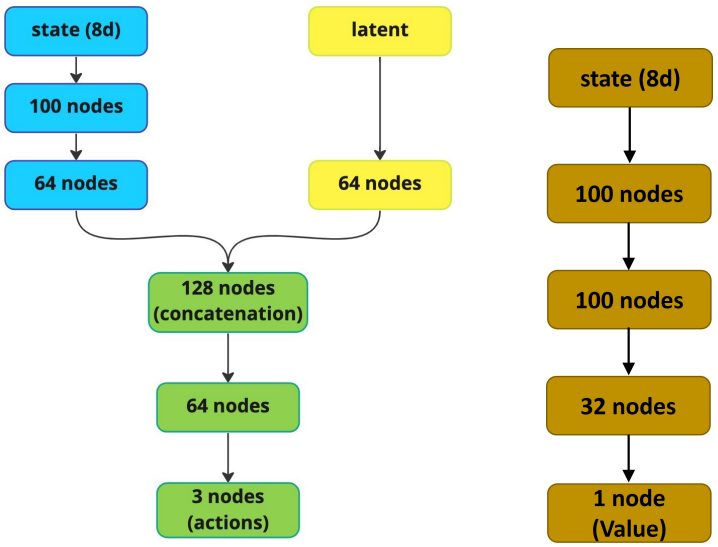
(d) descent phase

Figure 7.7: The result of the hierarchical agglomerative clustering approach in whole trajectories (a) and in sub-trajectories per flight phase (b-d).

ends up with six (6) clusters in the case of whole trajectories, and nine (9), six (6) and four (4) clusters for "climb", "cruise" and "descent" phases, respectively. Note that the clustering solution of the whole trajectories (six clusters) is in accordance with the results of the cruise phase that covers the biggest part of the flights.

According to the clustering procedure, the size of clusters is imbalanced. Even though we could follow a data augmentation strategy over small clusters, we favor a fair sampling scheme during learning so as not to affect the distribution of data. Specifically, a random selection among the available modes (with equal probability) is made iteratively, followed by the sampling of a trajectory from the chosen mode. This sampling scheme resembles data augmentation and also has the advantage of not allowing the generator's buffer to be overpopulated with samples that belong to the largest mode.

7.5.3 Experimental setting



(a) policy network (b) critic network

Figure 7.8: The architecture of the generator consisting of (a) the policy and (b) the critic network used in both Triple-GAIL-GP and sInfo-GAIL-GP methods

This section provides: (a) implementation details about Triple-GAIL-GP and sInfo-GAIL-GP, as well as (b) training details and (c) the measures used for evaluating the accuracy of the proposed methods.

The generator, G_θ, in both methods follows an actor-critic architecture with two (2) networks: a policy and a value function. The policy network (Fig. 7.8(a)) takes as input state features and a latent variable. The state information passes through two (2) fully connected layers before it is concatenated with the latent variable input, which passes through one (1) fully connected layer. Then, the concatenated input is passed through another fully connected layer before it reaches the output layer, which produces the three (3) spatial quantities (i.e., $\Delta lon, \Delta lat, \Delta alt$) that specify the action to be taken. Moreover, the policy outputs are stochastic and represented as the mean Gaussian distribution of each action. It must be noted that the policy parameters are initialized using BC by minimizing the mean squared error between expert and estimated actions, using the Adam optimizer. Regarding the critic network (Fig. 7.8(b)), it takes as input only the state information and consists of three hidden layers with 100-100-32 nodes and an output layer of one (1) node that represents

the value of the state.

On the other hand, the discriminator, D_w, consists of two (2) hidden layers with 100 (*tanh*) nodes each and an output layer of a single node. The difference between the discriminator of Triple-GAIL and that of Info-GAIL lies in the input: The former uses (state, action, latent variable) samples, while the latter uses (state, action) pairs. Finally, the posterior, Q_ψ, and the selector, C_ψ, networks consist of two (2) hidden layers with 100 nodes and their output layers comprises a number of nodes equal to the number of different modes, M, providing the posterior probabilities of modes given a (state, action) pair.

The roll-outs generated by G_θ are terminated if one of the following conditions occurs:

- the trajectory reaches a point that lies within a radius of $10\ km$ from the goal state (target airport or phases' goal states specified by waypoints),
- the trajectory exceeds a pre-defined time-step limit,
- the trajectory reaches a point outside a spatial box which is bounded to include the expert trajectories, or
- the aircraft exceeds an upper or lower altitude limit.

The proposed IL methods are evaluated regarding their ability to predict modalities and model flight trajectories according to the following seven (7) measures:

- **Accuracy** of predicting the modality,
- **Success rate (%)** of methods in terms of reaching the goal state,
- **RMSE** in nautical miles (nm), for each of the spatial dimensions, as well as in all three (3) spatial dimensions (3D),
- **Along-Track Error (ATE)** [190], which is the distance in nautical miles (nm) between the predicted aircraft position and the projection of the corresponding actual aircraft position on the predicted trajectory, across the predicted trajectory,
- **Cross-Track Error (CTE) (or lateral error)** [190], that corresponds to the distance in nautical miles (nm) between the actual aircraft position and the corresponding predicted trajectory point, perpendicular to the predicted trajectory,

- **Vertical deviation (V)** in feet, and
- **Estimated time of arrival (ETA)** error in seconds.

To measure the predicted errors between trajectories with different length, a trajectory-matching step is required. Specifically, the actual trajectory points are matched to the closest predicted trajectory points, as measured by the DTW distance between the two trajectories.

In addition, the growth rate of the prediction error per minute of the prediction temporal horizon is reported. This is crucial, as most of the prediction methods accumulate prediction errors considerably for long prediction horizons, and shows how the modular scheme proposed manages to report low growth rates of prediction errors for long prediction horizons, compared to the uni-module imitation method.

The experimental settings comprise:

- the multi-module multi-modal imitation learning (*MMIL*) proposed scheme, and
- the single-module multi-modal imitation learning (*SMIL*) scheme,

using the *Triple-GAIL-GP* and the *sInfo-GAIL-GP* IL approaches.

It should be noted that these schemes have not been compared to uni-module and uni-modal IL methods (e.g. GAIL) since previous studies have shown that they do outperform them [159, 158]. Finally, all reported results are the mean values of ten (10) independent experiments.

7.5.4 Results

Accuracy of trajectory prediction

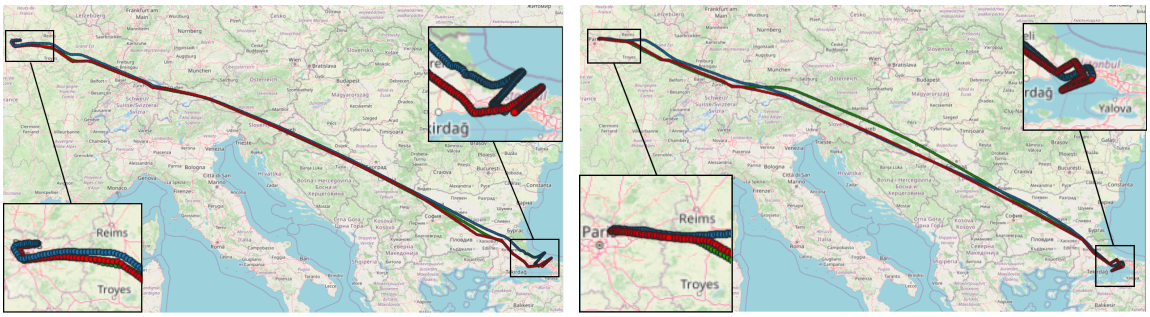
Table 7.2 presents comparative results for *MMIL* and *SMIL* using *Triple-GAIL-GP* and *sInfo-GAIL-GP*, in terms of mean RMSE (in nm) for every spatial dimension (longitude, latitude and altitude) and in 3D of the predicted versus the actual (test) trajectories. For *MMIL*, the column *Total* contains the results of the RMSE for the combined trajectory that occurs after merging the sub-trajectories of each phase, so as to have a fair comparison against *SMIL*. Moreover, it provides the RMSE for any of the distinct *phases* produced by *MMIL*. On the contrary, *SMIL* reports RMSE only for the whole trajectories (last column of Table 7.2).

RMSE (nm)	Method	MMIL				SMIL
		Phases			Total	
		climb	cruise	descent		
long	**Triple-GAIL-GP**	**0.66**	**15.94**	**0.67**	**10.29**	**16.62**
	sInfo-GAIL-GP	0.74	16.98	0.88	10.89	19.39
lat	**Triple-GAIL-GP**	**0.29**	**8.15**	**0.39**	**5.88**	**10.16**
	sInfo-GAIL-GP	0.43	8.70	0.62	6.06	10.54
alt	**Triple-GAIL-GP**	**0.04**	0.21	**0.02**	**0.16**	0.29
	sInfo-GAIL-GP	0.05	**0.19**	0.05	0.17	**0.27**
3D	**Triple-GAIL-GP**	**0.73**	**18.25**	**0.78**	**12.02**	**19.77**
	sInfo-GAIL-GP	0.87	19.27	1.09	12.53	22.36

Table 7.2: RMSE results for Paris-Istanbul test trajectories from origin to destination (total), as well as for three individual phases (climb, cruise and descent).

According to the results, *MMIL* outperforms *SMIL* in every spatial dimension and in 3D showing a significant improvement. An important observation regarding *MMIL* is that the "cruise" phase results in much higher RMSE, which can be attributed to the fact that it constitutes the longest part of the trajectory, hence the IL algorithms accumulate larger errors than in the other two shorter phases. The 3D errors for the other two phases, "climb" and "descent", are approximately the same even if the duration of the "descent" phase is triple from that of the "climb" phase. This can be attributed to the more complex patterns occurring in the "climb" phase of trajectories.

With regards to both multi-modal IL methods, *Triple-GAIL-GP* performs better than *sInfo-GAIL-GP* in every score. In cases where *sInfo-GAIL-GP* achieves a better result than *Triple-GAIL-GP* (e.g. *alt* row in "cruise" phase), the difference is insignificant between the two methods. Finally, the *3D* error for the whole trajectory is always in favor of *Triple-GAIL-GP*.



(a) Best MMIL case (b) Best SMIL case

Figure 7.9: Two trajectory prediction results using MMIL (green) and SMIL (blue) trajectories against actual trajectories (red).

To provide a better intuition regarding the quality of the generated trajectories and demonstrate *MMIL*'s prediction capabilities, Fig. 7.9 presents visualizations of two predicted trajectories following the policies of *MMIL* (green colored) and *SMIL* (blue colored). The actual (test) trajectories are shown in red color, and the *MMIL* trajectory is in its largest part "hidden" behind the actual one. Specifically, Fig. 7.9(a) shows the generated trajectories that correspond to the case of the lowest RMSE (3.86 nm in 3D) using the *MMIL*'s multi-modular policy. The performance of the *SMIL* scheme for the same case was significantly worst (17.57 nm in 3D) than that of *MMIL*. The reason is that the trajectory generated by *SMIL* has a remarkable discrepancy from the actual one during the "climb" and "descent" phases, since it can not accommodate the modalities existing in these phases. Figure 7.9 (b) shows the *MMIL* and *SMIL* generated trajectories in the case where *SMIL* manages to score the lowest RMSE (6.89 nm in 3D). In this case, the performance of the *MMIL* scheme was almost the same (6.98 nm in 3D), where the discrepancy of the predicted from the actual trajectory is mostly in the "cruise phase". It is worth mentioning that the predicted trajectories of the proposed *MMIL* method agree largely with the actual ones, with a noticeable accuracy in the "climb" and "descent" phases.

In addition, Table 7.3 provides comparative results of both methods in terms of ATE, CTE, V, and ETA measures. Results are in compliance with those reported in Table 7.2 showing the superiority of the proposed *MMIL* against *SMIL*. *Triple-GAIL-GP* is also shown to perform, overall, better than *sInfo-GAIL-GP*, especially with regards to the prediction of the whole trajectory (column *Total*). Moreover, the effect of the modular approach is shown clearly in the prediction of the "climb" and

Track errors	Method	MMIL				SMIL
		Phases			Total	
		climb	cruise	descent		
ATE (nm)	**Triple-GAIL-GP**	**0.48**	**10.82**	**0.54**	**5.72**	**10.72**
	sInfo-GAIL-GP	0.53	11.69	0.69	6.11	12.57
CTE (nm)	**Triple-GAIL-GP**	**0.17**	7.94	0.54	**5.79**	**10.65**
	sInfo-GAIL-GP	0.38	**7.48**	**0.36**	5.90	11.69
V (ft)	**Triple-GAIL-GP**	**58.46**	236.10	**34.12**	**202.66**	390.69
	sInfo-GAIL-GP	82.18	**228.79**	56.92	210.81	**357.92**
ETA (sec)	**Triple-GAIL-GP**	**17.11**	**193.05**	**190.38**	**430.09**	**523.85**
	sInfo-GAIL-GP	20.74	217.91	200.94	498.41	655.85

Table 7.3: Comparative results in terms of ATE, CTE, V, and ETA error of generated trajectories from origin to destination. The same results are shown for any of the three individual phases.

"descent" phases.

To further assess the effect of *MMIL* on the prediction errors reported for each of the phases and for the whole trajectory, the *MMIL* policy is compared to the *SMIL* policy when the later is used for generating sub-trajectories separately for each of the phases identified. In this case, *SMIL* uses the initial point of the sub-trajectories of the corresponding phase. The obtained results are shown in Table 7.4 for RMSE errors, and in Table 7.5 for track errors. Comparing these with the results of *MMIL* in Tables 7.2 and 7.3, we can observe the significant improvement in prediction that *MMIL* achieves, in every measure used, especially in the case of the "climb" and "descent" phases.

RMSE (nm)	Method	Phases		
		climb	cruise	descent
long	**Triple-GAIL-GP**	**8.52**	**19.54**	**8.99**
	sInfo-GAIL-GP	9.25	22.42	9.58
lat	**Triple-GAIL-GP**	**2.88**	**14.76**	**3.85**
	sInfo-GAIL-GP	3.55	14.55	6.67
alt	**Triple-GAIL-GP**	**0.29**	**0.69**	**0.15**
	sInfo-GAIL-GP	0.30	0.72	0.26
3D	**Triple-GAIL-GP**	**9.29**	**24.84**	**9.87**
	sInfo-GAIL-GP	10.27	27.06	11.89

Table 7.4: RMSE results for SMIL applied to phases.

Track Errors	Method	Phases		
		climb	cruise	descent
ATE (nm)	**Triple-GAIL-GP**	**5.44**	**11.41**	**5.24**
	sInfo-GAIL-GP	5.74	13.69	5.88
CTE (nm)	**Triple-GAIL-GP**	**3.09**	**8.82**	**5.15**
	sInfo-GAIL-GP	3.93	9.98	6.92
V (ft)	**Triple-GAIL-GP**	**425.51**	**727.12**	**216.12**
	sInfo-GAIL-GP	430.62	767.59	362.08
ETA (sec)	**Triple-GAIL-GP**	**46.07**	**248.15**	**82.23**
	sInfo-GAIL-GP	48.12	250.69	89.77

Table 7.5: ATE, CTE, V, and ETA error of SMIL applied to phases.

Accuracy in disentangling modalities & success rate

		Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6	Policy 7	Policy 8	Policy 9
Mode 1	Triple-GAIL-GP	**0.85**	0.00	0.11	0.00	0.00	0.00	0.02	0.00	0.02
	sInfo-GAIL-GP	0.83	0.00	0.16	0.00	0.00	0.00	0.00	0.00	0.01
Mode 2	Triple-GAIL-GP	0.00	**0.76**	0.00	0.09	0.00	0.13	0.02	0.00	0.00
	sInfo-GAIL-GP	0.00	0.71	0.00	0.12	0.00	0.14	0.02	0.01	0.00
Mode 3	Triple-GAIL-GP	0.11	0.01	**0.86**	0.00	0.01	0.01	0.00	0.00	0.00
	sInfo-GAIL-GP	0.15	0.03	0.79	0.00	0.00	0.03	0.00	0.00	0.00
Mode 4	Triple-GAIL-GP	0.00	0.14	0.00	**0.73**	0.01	0.11	0.01	0.00	0.00
	sInfo-GAIL-GP	0.00	0.20	0.00	0.66	0.04	0.08	0.02	0.00	0.00
Mode 5	Triple-GAIL-GP	0.00	0.00	0.00	0.01	**0.53**	0.13	0.21	0.10	0.02
	sInfo-GAIL-GP	0.00	0.00	0.00	0.06	0.45	0.15	0.22	0.12	0.00
Mode 6	Triple-GAIL-GP	0.00	0.12	0.00	0.12	0.09	**0.47**	0.20	0.00	0.00
	sInfo-GAIL-GP	0.00	0.15	0.00	0.14	0.10	0.43	0.18	0.00	0.00
Mode 7	Triple-GAIL-GP	0.00	0.00	0.00	0.01	0.13	0.17	**0.49**	0.01	0.19
	sInfo-GAIL-GP	0.00	0.00	0.00	0.01	0.16	0.18	0.43	0.00	0.22
Mode 8	Triple-GAIL-GP	0.00	0.00	0.00	0.00	0.14	0.00	0.00	**0.86**	0.00
	sInfo-GAIL-GP	0.00	0.03	0.00	0.00	0.21	0.00	0.02	0.74	0.00
Mode 9	Triple-GAIL-GP	0.00	0.00	0.00	0.00	0.04	0.00	0.16	0.02	**0.78**
	sInfo-GAIL-GP	0.01	0.00	0.00	0.00	0.09	0.00	0.20	0.01	0.69

Table 7.6: The mean posterior probabilities of the climb phase as calculated by the selector (Triple-GAIL-GP) and the posterior (sInfo-GAIL-GP) networks in the proposed MMIL scheme

In order to show the accuracy for the prediction of sub-modalities, Tables 7.6, 7.7 and 7.8 present the average posterior probabilities obtained from the selector (*Triple-GAIL-GP*) and posterior (*sInfo-GAIL-GP*) networks on the testing trajectories. To begin with, it should be noted that both multi-modal IL methods are capable of perfectly disentangling the modalities in each phase. However, *Triple-GAIL-GP* outperforms *sInfo-GAIL-GP* in terms of higher posterior probabilities in every case, which demonstrates the robustness of this method.

		Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6
Mode 1	Triple-GAIL-GP	**0.92**	0.02	0.00.00	0.03	0.02	0.01
	sInfo-GAIL-GP	0.87	0.05	0.00	0.06	0.01	0.01
Mode 2	Triple-GAIL-GP	0.00	**0.84**	0.01	0.09	0.04	0.02
	sInfo-GAIL-GP	0.00	0.80	0.03	0.14	0.01	0.02
Mode 3	Triple-GAIL-GP	0.00	0.00	**0.83**	0.07	0.03	0.07
	sInfo-GAIL-GP	0.00	0.00	0.75	0.10	0.10	0.05
Mode 4	Triple-GAIL-GP	0.00	0.02	0.06	**0.55**	0.08	0.29
	sInfo-GAIL-GP	0.00	0.01	0.09	0.45	0.12	0.33
Mode 5	Triple-GAIL-GP	0.01	0.01	0.02	0.09	**0.68**	0.19
	sInfo-GAIL-GP	0.02	0.03	0.02	0.14	0.61	0.18
Mode 6	Triple-GAIL-GP	0.00	0.01	0.07	0.26	0.13	**0.53**
	sInfo-GAIL-GP	0.00	0.00	0.05	0.31	0.16	0.48

Table 7.7: The mean posterior probabilities of the cruise phase as calculated by the selector (Triple-GAIL-GP) and the posterior (sInfo-GAIL-GP) networks in the proposed MMIL scheme

		Policy 1	Policy 2	Policy 3	Policy 4
Mode 1	Triple-GAIL-GP	**0.70**	0.00	0.19	0.11
	sInfo-GAIL-GP	0.62	0.01	0.23	0.14
Mode 2	Triple-GAIL-GP	0.00	**0.62**	0.17	0.21
	sInfo-GAIL-GP	0.00	0.54	0.21	0.25
Mode 3	Triple-GAIL-GP	0.18	0.17	**0.65**	0.00
	sInfo-GAIL-GP	0.21	0.23	0.55	0.01
Mode 4	Triple-GAIL-GP	0.22	0.20	0.00	**0.58**
	sInfo-GAIL-GP	0.20	0.26	0.01	0.53

Table 7.8: The mean posterior probabilities of the descent phase as calculated by the selector (Triple-GAIL-GP) and the posterior (sInfo-GAIL-GP) networks in the proposed MMIL scheme

It is worth mentioning that in certain cases, the posterior probabilities for each method are low. To explain this, let's consider the cases of Modes 4 and 6 in the

Figure 7.10: Cruise phase modes 4 and 6. The significant overlap between these modalities is apparent.

"cruise" phase (Table 7.7). As can be observed in Fig. 7.10, there is significant overlap between these two distinct modalities. As a result, the posterior models cannot accurately identify the correct modality in such cases. However, by considering the entire generated trajectory, we can ultimately identify the correct sub-modality.

Success Rate (%)	**climb**	**cruise**	**descent**
Triple-GAIL-GP	100	85	100
sInfo-GAIL-GP	100	80	97

Table 7.9: Success rate of methods in terms of reaching the goal states in every phase

Furthermore, the success rate of the generated sub-trajectories in reaching the goal states (i.e. the waypoints at the end of each phase) is measured in Table 7.9. In a compositional approach, this is a crucial aspect because the predicted trajectory from origin to destination is constructed by concatenating sub-trajectories. As mentioned earlier, the successful termination of a sub-trajectory occurs when a flight reaches a distance within a $10\ km$ radius from the goal state. Although both methods can generate high-quality sub-trajectories for the "climb" and "descent" phases, as shown in Table 7.9, the success rate for the cruise phase is diminished. This can be attributed to the extended duration of this phase and the accumulation of the prediction errors.

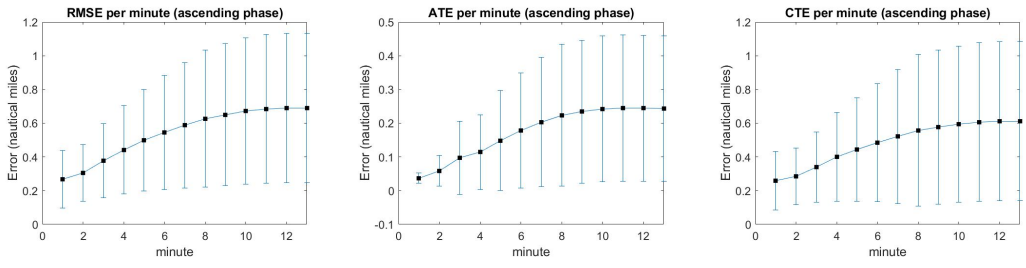
Nevertheless the results are still notable given the requirement for flights to terminate in a condensed area close to a waypoint at the end of a phase (as shown for instance in Fig. 7.7(c)).

Growth rates of prediction errors

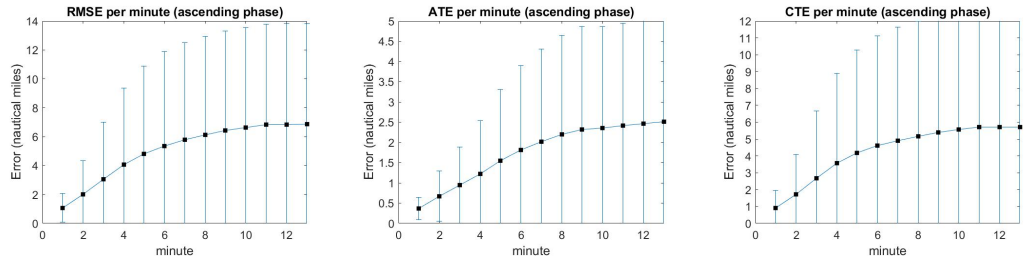
This section reports on the growth rate of the prediction error as the prediction horizon expands. It shows how the proposed modular scheme manages to report low growth rates of prediction errors for long prediction horizons, compared to the uni-module version. This is a crucial finding, since most of the prediction methods accumulate prediction errors for long-horizon tasks.

Specifically, the error growth rate measures the accumulation of the prediction error after every minute of look-ahead time. In accordance with the trajectory prediction standards set by EUROCONTROL [191], the lateral prediction error growth rate should be less or equal to $0.1\ nm/min$, while the longitudinal prediction error growth rate should be less or equal to $0.1\ nm/min$ for the cruise phase and $0.2\ nm/min$ for the other phases. Figures 7.11, 7.12 and 7.13 present the comparative growth rates of RMSE, ATE and CTE in terms of mean and standard deviation (std) of errors in nm (axis y) per minute of prediction (axis x) for every phase. It must be noted that these results concern the *MMIL* and *SMIL* policies learned by *Triple-GAIL-GP*. For a fair comparison, the *SMIL* policy which was trained for the whole trajectories, in this case is used for generating sub-trajectories separately for each of the phases identified, where the starting state is the initial point of the corresponding phase for the predicted trajectory.

As shown in Fig. 7.11, 7.12 and 7.13, *MMIL* manages to bound the error growth considerably in 3D, as measured by RMSE, as well for track errors ATE and CTE, in all phases compared to *SMIL*. This shows the capability of *MMIL* to bound compounding errors in long prediction horizons, with a smaller standard deviation compared to *SMIL*. This occurs also for the "cruise" phase, where the error reported is an order of magnitude higher compared to the other phases. Notably, error growth rates for the "cruise" phase are as follows: for RMSE(3D) $0.13\ nm/min$, for ATE $0.08\ nm/min$, and for CTE $0.08\ nm/min$. This indicates that this model can manage to bound the prediction errors even for long prediction horizons, close to the operational requirements for trajectory prediction. The corresponding growth rates for the "climb" and "descent" phases for RMSE(3D), ATE and CTE are even smaller for *MMIL*.

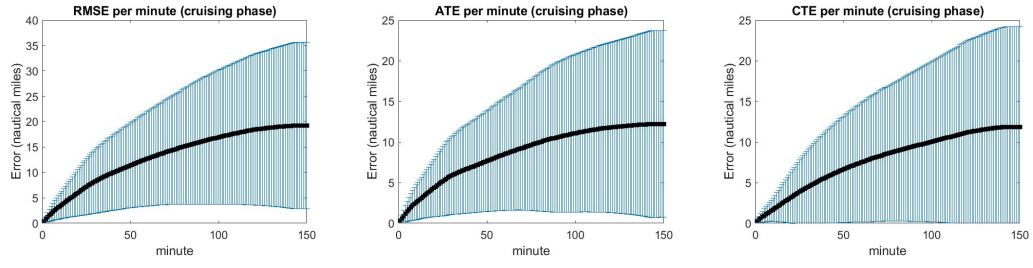


Mean RMSE Growth Rate=0.04 nm | Mean ATE Growth Rate=0.02 nm | Mean CTE Growth Rate=0.03 nm

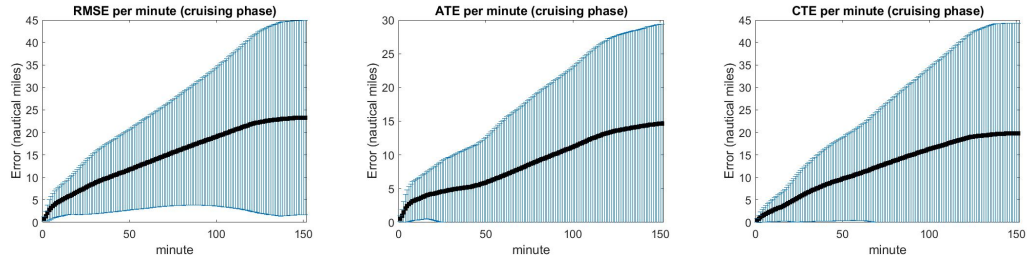


Mean RMSE Growth Rate=0.48 nm | Mean ATE Growth Rate=0.18 nm | Mean CTE Growth Rate=0.40 nm

Figure 7.11: RMSE, ATE & CTE growth rates (in nm) during the **climb phase** for MMIL and SMIL.



Mean RMSE Growth Rate=0.13 nm | Mean ATE Growth Rate=0.08 nm | Mean CTE Growth Rate=0.08 nm



Mean RMSE Growth Rate=0.15 nm | Mean ATE Growth Rate=0.09 nm | Mean CTE Growth Rate=0.13 nm

Figure 7.12: RMSE, ATE & CTE growth rates (in nm) during the **cruise phase** for MMIL and SMIL.

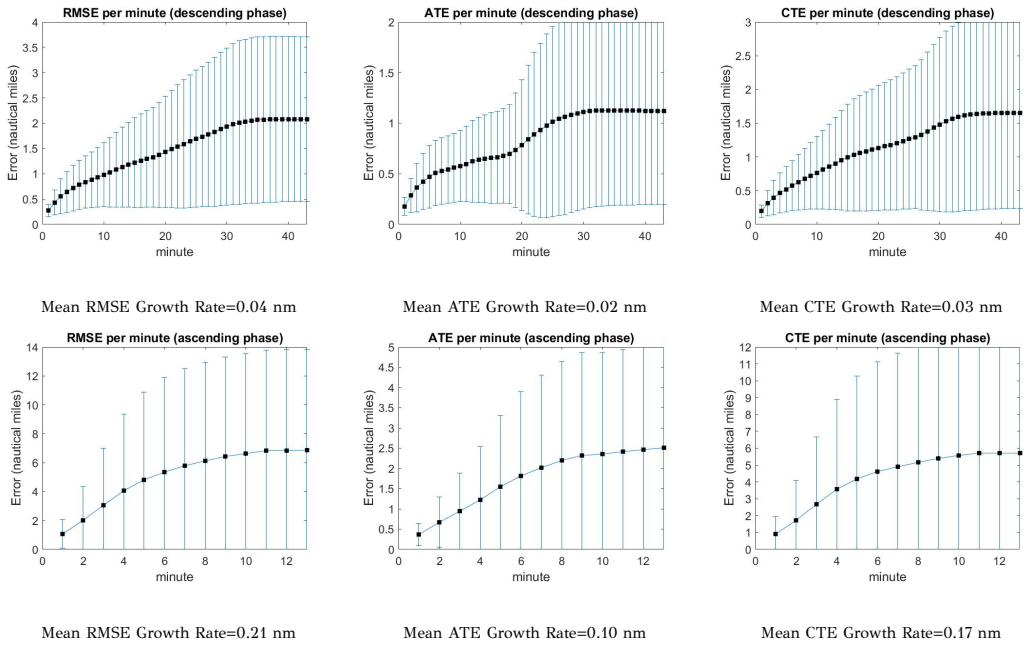


Figure 7.13: RMSE, ATE & CTE growth rates (in nm) during the **descent phase** for MMIL and SMIL.

7.6 Summary

Data-driven aircraft trajectory modeling constitutes an important research area due to the increasing need for efficient and safe air traffic management. This chapter aims to address this issue by proposing a modular, multi-modal IL approach for trajectory modeling, which leverages large-scale trajectory data and applies deep learning generative techniques to model the underlying behavior patterns and structures of aircraft trajectories. The modular approach allows the model to handle different phases of the flight, while the multi-modal nature enables the model to capture the diverse trajectory patterns exhibited by different flight modalities. By combining these two features, the generative agent can learn advantageous policies with the ability to generate high-quality trajectory predictions that are both accurate and diverse, while also providing realistic and robust solutions that can adapt to changing conditions.

Chapter 8

Conclusions and Future Study

This doctoral research is devoted to develop novel reinforcement learning agents in traffic applications, with emphasis on the urban road and the aviation domains. Several methods were proposed to (a) resolve congestion problems, and (b) modeling trajectories through imitating historical expert demonstrations, including techniques from deep reinforcement learning, hierarchical and modular learning, generative modeling, imitation learning and multi-agent systems. The depicted results by studying experimental, simulated and real traffic scenarios are compared and contrasted, demonstrating the remarkable applicability of the proposed methodologies. Through rigorous analysis and comprehensive evaluations, the findings not only validate the effectiveness of the methods but also provide invaluable insights into their practical implementation.

Traffic management and control is a very promising research area with the potential to impact not only various aspects of transportation, but also to become an attractive application area for studying and implementing machine learning approaches, learning schemes and mathematical models that will be able to offer complete and robust solutions. Due to the inherent complexity of the traffic domain, there is a multitude of exciting avenues for expansion and exploration. It offers the opportunity to study innovative strategies and solutions addressing challenges associated with traffic flow optimization, congestion mitigation, and sustainable transportation systems. By embracing these potential avenues of inquiry, the primary goal of this thesis is to develop innovative reinforcement learning methodologies and generative adversarial imitation learning modeling schemes for enhancing the efficiency and effectiveness of

traffic control mechanisms, such as adaptive traffic management, autonomous vehicle integration, dynamic route guidance, and intelligent transportation systems.

Chapters 3 and 4 discussed the problem of autonomously navigating a large number of vehicles in traffic networks with unsignalized intersections. The aforementioned problem was formulated as a MAMDP and tackled using deep MARL methods. Firstly, the problem was rigorously defined and the notion of *route-agents* was introduced. Then the development of rich state spaces and informative reward functions for the MAMDPs was shown. Moreover, the evaluation of the proposed methods was conducted on both artificial and real-world scenarios of various level of difficulty using the well-known traffic simulator SUMO. The results of these experiments have demonstrated the ability of the proposed methods to safely navigate a large number of vehicles inside large-scale traffic networks and minimize the traveling time for the participants of the multi-agent environment. Furthermore, a significant aspect of the work presented in Chapter 4 is the knowledge reuse (i.e., transfer learning) of the learned policies to unknown scenarios with much larger number of vehicles and increased stochasticity. The results obtained under the transfer learning settings were very promising which lead us to look into future directions for the problem. Such directions could be:

- Devise alternative reward function schemes that may contain more discriminating features.
- Extend the proposed method to handle larger traffic networks.
- Study advanced policy gradient RL approaches that allow working with continuous action spaces.
- Explore the problem under imitation learning by utilizing expert trajectories.
- Use the proposed methodology in different congestion problems for navigating swarms of entities in multi-agent environments.

Chapter 5 addresses the DCB problem in the ATM domain, where multiple flight-agents need to collaborate in order to develop optimal joint strategies that allocate the shared resources efficiently. This problem arises in ATM when multiple aircrafts want to pass through air sectors that have limited capacity. The aforementioned problem is initially formulated as a collaborative MAMDP and later is extended to the hierarchical

case. Two methods are proposed to solve the MDPs: Firstly, a collaborative MARL method that assigns delays to the flights at the pre-tactical stage of the operation. Then, a hierarchical MARL framework that supports state abstractions at various levels is proposed. The hierarchical framework offers refined solutions by initializing state-action values based on previous abstraction levels. In practice, the method works on two stages, where in the high-level stage an initial abstract solution is found, and on the second stage a refined (near) optimal solution is obtained. The effectiveness of the proposed method has been evaluated on real-world cases comprising historical flights above Spain. The results indicate that the hierarchical collaborative method has the ability to handle complex congestion settings with a large number of agents. The following directions have been set as future works:

- Explore deep reinforcement learning schemes that can handle continuous state spaces. By incorporating these approaches, the aim is to enhance the generalization capabilities of the methods.
- Use additional levels of abstraction to provide even more refined solutions.
- Study alternative reward function schemes. This will involve considering state-of-the-art reward schemes utilized in multi-agent congestion problems, which encompass additional features relevant to the DCB problem in ATM.
- Work on larger scenarios with more air sectors.
- Apply the proposed hierarchical methodology to different problems, where agents should coordinate actions in multiple levels of abstraction in order to find an optimal joint strategy.

Chapter 6 considers the problem of trajectory modeling in the aviation domain as an IRL problem, where a dataset of expert demonstrations (actual flights) acted as a teacher for the learning method. Under these settings, the proposed *apprenticeship learning* approach tried to extract the expert reward function that explains the behavior of the expert. To do so, a rich state space from RBF kernels was devised by considering spatial, temporal and meteorological features and a set of heading angles was used as actions for the aircraft to follow. In the end, DQN was trained to approximate the Q-values and choose among the actions. It is of utmost importance that the proposed method was tested on real-world expert data concerning flights from Barcelona to

Madrid, and the results were very competent to the actual flight trajectories. The focus is on the following points for future research:

- Use alternative deep neural networks structures as value function approximation schemes.
- Study other parametric model-based reward functions for imitating expert behavior.
- Use more complex continuous action spaces.
- Extend the experimental study with aircraft trajectories of longer distances.

Chapter 7 extends the flight trajectory modeling problem to cases where multiple expert behaviors (modalities) are present. Specifically, the suggested framework initially segments the expert trajectories into flight phases, and then proceeds by applying state-of-the-art multi-modal IL techniques for learning to mimic the multi-modal expert behaviors inside each phase. The objective functions of the proposed algorithms are enriched with regularization terms that enhance their performance and increase their generalization capabilities. The importance of the presented framework is demonstrated on the experimental results conducted on real-world flight trajectories from Paris to Istanbul that contained a plethora of ways (multi-modality) to reach the destination airport. In the future, further extensions on this research can be explored in the following directions:

- Use the proposed methodology to transfer the learned models between different origin-destination airport pairs, as well as to transfer models between similar patterns within the same phase, or among distinct phases.
- Develop an automated process to segregate trajectories in multiple sub-trajectories, also identifying even more detailed flight phases.
- Use offline reinforcement learning algorithms for imitating the expert trajectories.
- Generalize and evaluate the methodology for different types of trajectories in executing various tasks.

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] M. Gong, X. Niu, P. Zhang, and Z. Li, "Generative adversarial networks for change detection in multispectral imagery," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, pp. 2310–2314, 2017.

[3] Q. Wei, R. Song, and P. Yan, "Data-driven zero-sum neuro-optimal control for a class of continuous-time unknown nonlinear systems with disturbance using adp," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, pp. 444–458, 2016.

[4] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Journal of Symbolic Logic*, vol. 9, pp. 49–50, 1944.

[5] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[6] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2014.

[7] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, 1993, p. 3–10.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[10] J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 36, pp. 48–49, 1950.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training GANs," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, p. 214–223.

[13] C. Villani, *Optimal Transport: Old and New*. Springer, 2008, vol. 338.

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein GANs," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, p. 2180–2188.

[16] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," *ArXiv*, 2023.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[19] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[21] H. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems 23*, 2010, pp. 2613–2621.

[22] H. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[23] D. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, pp. 88–97, 1991.

[24] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 4950–4957.

[25] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[26] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd National Conference on Artificial Intelligence*, vol. 3, 2008, pp. 1433–1438.

[27] E. T. Jaynes, "Information theory and statistical mechanics," *Physical Review*, vol. 106, pp. 620–630, 1957.

[28] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, 2015, pp. 1889–1897.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, 2017.

[31] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, "Junior: The stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, p. 569–597, 2008.

[32] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, S. Kolski, A. Kelly, M. Likhachev, M. Mcnaughton, N. Miller, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, pp. 425–466, 2008.

[33] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knöppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, "Making Bertha drive—an autonomous journey on a historic route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, pp. 8–20, 2014.

[34] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[35] V. Talpaert, I. Sobh, B. R. Kiran, P. Mannion, S. K. Yogamani, A. E. Sallab, and P. Perez, "Exploring applications of deep reinforcement learning for real-world autonomous driving systems," *ArXiv*, 2019.

[36] X. Qian, S. Diemer, J. Gregoire, F. Moutarde, S. Bonnabel, I. Llatser, A. Festag, K. Sjöberg, A. de La Fortelle, A. Martinoli, and A. Marjovi, "Network of automated vehicles: The Autonet2030 vision," in *Proceedings of the 21st World Congress on the ITS World Congress*, 2014, pp. 2618–2626.

[37] 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, pp. 128–138, 2012.

[38] M. Wiering, "Multi-agent reinforcement learning for traffic light control," in *Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems*, 2000, pp. 1151–1158.

[39] M. Wiering, J. Vreeken, J. Veenen, and A. Koopman, "Simulation and optimization of traffic in a city," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2004, pp. 453 – 458.

[40] E. Camponogara and W. Kraus, "Distributed learning agents in urban traffic control," in *Proceedings of the 11th Portuguese Conference on Artificial Intelligence*, 2003, p. 324–335.

[41] M. Tubaishat, Y. Shang, and H. Shi, "Adaptive traffic light control with wireless sensor networks," *Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference*, pp. 187–191, 2007.

[42] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning based multi-agent system for network traffic signal control," *IET Intelligent Transportation Systems*, vol. 4, p. 128–135, 2010.

[43] T. Li, D. B. Zhao, and J. Q. Yi, "Adaptive dynamic programming for multi-intersections traffic signal intelligent control," in *Proceeding of the 11th International IEEE Conference on Intelligent Transportation Systems*, 2008, p. 286–291.

[44] A. Salkham, R. Cunningham, A. Garg, and V. Cahill, "A collaborative reinforcement learning approach to urban traffic control optimization," *International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 560–566, 2008.

[45] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, "Multiagent reinforcement learning for urban traffic control using coordination graph," in *Proceedings of the 19th Machine Learning and Knowledge Discovery in Databases*, 2008, p. 656–671.

[46] J. R. Kok and N. Vlassis, "Collaborative multiagent reinforcement learning by payoff propagation," *The Journal of Machine Learning Research*, vol. 7, pp. 1789–1828, 2006.

[47] E.-T. Samah, A. Baher, and A. Hossam, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): Methodology and large-scale application on downtown toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, pp. 1140 – 1150, 2013.

[48] H. Ge, Y. Song, C. Wu, J. Ren, and G. Tan, "Cooperative deep Q-learning with Q-value transfer for multi-intersection signal control," *IEEE Access*, vol. 7, pp. 40 797–40 809, 2019.

[49] S. P. Sahu, D. K. Dewangan, A. Agrawal, and T. Sai Priyanka, "Traffic light cycle control using deep reinforcement technique," in *Proceedings of the International Conference on Artificial Intelligence and Smart Systems*, 2021, pp. 697–702.

[50] K. Bálint, T. Tamás, and B. Tamás, "Deep reinforcement learning based approach for traffic signal control," *Transportation Research Procedia*, vol. 62, pp. 278–285, 2022.

[51] N. Kumar, S. Mittal, V. Garg, and N. Kumar, "Deep reinforcement learning-based traffic light scheduling framework for SDN-enabled smart transportation system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 2411–2421, 2022.

[52] A. Abhishek, P. Nayak, K. P. Hegde, A. Lakshmi Prasad, and K. S. Nagegowda, "Smart traffic light controller using deep reinforcement learning," in *Proceedings of the 3rd International Conference for Emerging Technology*, 2022, pp. 1–5.

[53] I. Zohdy and H. Rakha, "Optimizing driverless vehicles at intersections," in *Proceedings of the 19th ITS World Congress*, 2012.

[54] J. Chen, Z. Wang, and M. Tomizuka, "Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors," in *Proceeding of the IEEE Intelligent Vehicles Symposium*, 2018, pp. 1239–1244.

[55] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," *ArXiv*, 2018.

[56] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proceeding of the 4th International Conference on Learning Representations*, 2016.

[57] A. R. Fayjie, S. Hossain, D. Oualid, and D. Lee, "Driverless car: Autonomous driving using deep reinforcement learning in urban environment," in *Proceeding of the International Conference on Ubiquitous Robots*, 2018, pp. 896–901.

[58] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 2070–2075, 2018.

[59] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, vol. 48, 2016, p. 1928–1937.

[60] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep Q-learning," in *Proceedings of the 21st International Conference on Intelligent Transportation Systems*, 2018, p. 3169–3174.

[61] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Härtl, F. Dürr, and J. M. Zöllner, "Learning how to drive in a real world simulation with deep Q-networks," in *Proceeding of the IEEE Intelligent Vehicles Symposium*, 2017.

[62] Z. Qiao, K. Muelling, J. Dolan, P. Palanisamy, and P. Mudalige, "POMDP and hierarchical options MDP with continuous actions for autonomous driving at intersections," in *Proceeding of the IEEE International Conference on Intelligent Transportation Systems*, 2018, pp. 2377–2382.

[63] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *Proceeding of the IEEE International Conference on Robotics and Automation*, 2018, pp. 2034–2039.

[64] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," in *Proceeding of the Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, p. 359–367.

[65] M. Zhou, Y. Yu, and X. Qu, "Development of an efficient driving strategy for connected and automated vehicles at signalized intersections: A reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, pp. 433–443, 2020.

[66] Z. Cao, H. Guo, J. Zhang, U. Fastenrath, and F. Oliehoek, "Multiagent-based route guidance for increasing the chance of arrival on time," in *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, 2016, p. 3814–3820.

[67] Z. Cao, H. Guo, J. Zhang, F. Oliehoek, and U. Fastenrath, "Maximizing the probability of arriving on time: A practical Q-learning method," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[68] N. O. Alsrehin, A. F. Klaib, and A. A. Magableh, "Intelligent transportation and control systems using data mining and machine learning techniques: A comprehensive study," *IEEE Access*, vol. 7, pp. 49 830–49 857, 2019.

[69] G. Rodrigues de Campos, P. Falcone, R. Hult, H. Wymeersch, and J. Sjöberg, "Traffic coordination at road intersections: Autonomous decision-making algorithms using model-based heuristics," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, pp. 8–21, 2017.

[70] National Traffic Highway Safety Association, "Overview of motor vehicle traffic crashes in 2021," 2021.

[71] K. Zhang, D. Zhang, A. de La Fortelle, X. Wu, and G. J., "State-driven priority scheduling mechanisms for driverless vehicles approaching intersections," *IEEE Transactions of Intelligent Transportation Systems*, vol. 16, pp. 2487–2500, 2015.

[72] E. Choi, "Crash factors in intersection-related crashes: An on-scene perspective," *US DOT National Highway Traffic Safety Administration*, 2010.

[73] M. Bouton, A. Cosgun, and M. J. Kochenderfer, "Belief state planning for autonomously navigating urban intersections," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 825–830.

[74] X. Huang, S. Hong, A. Hofmann, and B. C. Williams, "Online risk-bounded motion planning for autonomous vehicles in dynamic environments," in *Proceedings of the 29th International Conference on Automated Planning and Scheduling*, 2019, pp. 214–222.

[75] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, 2008.

[76] J. Lee and B. Park, "Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment," *IEEE Intelligent Transportation Systems Magazine*, vol. 13, pp. 81–90, 2012.

[77] R. Kala, *On-Road Intelligent Vehicles: Motion Planning for Intelligent Transportation Systems*. Butterworth-Heinemann, 2016.

[78] G. Taylor and R. Parr, "Kernelized value function approximation for reinforcement learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 1017–1024.

[79] F. L. Da Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," *Journal of Artificial Intelligence Research*, vol. 64, pp. 645–703, 2019.

[80] F. L. Da Silva, M. E. Taylor, and A. H. Reali Costa, "Autonomously reusing knowledge in multiagent reinforcement learning," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018, pp. 5487–5493.

[81] M. J. Kochenderfer, *Decision making under uncertainty: Theory and application*. MIT press, 2015.

[82] M. Treiber and A. Kesting, *Traffic Flow Dynamics: Data, Models and Simulation*. Springer, 2013.

[83] R. W. Rosenthal, "A class of games processing pure-strategy nash equilibria," *International Journal of Game Theory*, vol. 2, pp. 65–67, 1973.

[84] I. Milchtaich, "Social optimality and cooperation in nonatomic congestion games," *Journal of Economic Theory*, vol. 114, pp. 56–87, 2004.

[85] C. Meyers, "Network flow problems and congestion games: complexity and approximation results," Ph.D. dissertation, MIT, 2006.

[86] M. Penn, M. Polukarov, and M. Tennenholtz, "Congestion games with failures," *Discrete Applied Mathematics*, vol. 159, pp. 1508–1525, 2011.

[87] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, "Optimization problems in congestion control," in *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, 2000, pp. 66–74.

[88] A. L. C. Bazzan, J. Wahle, and F. Klügl, "Agents in traffic modelling - from reactive to social behaviour," *Proceedings of the 23rd Annual German Conference on Artificial Intelligence*, pp. 303–306, 1999.

[89] T. Kravaris, G. Vouros, C. Spatharis, K. Blekas, and G. Chalkiadakis, "Learning policies for resolving demand-capacity imbalances during pre-tactical air traffic management," *Multiagent System Technologies*, pp. 238–255, 2017.

[90] K. Tumer, Z. Welch, and A. Agogino, "Aligning social welfare and agent preferences to alleviate traffic congestion," in *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2008, pp. 655–662.

[91] M. Colby and K. Tumer, "Multiagent reinforcement learnng in a distributed sensor network with indirect feedback," in *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2013, pp. 941–948.

[92] S. Delvin, L. Yliniemi, D. Kudenko, and K. Tumer, "Potential-based difference rewars for multiagent reinforcement leraning," in *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2014, pp. 165–172.

[93] R. Radulescu, P. Vrancx, and A. Nowe, "Analysing congestion problems in multi-agent reinforcement learning," in *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2017, pp. 1705–1707.

[94] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, 1998, p. 1043–1049.

[95] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.

[96] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 5, 1992.

[97] T. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, 2000.

[98] A. Bai, S. Srivastava, and S. Russell, "Markovian state and action abstractions for MDPs via hierarchical MCTS," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016, p. 3029–3037.

[99] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, p. 1726–1734.

[100] M. Riemer, M. Liu, and G. Tesauro, "Learning abstract options," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, p. 10445–10455.

[101] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 3682–3690.

[102] A. Ma, M. Ouimet, and J. Cortés, "Hierarchical reinforcement learning via dynamic subspace search for multi-agent planning," *Autonomous Robots*, 2019.

[103] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *ArXiv*, 2017.

[104] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, p. 1553–1561.

[105] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, p. 3307–3317.

[106] C. Spatharis, T. Kravaris, G. A. Vouros, K. Blekas, G. Chalkiadakis, J. M. C. Garcia, and E. C. Fernandez, "Multiagent reinforcement learning methods to resolve demand capacity balance problems," in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, 2018.

[107] A. J. Cook and G. Tanner, "European airline delay cost reference values," 2015. [Online]. Available: http://www.eurocontrol.int/publications/european-airline-delaycost-reference-values

[108] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, 2002, pp. 227–234.

[109] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored MDPs," in *Advances in Neural Information Processing Systems*, vol. 14, 2001.

[110] F. Giunchiglia and T. Walsh, "A theory of abstraction," *Artificial Intelligence*, vol. 57, pp. 323–389, 1992.

[111] D. F. Rogers, R. D. Plante, R. T. Wong, and J. R. Evans, "Aggregation and disaggregation techniques and methodology in optimization," *Operations Research*, vol. 39, pp. 553–582, 1991.

[112] C. Spatharis, A. Bastas, T. Kravaris, K. Blekas, G. A. Vouros, and J. M. Cordero, "Hierarchical multiagent reinforcement learning schemes for air traffic management," *Neural Computing and Applications*, vol. 35, pp. 147–159, 2021.

[113] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, pp. 469–483, 2009.

[114] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the 11th Annual Conference on Computational Learning Theory*, 1998, p. 101–103.

[115] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the 17th International Conference on Machine Learning*, 2000, p. 663–670.

[116] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd International Conference on Machine Learning*, 2006, p. 729–736.

[117] D. Silver, J. Bagnell, and A. Stentz, "High performance outdoor navigation from overhead data using imitation learning," in *Robotics: Science and Systems*, 2008.

[118] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, pp. 25–53, 2009.

[119] B. D. Ziebart, J. A. D. Bagnell, and A. Dey, "Modeling interaction via the principle of maximum causal entropy," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 1255 – 1262.

[120] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *ArXiv*, 2015.

[121] N. Aghasadeghi and T. Bretl, "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 1561–1566.

[122] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, vol. 15, 2011, pp. 182–189.

[123] N. Ratliff, J. A. Bagnell, and S. S. Srinivasa, "Imitation learning for locomotion and manipulation," in *Proceedings of the 7th IEEE-RAS International Conference on Humanoid Robots*, 2007, pp. 392–397.

[124] U. Syed and R. E. Schapire, "A reduction from apprenticeship learning to classification," in *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, vol. 2, 2010, p. 2253–2261.

[125] F. S. Melo and M. Lopes, "Learning from demonstration using MDP induced metrics," in *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*. Springer-Verlag, 2010, p. 385–401.

[126] B. Piot, M. Geist, and O. Pietquin, "Boosted and reward-regularized classification for apprenticeship learning," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, 2014, p. 1249–1256.

[127] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, pp. 286–298, 2007.

[128] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *IEEE Robotics & Automation Magazine*, vol. 17, pp. 44–54, 2010.

[129] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse reinforcement learning through structured classification," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[130] E. Klein, B. Piot, M. Geist, and O. Pietquin, "A cascaded supervised learning approach to inverse reinforcement learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013, pp. 1–16.

[131] M. Fahad, Z. Chen, and Y. Guo, "Learning how pedestrians navigate: Adeep inverse reinforcement learning approach," in *Proceedings of the International Conference on Intelligent Robots and Systems*, 2018, pp. 819–826.

[132] P. Abbeel, A. Coates, M. Quigley, and A. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information Processing Systems*, 2007.

[133] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang, "Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning," in *Proceeding of the IEEE Intelligent Vehicles Symposium*, 2018, pp. 1251–1256.

[134] M. Guo, Z. Zhong, W. Wu, D. Lin, and J. Yan, "Irlas: Inverse reinforcement learning for architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9013–9022.

[135] B. Ziebart, Anind, K. Dey, and J. Bagnell, "Probabilistic pointing target prediction via inverse optimal control," in *Proceedings of the International Conference on Intelligent User Interfaces*, 2012, pp. 1–10.

[136] O. M. Messer and P. Ranchod, "The use of apprenticeship learning via inverse reinforcement learning for generating melodies," in *Proceedings of the International Conference on Mathematics and Computing*, 2014, pp. 1793–1799.

[137] L. Viet, L. Siyuan;, and L. Chuin, "A reinforcement learning framework for trajectory prediction under uncertainty and budget constraint," in *Proceedings of the 22nd European Conference on Artificial Intelligence*, 2016, pp. 347–354.

[138] R. Wang, C. Cilibert, P. Amadori, and Y. Demiris, "Random expert distillation: Imitation learning via expert policy support estimation," in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 6536–6544.

[139] K. Shiarlis, J. Messias, and S. Whiteson, "Inverse reinforcement learning from failure," in *Proceedings of the 16th International Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 1060–1068.

[140] P. Englert, N. Vien, and M. Toussaint, "Inverse KKT: Learning cost functions of manipulation tasks from demonstrations," *The International Journal of Robotics Research*, vol. 36, 2017.

[141] S. Ross, G. Gordon, and J. Bagnell, "No-regret reductions for imitation learning and structured prediction," *ArXiv*, 2010.

[142] A. Kuefler and M. J. Kochenderfer, "Burn-in demonstrations for multi-modal imitation learning," *ArXiv*, 2017.

[143] Z. Wang, J. Merel, S. Reed, G. Wayne, N. de Freitas, and N. Heess, "Robust imitation of diverse behaviors," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, p. 5326–5335.

[144] N. Deo and M. Trivedi, "Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms," in *Proceedings of the Intelligent Vehicles Symposium*, 2018.

[145] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "Precog: Prediction conditioned on goals in visual multi-agent settings," in *Proceedings of the International Conference on Computer Vision*, 2019.

[146] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially acceptable trajectories with generative adversarial networks," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2018.

[147] V. Kosaraju, A. Sadeghian, R. Martin-Martin, I. Reid, H. Rezatofighi, and S. Savarese, "Social-BiGAT: Multimodal trajectory forecasting using bicycle-GAN and graph attention networks," in *Proceedings of the Neural Information Processing Systems*, 2019.

[148] J. Merel, Y. Tassa, T. Dhruva, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. M. O. Heess, "Learning human behaviors from motion capture by adversarial imitation," *ArXiv*, 2017.

[149] J. Lin and Z. Zhang, "ACGAIL: Imitation learning about multiple intentions with auxiliary classifier GANs," in *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, 2018.

[150] K. Kobayashi, T. Horii, R. Iwaki, Y. Nagai, and M. Asada, "Situated GAIL: Multitask imitation using task-conditioned adversarial inverse reinforcement learning," *ArXiv*, 2019.

[151] S. Piao, Y. Huang, and H. Liu, "Online multi-modal imitation learning via lifelong intention encoding," in *Proceedings of the 4th International Conference on Advanced Robotics and Mechatronics*, 2019, pp. 786–792.

[152] Y. Qiu, J. Wu, Z. Cao, and M. Long, "Out-of-dynamics imitation learning from multimodal demonstrations," *ArXiv*, 2022.

[153] F.-I. Hsiao, J.-H. Kuo, and M. Sun, "Learning a multi-modal policy via imitating demonstrations with mixed behaviors," *ArXiv*, 2019.

[154] J.-W. Peng, M.-C. Hu, and T.-W. Chu, "An imitation learning framework for generating multi-modal trajectories from unstructured demonstrations," *Neurocomputing*, vol. 500, 2022.

[155] H. Cheng, W. Liao, X. Tang, M. Yang, M. Sester, and B. Rosenhahn, "Exploring dynamic context for multi-path trajectory prediction," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2021, pp. 12 795–12 801.

[156] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, "Multi-modal motion prediction with stacked transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7573–7582.

[157] T. Phan-Minh, E. Grigore, F. Boulton, O. Beijbom, and E. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 074–14 083.

[158] Y. Li, J. Song, and S. Ermon, "Inferring the latent structure of human decision-making from raw visual inputs," *ArXiv*, 2017.

[159] C. Fei, B. Wang, Y. Zhuang, Z. Zhang, J. Hao, H. Zhang, X. Ji, and W. Liu, "Triple-GAIL: A multi-modal imitation learning framework with generative adversarial nets," in *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, 07 2020, pp. 2901–2907.

[160] C. Simpkins and C. Isbell, "Composable modular reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 4975–4982.

[161] J. Xue and A. Frédéric, "Multi-task learning with modular reinforcement learning," in *Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior*, 2022.

[162] K. Narendra, J. Balakrishnan, and M. Ciliz, "Adaptation and learning using multiple models, switching, and tuning," *IEEE Control Systems magazine*, vol. 15, p. 37–51, 1995.

[163] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 3540–3549.

[164] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-optimal representation learning for hierarchical reinforcement learning," in *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[165] A. Levy, G. D. Konidaris, R. P. Jr., and K. Saenko, "Learning multi-level hierarchies with hindsight," in *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[166] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017, pp. 2169–2176.

[167] R. Yang, H. Xu, Y. WU, and X. Wang, "Multi-task reinforcement learning with soft modularization," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 4767–4777.

[168] S. Mittal, A. Lamb, A. Goyal, V. Voleti, M. Shanahan, G. Lajoie, M. Mozer, and Y. Bengio, "Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules," in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 6972–6986.

[169] A. Goyal, A. Lamp, J. Hoffmann, S. Sodhani, S. Levine, and B. Scholkopf, "Recurrent independent mechanisms," in *Proceedings of the International Conference on Learning Representations*, 2021.

[170] J. Mendez, H. Seijen, and E. Eaton, "Modular lifelong reinforcement learning via neural composition," in *Proceedings of the 10th International Conference on Learning Representations*, 2022.

[171] D. Esteban, L. Rozo, and D. Caldwell, "Hierarchical reinforcement learning for concurrent discovery of compound and composable policies," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 18 180–1825.

[172] J. López-Leonés, M. A. Vilaplana, E. Gallo, F. A. Navarro, and C. Querejeta, "The aircraft intent description language: A key enabler for air-ground synchronization in trajectory-based operations," in *Proceedings of the IEEE/AIAA 26th Digital Avionics Systems Conference*, 2007.

[173] W. Kun and P. Wei, "A 4-D trajectory prediction model based on radar data," in *Proceedings of the 27th Chinese Control Conference*, 2008, pp. 591–594.

[174] M. G. Hamed, D. Gianazza, M. Serrurier, and N. Durand, "Statistical prediction of aircraft trajectory : regression methods vs point-mass model," in *USA/Europe Air Traffic Management Research and Development Seminar*, 2013, pp. 1–10.

[175] Y. Le Fablec and J. Alliot, "Using neural networks to predict aircraft trajectories," in *Proceedings of the International Conference on Artificial Intelligence*, 1999, pp. 524–529.

[176] T. Cheng, D. Cui, and P. Cheng, "Data mining for air traffic flow forecasting: a hybrid model of neural network and statistical analysis," *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, vol. 1, pp. 211–215, 2003.

[177] Y. Liu and M. Hansen, "Predicting aircraft trajectories: A deep generative convolutional recurrent neural networks approach," 2018.

[178] A. de Leege, M. van Paassen, and M. Mulder, *A Machine Learning Approach to Trajectory Prediction*. Springer, 2013.

[179] K. Tastambekov, S. Puechmorel, D. Delahaye, and C. Rabut, "Aircraft trajectory forecasting using local functional regression in Sobolev space," *Transportation Research part C: Emerging Technologies*, vol. 39, pp. 1–22, 2014.

[180] S. Hong and K. Lee, "Trajectory prediction for vectored area navigation arrivals," *Journal of Aerospace Information Systems*, vol. 12, pp. 490–502, 2015.

[181] S. Ayhan and H. Samet, "Aircraft trajectory prediction made easy with predictive analytics," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, p. 21–30.

[182] H. Georgiou, S. Karagiorgou, Y. Kontoulis, N. Pelekis, P. Petrou, D. Scarlatti, and Y. Theodoridis, "Moving objects analytics: Survey on future location & trajectory prediction methods," 2018.

[183] C. Spatharis, K. Blekas, and G. A. Vouros, "Apprenticeship learning of flight trajectories prediction with inverse reinforcement learning," in *Proceedings of the 11th Hellenic Conference on Artificial Intelligence*, 2020, pp. 241–249.

[184] P. N. Tran, H. Q. V. Nguyen, D.-T. Pham, and S. Alam, "Aircraft trajectory prediction with enriched intent using encoder-decoder architecture," *IEEE Access*, vol. 10, pp. 17 881–17 896, 2022.

[185] A. Bastas, T. Kravaris, and G. A. Vouros, "Data driven aircraft trajectory prediction with deep imitation learning," *ArXiv*, 2020.

[186] T. Kravaris, A. Bastas, and G. A. Vouros, "Predicting aircraft trajectories via imitation learning," in *Adaptive and Learning Agents Workshop Workshop, International Conference on Autonomous Agents and Multiagent Systems*, 2021.

[187] K. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[188] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1994, p. 359–370.

[189] P. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, p. 53–65, 1987.

[190] C. Gong and D. McNally, *A Methodology for Automated Trajectory Prediction Analysis*. Springer, 2004.

[191] S. Morton, P. Dias, J. Garnier, and B. Redeborn, "EUROCONTROL specification for trajectory prediction," EUROCONTROL-SPEC, Tech. Rep., 2010.

Author's Publications

Journal papers

(*J*1) **Christos Spatharis**, Alevizos Bastas, Theocharis Kravaris, Konstantinos Blekas, George A. Vouros, Jose Manuel Cordero. *Hierarchical multiagent Reinforcement Learning schemes for air traffic management*. In Neural Computing and Applications, 2021

(*J*2) **Christos Spatharis**, and Konstantinos Blekas. *Multiagent Reinforcement Learning for autonomous driving in traffic zones with unsignaled intersections*. Journal of Intelligent Transportation Systems: Technology, Planning, and Operations, 2022

(*J*3) Piyabhum Chaysri, **Christos Spatharis**, Konstantinos Blekas, and Kostas Vlachos. *Unmanned Surface Vehicle Navigation through Generative Adversarial Imitation Learning*. Ocean Engineering, 2023 (accepted)

(*J*4) **Christos Spatharis**, Konstantinos Blekas, and George A. Vouros. *Modelling Flight Trajectories with Multi-modal Generative Adversarial Imitation Learning*. Submitted to scientific journal (under review)

(*J*5) **Christos Spatharis**, Konstantinos Blekas, and George A. Vouros. *Modular and Multi-modal Generative Adversarial Imitation Learning for Modelling Flight Trajectories*. Submitted to scientific journal (under review)

Conference papers

($C1$) Theocharis Kravaris, George A. Vouros, **Christos Spatharis**, Konstantinos Blekas, Georgios Chalkiadakis, and Jose Manuel Cordero Garcia. *Learning Policies for Resolving Demand-Capacity Imbalances During Pre-Tactical Air Traffic Management*. In 15th German Conference Multiagent System Technologies (MATES), 2017

($C2$) Theocharis Kravaris, George A. Vouros, **Christos Spatharis**, Konstantinos Blekas, Georgios Chalkiadakis, and Jose Manuel Cordero Garcia. *Multiagent Reinforcement Learning Methods for Resolving Demand - Capacity Imbalances*. In IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), 2018

($C3$) **Christos Spatharis**, Theocharis Kravaris, George A. Vouros, Konstantinos Blekas, Georgios Chalkiadakis, Jose Manuel Cordero Garcia, Esther Calvo Fernandez. *Multiagent Reinforcement Learning Methods to Resolve Demand Capacity Balance Problems*. In 10th Hellenic Conference on Artificial Intelligence (SETN), 2018

($C4$) **Christos Spatharis**, Konstantinos Blekas, Alevizos Bastas, Theocharis Kravaris, George A. Vouros. *Collaborative multiagent Reinforcement Learning schemes for air traffic management*. In 10th International Conference on Information, Intelligence, Systems and Applications (IISA), 2019

($C5$) **Christos Spatharis**, Konstantinos Blekas and George A. Vouros. *Apprenticeship learning of flight trajectories prediction with inverse Reinforcement Learning*. In 11th Hellenic Conference on Artificial Intelligence (SETN), 2020

($C6$) **Christos Spatharis**, and Konstantinos Blekas. *Double Deep Multiagent Reinforcement Learning for Autonomous Driving in Traffic Maps with Road Segments and Unsignaled Intersections*. In 23rd IEEE International Conference on Intelligent Transportation Systems (ITSC), 2020

Short Biography

Christos Spatharis acquired his B.Sc. and M.Sc. degrees in Computer Science from the Department of Computer Science and Engineering, University of Ioannina, Greece in 2016 and 2018, respectively. He is currently a PhD Student at the same department. During his academic career, he worked as a research associate in several European projects including "DART: Data Driven Aircraft Trajectory Prediction Research" (2017-2018), "Data-Driven Trajectory Imitation with Reinforcement Learning" (2019-2020) and "SIMBAD: Combining Simulation Models and Big Data Analytics for ATM Performance Analysis" (2021 - 2022). Moreover, he participated in the publication of nine research papers of which he presented four at SETN(2018), IISA(2019), SETN(2020) and ITSC(2020). His main research interests lie on the field of reinforcement learning, deep learning, imitation learning, pattern recognition, and multi-agent systems.