SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dottorato di Ricerca in Automatica, Bioingegneria e Ricerca Operativa
(ABRO)

Thesis For The Degree Of Doctor Of Philosophy

Control and Data-driven Operational Strategies for Industry 4.0 and Sustainable City Logistic

Thesis Advisor
Prof. Francesco Delli Priscoli

Candidate
Muhammad Imran
1893943

Academic Year 2019-2022 (XXXV cycle)

Abstract

Digital transformation and automation affect every sector of society and industry and can significantly increase productivity by assisting decision-makers in complex environments. It is no surprise that the *Internet of Things (IoT)* and learning-based techniques have been an enormous boon and key enabler in addressing the complex, data-driven, and time-taking industrial and societal challenges. The cornerstone of these transformations is solid mathematical foundations, control and learning-based new system design techniques to cope with this fast-moving field. The present thesis focuses on developing a control and data-driven framework for time-critical operations where an asset (i.e., equipment, buildings, plants, and machinery) performance directly impacts business outputs, such as manufacturing, energy, services, and transportation sectors. The developed framework can be deployed in cross-disciplinary sectors to address the user-defined *Key Performance Indicators (KPIs)*. We evaluated it against the most significant *KPIs* from manufacturing and transportation sectors by focusing on two specific use cases, given by the control of an assembly line and a traffic intersection. Relevant KPIs involved are linked with the control goals of minimizing the assembly line resource utilization and optimally controlling the traffic signals to minimize the vehicles' waiting time and balance the queues. Human safety is the central point of interest for real-time deployment in the proposed *KPIs*. Considering the sensitivity, firstly, the problem is addressed from a control perspective by using the *Model Predictive Control (MPC)* technique, which requires developing a comprehensive mathematical formulation of the assembly line control problem and the traffic signal control problem.

The *MPC* controller is in charge of interpreting the state of the plant and traffic signal to obtain the desired control inputs to optimize the performance of the system, according to the selected *Key Performance Indicator (KPI)*s. Secondly, to overcome the limitation regarding scalability, prediction horizon and extensive computation time that the *MPC* technique brings in large scenarios, a *Deep Reinforcement Learning (DRL)*-based control framework was developed to address both use cases. In this respect, one of the critical challenges for *DRL*-based control is ensuring constraint satisfaction (e.g., precedence constraints among the tasks, constraints on needed resources to run tasks, deadlines, signals, etc.). The proposed *DRL* method allows us to consider all the constraints explicitly. Once training is complete, it can be used in real-time to dynamically control the assembly line and the traffic intersection. Another motivation for the proposed work is the method's ability to work in complex scenarios and the presence of uncertainties. The use of deep neural networks allows for learning the model of the environment, in contrast with, e.g., optimization-based techniques, which require explicitly writing all the equations of the model of the problem. We adopt a learning scheme to speed up the training phase in which more agents are trained in parallel. Simulations show that the proposed method can provide adequate real-time decision support to industrial operators for scheduling and rescheduling activities, achieving the goal of minimizing the total execution time. To prove the capabilities of the developed algorithm, several simulation scenarios are proposed through the variation in the number of launch vehicles to be assembled and the presence of disturbances affecting the plant. The results are also validated through a microscopic and continuous multi-modal traffic simulator *Simulation of Urban Mobility (SUMO)*for the traffic control use case. The work is carried out in the context of the EU-funded project Smart European Space Access thru Modern Exploitation of Data Science (SESAME).

Acknowledgements

I sincerely thank my supervisor, **Prof. Francesco Delli Priscoli**, and co-supervisor **Dr. Francesco Liberati** for allowing me to work in a stimulating environment and on exciting topics. They provide constant support to engage and compete with the best European academic and industrial communities and enriched me as a person, researcher and, in general, a problem solver. A special thanks go to **Dr.Andrea Tortorelli**, with whom I had the chance to collaborate during the last part of my Ph.D. His exceptional research and project manager skills allowed me to acquire a broader panoramic and new knowledge. The results would not have been obtained without the interaction with and help from the CRAT Team. The environment in the laboratory has always been very stimulating and full of people willing to help and share their know-how. In particular, I would like to thank Martina for being a reference in many technical, everyday and administrative issues. Another thank goes to SESAME Consortia, where I had the chance to conduct some of the research activities presented in this thesis. Of course, a sincere thank goes to my **family and friends**.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ALBP Assembly Line Balancing Problems

ALWABP Assignment and Balancing Problem

BIM Building Information Modelling

CAGR Compound Annual Grow Rate

CPS Cyber-physical Sytsem

CWT Cumulative Waiting Time

DQN Deep Q-Network

DRL Deep Reinforcement Learning

DTSE Discrete Traffic State Encoding

FIFO First in First Out

GALBP Generalized Assembly Line Balancing Problems

IDS Industrial Data Space

IIOT Industrial Internet of Things

IoT Internet of Things

JIT Just-in-Time

JSSP Job Shop Scheduling Problem

KPI Key Performance Indicator

KPIs Key Performance Indicators

LE Line Efficiency

LT Line Time

MDP Mark of decision Process

MILP Multi Integer Linear Programming

ML Machine Learning

MMAL Mixed Model Assembly Line

MMALBP Mixed-Model lines Assembly Line Balancing Problems

MOPSO Multi-Objective Particle Swarm Optimization

MPC Model Predictive Control

MPS Minimum Part Set

NASA National Aeronautics and Space Administration

NN Neural Network

NSGA-II Non-dominated Sorting Genetic Algorithm

OR Operational Research

PALBPS Resource Constrained Assembly Line Balancing Problem

PPO Proximal Policy Optimization

PTALBP Programming model for the Two-sided Assembly Line Balancing Problem

RCALBP Resource Constrained Assembly Line Balancing Problem

RL Reinforcement Learning

SA Simulated Annealing

SALBP SingleAssembly Line Balancing Problems

SI Smoothness Index

SPT Shortest Processing Time

STL Static Traffic Light

SUMO Simulation of Urban Mobility

TD Temporal Difference

TL Traffic Light

TSC Traffic Signal Control

UALBP U-shape lines Assembly Line Balancing Problems

Chapter 1

Introduction

This research focuses on developing Control and data-driven strategies to achieve operational excellence in the industrial environment and city logistics management. The underlying technological developments include: (i) deployment and management of pervasive (wireless) sensor networks, (ii) creation of virtualized environments for simulations, (iii) development of augmented reality applications, (iv) use of big data, (v) full integration of factories with the ICT continuum [18]. These elements are expected to increase productivity significantly, flexibility, quality of the manufactured products and total revenues [68]. *Artificial Intelligence (AI)* is a key enabler of core features, such as synchronized planning and scheduling in smart factories and city ecosystems.

Technological development can significantly increase productivity by assisting the decision-makers in highly complex, data-driven, heterogeneous, expensive, and time-taking processes. Digital technologies offer new solutions to companies with dynamic and resilient tools to address societal and industrial challenges in line with the united nation sustainable goals. The research activities are carried out in the context of the SESAME project. The main objective of the project is to provide the European Launcher Industry with a common framework to produce the data in the form of a platform with attributes such as digital, open, intelligent, secure and standard, encouraging human integration in the processes, allowing engineers and operators in the development, production, testing and operations of the launchers while implementing proactive risk management.

Moreover, SESAME aims to reduce costs through production improvements and operational effectiveness. Ariane Group is the project's coordinator, with a total assigned budget of almost €3.5 million. There are a total of seven participants from four countries. From France, National Centre for Space Studies (CNES), Predict, Capgemini Technology Servicies; from Italy Consorzio per la Ricerca nell'Automatica e nelle Telecomunicazioni (CRAT) and VItrociset; from Romania Scoala Nationala de Studii Politice si Administrative (SNSPA) and finally from Spain Fundacio Eurecat.

The future city management systems, including traffic lights, are challenging problems within the transport system. The most innovative and cheapest solution to address this issue is to improve the performance of the traffic light system, a low-cost intervention enabling the preservation of existing infrastructures. It urgently needs to address this issue as the last century is witnessing the world continually becoming more dependent on road transportation for conveying persons and goods. At the same time, authorities need to fulfil the United Nations Sustainable goals by reducing the enormous growth of CO_2 emissions and providing efficient services to the citizen.

1.1 Motivation

To compete in today's global market, business activities are fostered to reshape and optimize their business model to meet sustainability goals. This applies particularly to manufacturing industries, which are complex systems characterized by many operational constraints and vital efficiency requirements. Over the years, a manufacturing process, such as designing efficient assembly lines, received considerable attention from small and medium enterprises and the academic world, mainly through innovations and technologies to provide high-end quality services or products at the least cost. Connectivity and intelligent communication between machines, humans and assembly components will increase productivity and quality dramatically and elevate mass customization to new levels. Various industry and research reports indicate tremendous potential in transforming industrial production.

By 2020, European industrial companies will invest EUR 140 billion annually in Industry 4.0 applications, and more than 80% companies will have digitized their value chain. The aim of the present work will therefore be to highlight the effectiveness of these artificial intelligence techniques to provide valuable decision support to human controllers [73]. Digitalised manufacturing will result in many changes to manufacturing processes, outcomes and business models[21]. Smart factories allow increased **flexibility** in production; using configurable robots means that various products can be produced. The **speed** with which a product can be produced will also improve. Productivity improvements can be gained from information sharing in real-time, and resources can be utilized more efficiently by dynamic allocation. Productivity improvements can be gained from information sharing in real-time, and resources can be utilized more efficiently by dynamic allocation. The product quality will improve from highly controlled production processes with automatic monitoring and quality assurance. In a recent report from Tractica, annual worldwide manufacturing sector investment in AI software, hardware, and services will increase from 2.9 billion in 2018 to 13.2 billion by 2025 [12].

In this context, the explosion of the Industry 4.0 paradigm highlighted these issues. In the next few years (2020 – 2025), *Compound Annual Grow Rate (CAGR)* generated by Industry 4.0 paradigm is expected to be more than 20% [30]. Considering an extended prediction horizon (2020– 2027), the CAGR is expected to be even more significant at almost 25% [45]. Several studies [37], [17] [71] already highlighted how, in the context of the COVID-19 pandemic and potentially of other events of such nature, Industry 4.0 represents a powerful tool helping in the minimization of the diffusion and impact of the virus. Hence, the trend mentioned above in such a market is expected to be confirmed in the next decade.

The transportation sector is primarily responsible for carbon emissions; Figure 1.1 highlights its increment in the last 15 years, Figure 1.2 summarizes the Another critical drawback of traffic congestion is the massive loss of money due to the time spent by each driver in line Furthermore, in Figure 1.3 The pandemic period also highlighted how much traffic influences the air quality since it was seen, particularly for the most polluted cities.

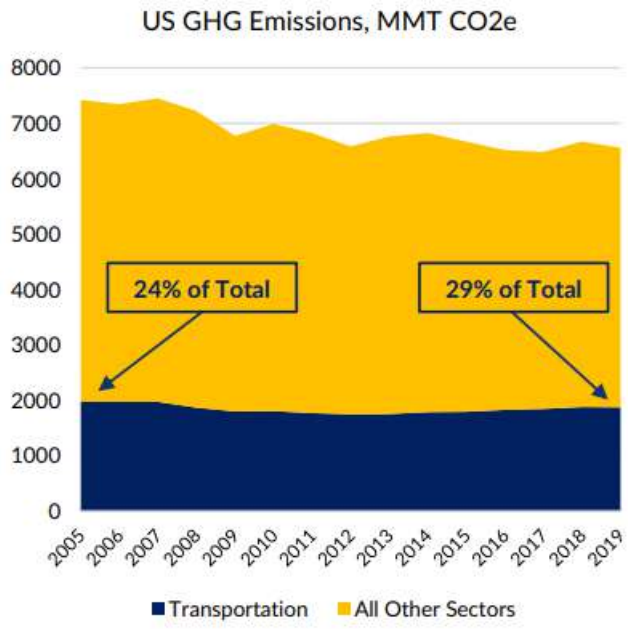

Figure 1.1: The transportation sector's share of emissions has increased over the last 15 years.

2021 U.S. Rank (2020)	Urban Area	Delay 2021 (2020) [2019]	Compared to Pre-COVID	Cost per Driver	Cost per City	Downtown Trips
1 (1)	New York	102 (100) | [140]	-27%	$1,595	$8.3B	-18%
2 (3)	Chicago	104 (86) | [145]	-28%	$1,622	$5.8B	-21%
3 (2)	Philadelphia	90 (94) | [142]	-37%	$1,404	$3.3B	-22%
4 (4)	Boston	78 (48) | [149]	-47%	$1,223	$2.3B	-23%
5 (9)	Miami	66 (35) | [81]	-19%	$1,028	$2.6B	-20%
6 (5)	Los Angeles	62 (45) | [103]	-40%	$968	$5.2B	-28%
7 (6)	San Francisco	64 (47) | [97]	-34%	$1,001	$1.6B	-49%
8 (8)	Houston	58 (35) | [81]	-29%	$897	$2.6B	-25%
9 (7)	New Orleans	63 (42) | [79]	-21%	$977	$501M	-28%
10 (22)	Atlanta	53 (20) | [82]	-36%	$820	$2B	-16%

Figure 1.2: Money loss due to traffic delay.

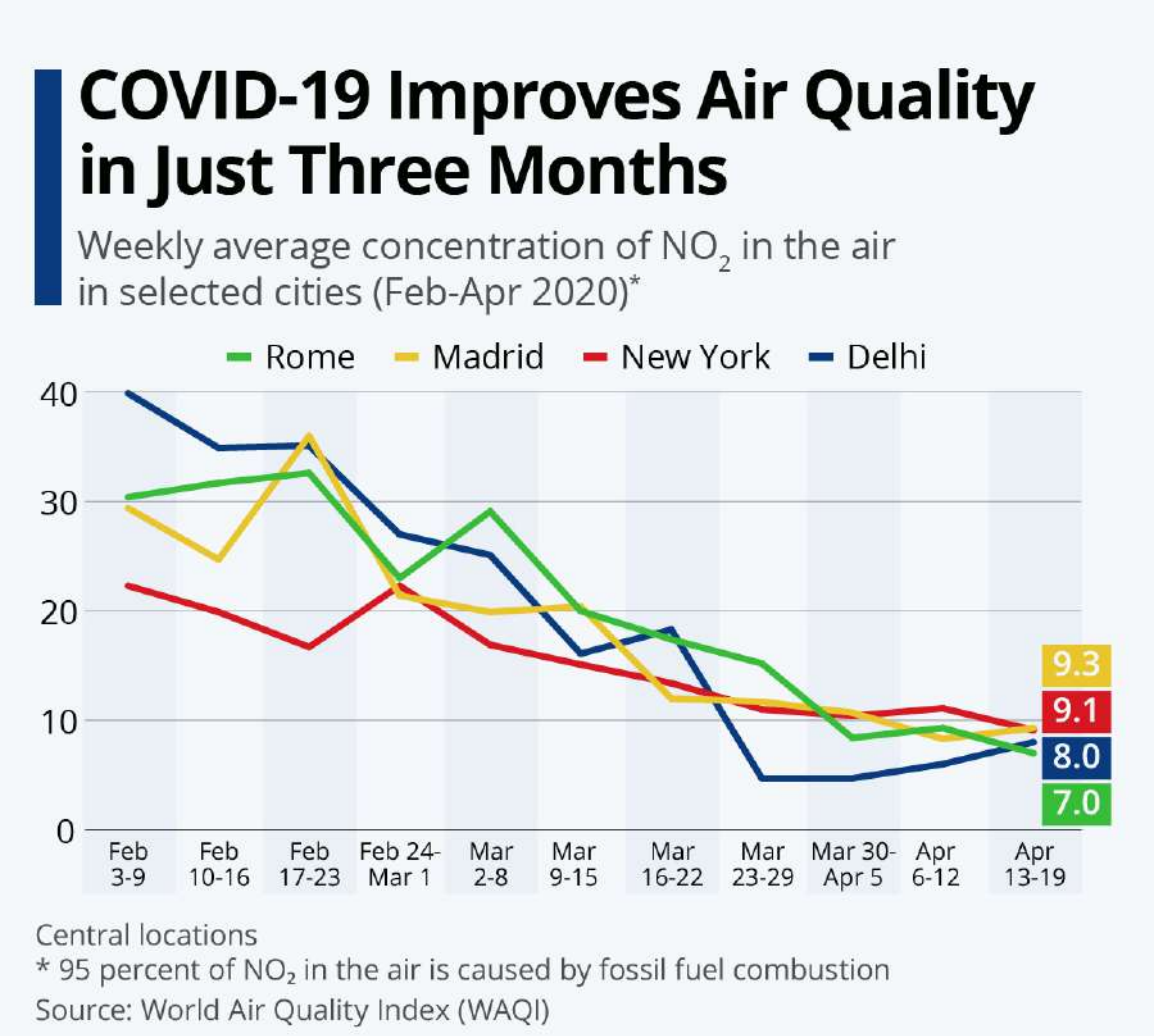

Figure 1.3: Improvement of the air quality during the pandemic period. https://inrix.com/

The INRIX Signals Scorecard estimated that 7-10% of total trip time is delayed at signals, meaning upwards of 25% of delay is spent sitting at traffic lights. Therefore, improving signal timing is crucial for reducing the environmental impact of transportation. In particular, it was seen that enhancing traffic flow during peak periods significantly affects air quality. The city of Atlanta was token as an example, and it was shown that more than 10,000 tons of CO_2 were reduced due to decreased delay. A valuable tool to support industries is certainly machine learning. The applications of Artificial Intelligence are used mainly in complex decision-making situations, and most predictive maintenance systems rely on this technique to formulate predictions. The advantages are numerous and can significantly reduce costs and carbon emissions and consistently maintain the high quality of city life and industrial services.

1.2 Objective, Contribution and Outlines of Thesis

Objective

The main objective of this thesis is to develop new tools for time-critical industrial and city logistic management systems. The nature of these systems is complex and dynamic; specifically, we address the following two problems:

- Operations management of industrial assembly-lines
- Traffic junctions management in city logistics

These two problems can be mapped into direct objectives involving specific methodological and technological development, indirect objectives for business efficiency, and environmental effects, as described below.

1. Operation management:
    - **Direct objective:** Developing control and data-driven strategies to optimally control the factory floor activities.
    - **Indirect objective:** Energy consumption, Business process modelling and Return on assets.
2. Traffic junctions management:
    - **Direct objective:** Developing control and data-driven strategies to manage traffic signals optimally.
    - **Indirect objective:** Environmental effects, carbon emission and economic benefits.

We formulated both problems regarding Control and Learning-based techniques to achieve the defined objectives. Especially in *Model Predictive Control (MPC)* and *Deep Reinforcement Learning (DRL)* perspective because *MPC* approaches are mature and already used by industries, in industrial processes independently, but the nature of industrial systems are drastically changing with the usage of *Industrial Internet of Things (IIOT)*. So, it is a natural way to gain confidence from the industrial community and investors by providing alternative solutions that are economical, optimal and trust-able. We need new tools, techniques, and talent to meet industrial and societal challenges. Ultimately, thats matter for business, society, startup and the individual.

Contributions

1. Our contributions can be listed as follows:
    - A comprehensive mathematical formulation of industrial assembly lines considering every aspect is presented in *MPC* way and results are published in [42].
    - we also point out the proposed methodology is not scale-able and needs to be more scalable and adaptable. Still, the control-based approach is computationally expensive and incompatible with energy consumption.

- We proposed a *DRL* approach to overcome these limitations and developed a scaleable DRL framework [76] for industrial assembly lines to reduce the overall cycle time, which **DeepMind** recently cited in [15]. It is scalable and in line with the dynamic nature of the industrial production systems to overcome the domain-dependent heuristics extensively used in the manufacturing industry.

2.
   - We further addressed city logistics management problems, including traffic lights, a unit of the overall city management system. The developed framework reduces the vehicle waiting time at junctions via optimal traffic signal scheduling.
   - We attempt to model the problem in terms of *MPC* and *DRL*; a quadratic mixed-integer nature and a data-driven problem are formulated and solved, respectively. Simulation is performed in a real-time scenario, and we had promising results from *DRL*-based techniques compared to *MPC* and results are submitted for publication.

The remainder of this thesis is organized as follows.

1. Chapter 2:A Comprehensive State of the art of Industrial assembly lines, Traffic signals control and preliminaries of proposed techniques Model predictive Control and Deep Reinforcement Learning-based techniques are presented.

2. Chapter 3: We present the mathematical modelling of the proposed framework, which is the backbone of the system design, as follows:

   - In the First half, a detailed mathematical formulation of Industrial assembly line use cases in terms of Model Predictive Control and Deep Reinforcement Learning is presented.
   - The second half presents a detailed mathematical formulation of Traffic signal control use cases in terms of Model Predictive Control and Deep Reinforcement Learning.

3. Chapter 4: We tested the developed framework against simple to highly complex scenarios. A comprehensive comparison of proposed techniques and existing techniques is presented.

4. Chapter 5: We conclude both use cases and promising future research directions, especially those related to the role of learning and control techniques in industrial, societal and business impact in future industries.

1.3 Publications

1. Published:

   - F. Liberati, A. Tortorelli, C. Mazquiaran, **M. Imran** and M. Panfili, "Optimal Control of Industrial Assembly Lines," 2020 7th International Conference on Control, Decision and Information Technologies (CoDIT), Prague, Czech Republic, 2020, pp. 721-726, doi: 10.1109/CoDIT49905.2020.9263946.
   - A. Tortorelli , **M. Imran**, F. Delli Priscoli, F. Liberati . "A Parallel Deep Reinforcement Learning Framework for Controlling Industrial Assembly Lines". Electronics. 2022; 11(4):539. https://doi.org/10.3390/electronics11040539

- **M. Imran**, F. Liberati . "A Framework for Intelligent Decision Making in Networks of Heterogeneous Systems (UAV and Ground Robots) for Civil Applications" Engineering Proceedings. 2022; 17(1):13. https://doi.org/10.3390/engproc2022017013

2. Submitted:

   - **M. Imran**, R. Izzo A. Tortorelli,F. Liberati, "Traffic Control with Model Predictive Control and Deep Reinforcement Learning: a Comparison," 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT), Rome, Italy, 2023, **Submitted**
   - **M. Imran**, G.Antonucci, Di Giorgio, D.Priscoli, A. Tortorelli, F. Liberati, "Task Scheduling in Assembly Lines with Single-Agent Deep Reinforcement Learning." 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT), Rome, Italy, 2023, **Submitted**

Chapter 2

State of the art

This chapter investigates the state of the art of *Assembly Line Balancing Problems (ALBP)* and *Traffic Signal Control (TSC)* from a control and machine learning perspective. Firstly we present a comprehensive literature review of *ALBP* lines and *TSC* problems. Secondly, preliminaries related to *MPC* and *DRL* techniques are presented. A study on the *ALBP* is done to make logistics more efficient and improve agility and assembly capabilities. This section will present an overview of some of the most important aspects we should consider when an assembly line in a factory has to be mathematically modelled and optimized. This comes from the fact that we will abstract the entire Kourou complex as a single factory, where each building will be considered a single workstation.

2.1 Industrial Assembly Lines from Control Perspective

Assembly line methods were initially introduced to increase factory productivity and efficiency generically, defined in 2.1.1.

Definition 2.1.1. An assembly line is a set of workstations performing different operations (tasks) on items moving on the line according to a maximum or average time, called the cycle time.

The most critical issues concerning the development of an assembly line deal with the arrangement of the tasks to be performed (line scheduling) and the designing of production lines, which involves selecting the necessary operations to machine a part [33]. During the forty years of the assembly line's existence, only trial and error methods were used to balance the lines, subsequently providing mathematical attempts to solve problems as linear programs ming. Ultimately, heuristic methods have become the most popular technique for solving problems. So, the optimal planning and controlling of the execution of tasks in an assembly line result in a well-known problem in the literature in which many formulations have been proposed.

2.1.1 Classification based on types

One of the best-known problems in the manufacturing process is the *ALBP*. It deals with the optimal allocation of tasks among workstations so that a given objective function is optimized, satisfying the constraints of the assembly process and requirements on demand. Two leading families of *ALBP* are **type 1** aims to minimise the number of workstations without exceeding a given integer cycle time while satisfying the precedence constraints among tasks. In [7], two prominent families of

ALBP problems were identified:*SingleAssembly Line Balancing Problems (SALBP)* and *Generalized Assembly Line Balancing Problems (GALBP)*. The difference between them is that the production of a unique element characterizes the *SALBP*, a fixed expected cycle time, a one-sided serial line, precedence constraints, and deterministic times among other constraints [10]. Visual representations of assembly lines are shown in 2.1.

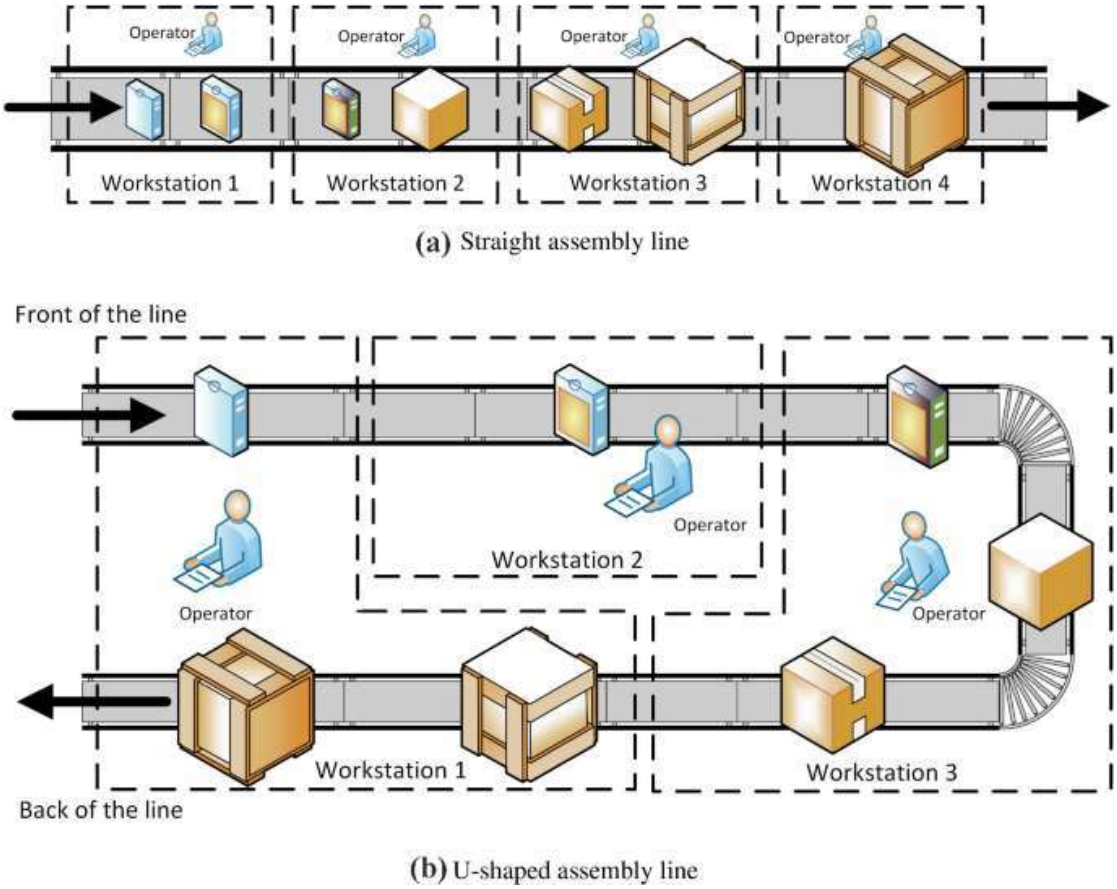

Figure 2.1: Assembly lines type based on factory layout

Among all the different kinds of problems we will expose later, we can differentiate them as Type-1 or Type-2. Type-1 means that the primary goal of the problem is to optimize the number of workstations for a given fixed cycle time. On the other hand, Type-2 refers to the fact that the number of workstations is fixed, and we have to optimize the cycle time. However, there have been other types proposed for specific cases. One of them is Type-E problems [22], whose objective function is the efficiency of the assembly line.

In contrast, Type-F problems aim to find a feasible balance between the number of workstations and cycle time, which are both fixed [8] [32]. The above standard problem types can be further particularized to specific applications by customizing the mathematical formulation of the optimization problem. So once the distinction on the Type has been made, we can introduce our problem as an *Mixed-Model lines Assembly Line Balancing Problems (MMALBP)* Type-2.

Definition 2.1.2. An *ALBP* of Type-2 is the main optimization objective to minimise the cycle time.

In today's competitive environment, assembly lines are sometimes insufficient to meet customer demands and expectations. It is necessary to increase the flexibility of the line to respond to

customer needs. One strategy involves the time required to perform a setup activity, called setup time. An academic study in [53] proposes a new line design concept called *Resource Constrained Assembly Line Balancing Problem (PALBPS)* with line-switching setups, which include travelling between two neighbouring lines and the setup activities for the workpiece/model on the other line. The same/different product models can be produced on each line with the exact/different cycle times. A large number of methods for solving this problem have been studied. In [67], the aim was to minimize the number of workstations needed and to optimize the workload balance among stations to reduce any possible overload effects of the bottleneck workstation by proposing the shortest route algorithm. At the same time, the *GALBP* formulation relaxes some *SALBP* assumptions for addressing more realistic scenarios. In particular, *GALBP* allows considering multi-model processes, zoning constraints, delays, parallel stations and more complex layouts. In *GALBP* problems, we can introduce the commonly known *U-shape lines Assembly Line Balancing Problems (UALBP)* and *MMALBP* as in 2.1.

The authors of [25] discuss some heuristics based on genetic algorithms, tabu search and annealing techniques, which almost lead to the reduction of the number of workstations (single parallel lines or u-line configuration) and productivity improvement. They put critical attention on the *KPI*s performance, such as *Line Efficiency (LE)*, *Line Time (LT)*, and *Smoothness Index (SI)*. The same measures of solution quality were presented in [11], where has been considered the dual problem *ALBP* of type 2, which aims to minimise the cycle time, considering the number of workstations, is given. Similarly,[39] shares the common purpose of improving the assembly line at the point of flow and, in addition, trying to eliminate the imbalance phenomenon by focusing on work efficiency. They use the graphical and symbolic representation of the activities flow process chart) and want to raise efficiency, reduce cost and increase product quality by improving the workers' skills in the bottleneck processes and machine supply equipment. Traditionally, *ALBP* configure one operator in each workstation, and then the so-called Assembly Line Worker *Assignment and Balancing Problem (ALWABP)* arise; it is another variant of the *ALBP*, solved by the ranch-and-bound technique.

Several real-world extensions of*ALBP* have been introduced in many cases but solved with straightforward heuristics. In [70], bidirectional branch-and-bound procedure (SALOME) can solve even large *ALBP*, extended with different assignment restrictions, mostly linked tasks or fixed stations, in a very effective manner. Thanks to the assembly line, production periods were shortened, equipment costs accelerated, and labour and management endeavoured to keep up with the changes. The following Table highlights some variants of the *ALBP* and the most adopted algorithms in the literature to derive an optimal assembly line solution.

2.1.2 Classification based on Mathematical Modeling

Assembly line balancing problem where one aims to integrate workers with very different task execution times in the line. At the same time, they also take on a second objective regarding cycle time, intending to maintain productivity levels. By assigning more than one operator at each workstation (multi-manned), multiple work units can be completed simultaneously, so we do not limit the productivity of the production line. This commonly occurs in industries with large product sizes. The study in [14] proposes a mixed integer programming model for the *Resource Constrained Assembly Line Balancing Problem (RCALBP)* (referred to as machines or tools) where in turn, each operator can perform various tasks on the identical product according to task precedence. A binary

linear mathematical programming model and a *Simulated Annealing (SA)* algorithm are proposed to model and solve the *PALBPS* very effectively and successfully. The last consideration leads to classification based on single, multi or mixed model production on lines.

The first example of the mixed model assembly line is presented in [87]. The authors tackle the problem of assembling two or more products of similar structures that are processed simultaneously with the optimization objective of the least number of workstations and the highest assembly line balancing rate. An ant colony optimization algorithm is designed to solve such a mixed-model assembly line balancing problem. However, *ALBP* the is often a simplification that bears little resemblance to a real-life situation. Focusing on the entire manufacturing system of the assembly line, we can assume that it is a real-time system that involves many tasks and many operation conditions. In [42], a deterministic formulation of the problem in real-time of the execution of a given set of tasks in a given assembly line is introduced. The objective is to present an approach for increasing the flexibility of industrial assembly lines through an optimization framework. The control and planning are done to improve the performance and to capture all the leading relations and constraints characterising resources (workers, tools, components, vehicles, etc.) and tasks. Another real-time control structure has been designed and implemented in [50], allowing automated control strategy for the complete manufacturing cycle of an assembly/disassembly mechatronics line; to model such systems, the author's used tools such as Petri Nets.

Therefore, the *ALBP* consists of the optimal assignment of tasks to workstations while always satisfying given constraints and requirements. Several formulations of the *ALBP* have been proposed. In [75], the author provides the book with a broad overview of assembly line planning and control. Starting from the basic concepts such as cycle time, precedence diagram and line balance, among others, it explains the theory of Assembly Planning and Inventory Requirements for several types of balancing problems. Furthermore, examples of techniques such as Make-to-Order and Make-to-Stock are provided in the following chapters. The difference is based on the fact that the former is deeply affected by the customer specifications. At the same time, the latter is more general, producing items more standardly and exporting them to customers under request. In [5], we can understand the process of developing a mixed integer *Programming model for the Two-sided Assembly Line Balancing Problem (PTALBP)*. Moreover, some extensions are presented, such as a cost-oriented model with time and space constraints and assignment restrictions. This kind of problem can also be found in the paper introduced by Özcan, Gökçen, and Toklu [54]. As they state, with this kind of problem, there are several different two-sided lines in parallel. Each line has tasks that can be performed on the left, right or either side.

The main aim of the problem is to obtain the optimal allocation for all the tasks to stations while optimizing specific performance criteria such that the precedence relations between tasks and the total task time in each station are not more significant than the cycle time. From now on, one of the main aspects to distinguish and understand from the beginning is the Type of *ALBP*.

In [61], the authors solve an *MMALBP* addressing the line balancing simultaneously and sequencing problems, comparing the performances of two algorithms:*Multi-Objective Particle Swarm Optimization (MOPSO)* and the *Non-dominated Sorting Genetic Algorithm (NSGA-II)*; results show that the latter has better performances. As we said before, with the function (3.14), we would fulfil the system's primary objective, but there can be others. For example, Razali [63] introduces an interesting function that aims to reduce the resources used to get an economical profit as much

as possible. This may be achieved by taking into account all the workstations $WK = \{1, ..., wk\}$ and resources $R = \{1, ...r\}$ to be used:

$$min \sum_{wk=1}^{WK} \sum_{r=1}^{R} R_{rs} \tag{2.1}$$

Where max R is the maximum available resources at the station W and R_{rs} is a binary variable equal to 1 if the resource r is used at workstation W, 0 otherwise.

Other examples of minimization of available resources are found in [32], where the authors address a SALBP-1 problem with resources constraints, while [33] studies a *RCALBP* proposes a simple, efficient heuristic based on the widely-used ranked positional weight (RPW) rule. Rabbani [59] introduces a new concept: the following. As in some stations, the operators will perform similar tasks between the different parts of the launchers and payloads; we would like to minimize the setup times between those different models. This can be achieved, according to the authors, using the following function:

$$min \sum_{n=1}^{N} \sum_{md=1}^{MD} \sum_{md_0=1}^{MD} \sum_{q=1}^{Q} Q^n_{md,md^\wp}, q^t_{md,md^\wp,n} \tag{2.2}$$

Where md and $md^\wp$ are the two different models to take into account and $Q^n_{md,md^\wp,q}$ is a binary decision variable; if models md and $md^\wp$ are sequentially timed at workstation wk at sequence q, the number of sequences, being q the number of sequences $Q = \{1, ..., q\}$, 0 otherwise. In the same way, to obtain an advantage by avoiding too many setup times,[63] propose an extra objective function where the demand of each model is taken into account:

$$min \sum_{k=1}^{Dt} \sum_{md=1}^{MD} \sum_{md=1}^{MD} (x_{md,k} - k \times \frac{d_k}{Dt})^2 \tag{2.3}$$

Where d_k is the demand of model k, $dk2$, Dt and $x\ mk$ is the total quantity of product/produced over stages 1 to k, $k = \{1, 2, ..., D_t\}$.

These objective functions are of capital importance, but we must not forget that they must be subject to constraints to optimise the results. As we are talking about an *ALBP* of type 2, we can recall that it means that the number of workstations is fixed, and we can change the cycle time c, the first constraint to come up with ensures that at each station and for each model, the time of the tasks is less or equal to c:

$$\sum_{n=1}^{TK} tk_n x_{md,n,wk} <= c \tag{2.4}$$

Where $x_{md,n,wk}$ is a binary decision variable if task tk_n of model md is assigned to workstation W_k, 0 otherwise.

This can be seen in its different forms in [48], [60], [56] [63] and[5] among others, as it is one of the main constraints. This function was straightforward as it defines the problem itself, but now we will deal with some other constraints that are not so obvious. The first one is the mathematical expression to ensure that all stations are assigned at least one task, while the second one will make that each task is assigned to a maximum of one station, as done by [56], [4] [87], among others:

$$\sum_{n=1}^{TK} x_{md,n,wk} >= 1 \forall wk \in WK, md \in MD \tag{2.5}$$

$$\sum_{wk=1}^{WK} x_{md,n,wk} >= 1 \forall tk \in TK, md \in MD \tag{2.6}$$

Continuing with the constraints on the tasks, we need some expression that forces the tasks in the workstations to follow a determined order for each model. This is done through the following expression, or a similar one, in, [86], [49], [60], [56] [57] [63] among others.

$$\sum_{wk=1}^{WK} x_{md,n,wk} - \sum_{wk} WK x_{md,b,wk} <= 0 \forall (a,b) \in Prev_{md,tk} \tag{2.7}$$

Where $Prev_{md,tk}$ is the set of previous task tasks tk in the model md.

In [60], the goal is to minimize the total utility work, workload smoothness, and setup cost. To do it, the author addresses the *Mixed Model Assembly Line (MMAL)* in a cyclic production strategy represented by a *Minimum Part Set (MPS)* that determines the sequence of models, which is the most important factor because there is considerable setup time between two consecutive batches.

In this paper, both problems, balancing and sequencing problems, are considered simultaneously. Some introduced constraints are to calculate the utility work, improve the operator's starting position after finishing each model and the ending position of the operator, and a way to express the sequencing of the models. The following equation, for example, enforces that in each sequence, exactly one model must be completed.

$$\sum_{wk=1}^{WK} \sum_{md}^{MD} Y^{s}_{md,1,q} = 1 \forall q \in Q \tag{2.8}$$

In this equation, $Y^{WK}_{md,1,q}$ is a binary variable that takes a value equal to 1 if the task $tk = 1$, the last one of model md at sequence q is used in the station wk. According to [60], in the case of having several models produced in sequences, another important constraint is related to the demand, as we have to fulfil each model one. For this, the author proposes the expression. This is shown in equation (2.9), which is an expression similar to equation 2.7 with a slight modification to fulfil the desired requested items.

$$\sum_{wk=1}^{WK} \sum_{q}^{Q} Y^{s}_{md,1,q} = 1 \forall md \in MD \tag{2.9}$$

Moreira [47] uses another relevant idea when considering the workers. It is clear that most authors consider that every worker can perform any task, but in real life, that may not be entirely true as specific certifications are needed to perform some jobs. In this case, it is talking about the fact of disabled workers that cannot perform a specific task. In any case, the workers are assigned tasks they can perform, and constraints ensure that the cycle time is respected at stations without and with disabled workers, respectively.

According to another paper written by the same author, [48], the study on the case of workers with abilities to perform specific tasks is continued, but in this case, making extra use of Miltenburg's

regularity criterion. This criterion introduces the product rate variation problem (known in the literature as PRV). This is a concept that appears in *Just-in-Time (JIT)* production systems, particularly in mixed-model assembly lines. In this kind of environment, the aim is to produce only the required quantities, so it is essential to maintain the usage rate of the line. Therefore, the goal of a regular distribution of products is relevant to them.

An exciting idea is applied in [86], where a term to reduce the workload variation in each workstation is introduced. With this idea, the authors try to reduce idle time in workstations. This is because having machinery and workers not performing tasks come with an economical cost. In this way, the workload is further reorganized. This is done using the term shown in **equation missing**. As we can see there, the authors try to minimize, for each model, the proportional to the demand difference in the workload at each workstation. This means that the objective is that the utilization of each workstation is as equal as possible.

$$min \frac{d_{md}}{Dt} \sqrt{\frac{1}{WK} \sum_{wk=1}^{WK} (ut_{wk,mt} - utav_{md}) 2 \forall md \in MD} \tag{2.10}$$

Related to the fact that there is a presence of several similar models in Mixed-model lines, we have to try to obtain a single equivalent graph, as stated in [8]. Still, inefficiencies in the result may appear. This comes from the fact that when we are reducing from a multi-model $ALBP$ To an equivalent single-model $ALBP$, the same task may have different task times attached. Accordingly, the completion time is different, and we must obtain a single value. This has an impact on work overload and idle times. So in the mentioned paper, the author presents different objective functions obtained from the literature, but we will focus on one.

$$min \sum_{wk=1}^{WK} \sum_{md}^{MD} |T_{wk,md-T_{md}}| \tag{2.11}$$

Equation (2.11) minimizes the sum of absolute differences between the average station time T_{md} for model md and the real station time $T_{wk,md}$.

In [42], an attempt is made to obtain an optimal control for a general assembly line of type-2 cases. The paper discusses this problem while introducing an optimal control formulation to improve assembly line performance regarding cycle time minimization, resource utilization, etc. A deterministic formulation of the problem is introduced based on mixed-integer linear programming. One of the critical aspects introduced here for the SESAME project is that not only the tasks and workstations are taken into account, but also the resources. What is more, something quite innovative is that tasks need a determined resource, and some resources are unique; their location must be known beforehand to perform the best possible optimization.

2.1.3 Preliminaries of Model Predictive Control in terms of Supply Chain

According to [69], the basic idea of *MPC* is that "at each discrete time instance, the control action is obtained by solving online a finite-horizon open-loop optimal control problem". For the previous reason, in the framework of this project, we will provide an overview of literature related o how *MPC* dynamically manages production to meet the expected demand of the customer. As this is a control-oriented strategy, it has the advantage that it can be tuned to provide the best performance

possible in the presence of uncertainties, errors and constraints in inventories and production levels.

There are three main elements in an *MPC* system:

- Internal dynamic model of the process, as we stated before.
- History of the past control actions
- Cost function J to be optimized over the prediction horizon.

At this point, a clarification should be done, as *MPC* is not a pure, specific control strategy but a set of control methods obtained by minimizing a specific objective function, which makes use of a dynamic model of the system to be controlled to obtain the control signal [13], The essence of an *MPC* is to optimize forecasts of process behaviour [62]. If the model is not rich enough, feedback may help to solve its deficiencies, [58]. The process to be controlled can be described in a discrete-time, deterministic linear state space model, as we will use in our case, in fact

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_x$$

In that space state model representation, as usual, uk represents the vector of inputs, and y describes the process outputs; x is the vector of states of the system. The *MPC* strategy can be summarized in several steps as follows:

- Future outputs are determined in a Prediction Horizon (ph) at each discrete time instant k using the process model. These outputs are denoted as $y_{(k+k|k)}$, where $k = \{1, ..., ph\}$ and they depend on the knowledge of past inputs, outputs until time t, and on future control inputs $u_{(k+k|k)}, k = \{0, ..., ch\}$ where ch is the Control Horizon, which are the ones to be calculated.
- The set of future control signals is calculated by optimizing a given criterion (objective function) to keep the process as close as possible to a reference signal. This criterion usually forms a quadratic function of the errors between the predicted output signal and the predicted reference trajectory. Still, there may be exceptions, as we will see.
- The control signal $u(k|k)$ is sent to the process while all the other signals are rejected as at the next sampling instant, in the strategy called Receding Horizon, $y_{(k+1)}$ is already known. At this point, we have to recalculate the previous steps with the new values again, so the new control signal $u_{(k+1|k+1)}$ will be different from the one we calculated in the first iteration $u_{(k+1|k)}$ due to this continuous update

One important thing to underline is that having a control horizon much smaller than the prediction horizon increases the performance of the system as we have to calculate fewer variables at each control interval, we have a robust system against possible delays, and we are promoting, not guaranteeing, an internally stable controller.

All the previous ideas may be better understood by looking at the below figure.

The process is schematically found in 2.2. We can obtain a series of predicted outputs compared to a reference/the desired trajectory using the model, past inputs and outputs, and proposed optimal

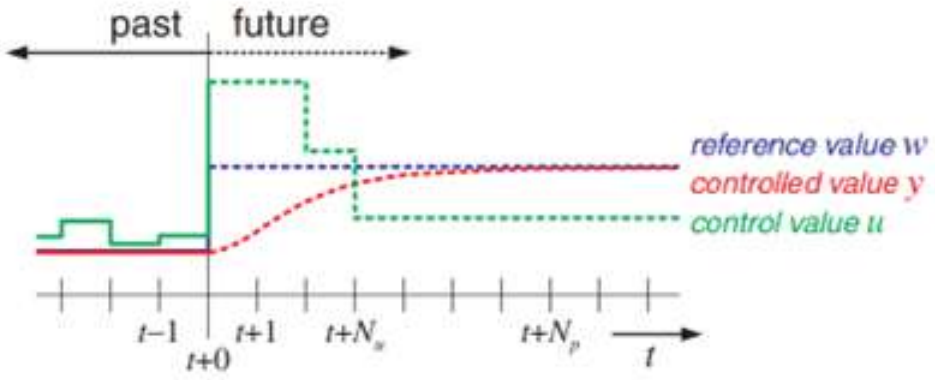

Figure 2.2: Model Predictive Control Structure

future control actions. This difference is considered at the optimizer, where a cost function and constraints are used to calculate the desired future inputs. With this scheme and explanation, we would like to highlight again the importance of the model and the optimizer, where a better objective function will provide better results.

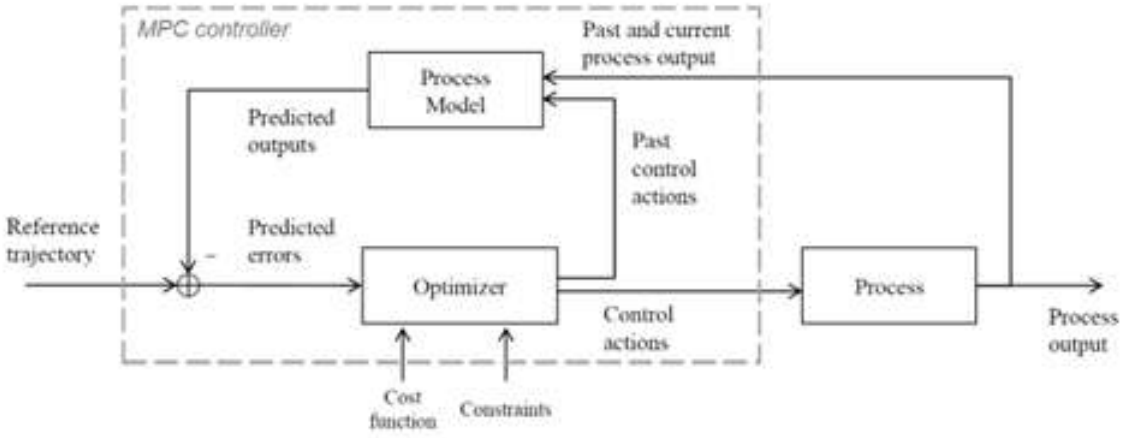

Figure 2.3: Model Predictive Control Structure

2.2 Industrial Assembly Lines from Learning Perspective

In this section, we present that heuristic, *Machine Learning (ML)* and *DRL* based techniques. The first is dedicated to heuristic and machine overall learning techniques, and in the second half, we discuss the role of *Reinforcement Learning (RL)* and its different algorithm versions.

2.2.1 Classification based on Heuristics and Classical Learning Techniques

Machine learning has different methods in the literature, such as supervised, unsupervised and *RL*. In *RL* is one type of machine learning algorithm, agents constantly adapt and learn strategies in a known or unknown environment through feedback (reward) received after each episode. The main task of the *RL* agent is to learn optimal policies to develop the scheduler regarding time minimization, efficient resource utilization and rescheduling the launch campaign activities. The

scheduling problem is traditionally solved by applying domain-specific constraints (time, capacity, computation, and resources) and involving domain experts. Traditional methods are not as efficient as current technological advancements in computation.

In [88], the author proposed using the Iterative Repair method to train the artificial agent base on some heuristics like a human expert for Scheduling and Rescheduling of *National Aeronautics and Space Administration (NASA)* NASA ground processing activities. This method's main contribution is reducing the duration of the NASA payload processing campaign at the Kennedy Space Centre. The main task of scheduling is to reduce the overall processing time and optimize the use of resources. In the Iterative Repair method, time is a constraint for job shop scheduling. In [85], the authors proposed a repair-based scheduling algorithm with a critical-path schedule with time and resource constraints.

The algorithm aims to find a short, conflict-free schedule for all activities. The *Temporal Difference (TD)* learning algorithm is applied to train a neural network to learn a heuristic evaluation function for choosing repair actions over schedules. A one-step look-ahead search procedure uses this learned evaluation function to find solutions to new scheduling problems. The author in paper [84] proposed a new time-delay neural network (TDNN) architecture to overcome the shortcoming of his previous hand-engineered method. They compared the result of the proposed algorithm with the hand-engineered method, and the proposed algorithm performed better in terms of time and schedule quality.

The advancement of artificial intelligence technology plays a promising role in manufacturing, space technology, control, etc. One branch of machine learning is a combination of different learning algorithms, both supervised and unsupervised, that is mainly depending on the already existing data. Another algorithm type is *RL*, which does not need training data.*RL* plays a promising role in decision-making applications. Scheduling is one kind of decision-making domain.

In [64], the authors formulate the job shop and flow shop scheduling problem in the *RL* domain and use the Q-learning algorithm as the decision-making agent for efficient resource utilisation and minimising the makespan of job execution. The results of Q-learning are promising as they compare with existing Operational Research (OR) Benchmarks. The author concludes that *RL* is performing better in quality and is the best alternative to *Operational Research (OR)*. As technology advanced, the complexity of problems also increased.

In [28], the authors address the problem of dynamic scheduling using combined machine learning and Gaussian processes approach. In existing scheduling techniques, machines can make decisions independently without considering the effect on the overall objective of the process. The main contribution of this article is that they consider the local and global views of machines' decisions and apply the same scenario to which already existing methods were applied. The proposed method performs better than the previous one in local minima. In [36], the author develops a mathematical model for adaptive control dispatching and gives an insightful introduction and guidelines for implementing the *RL* technique in terms of control. The authors validate this formulation for the semiconductor production industry's complex job shop scheduling problem. *Cyber-physical Sytsem (CPS)*, modern production systems and industrial sensors produce vast amounts of data, making it hard for humans to process efficiently.

The authors use this data in [35] to develop *RL* agents to handle the semiconductor industry's complex job shop scheduling problem. The authors apply the developed *RL* agent alongside existing

heuristics. In the same scenario, the agent outperforms the heuristics when the production machines' buffer size changes. Existing heuristics are limited in capacity, and an agent can train itself when interacting with a new environment and giving promising results. In [78], the authors introduce the *Deep Q-Network (DQN)* algorithm to solve the manufacturing industry's complex job shop scheduling problem. The authors formulate the scheduling problem regarding deep reinforcement learning and compare the results with existing job shop management systems and expert knowledge, showing that *DQN* performs better. They also introduce the concept of digital twins for real-time identification of any unusual event during the procedure.

The authors of article [6] propose a Q-learning algorithm to handle strict time constraints for scheduling problems. They applied the proposed algorithm to handle the real-time case study from the semiconductor industry. They compared results with established competitive heuristics, combined time constraint *First in First Out (FIFO)*, with *RL* agent performing better. There was only one case where the combined heuristic performed better, namely, when the violation of job execution happened.

In [27], the authors proposed a distributed, scalable, and generalized *RL* framework for reactive job shop scheduling problems to overcome the centralized scheduling problem. Using the graph neural network approach, they develop the inter-exchange information mechanism between the agents to take the global decision. The proposed framework solves job shop scheduling ten times faster than the existing *RL* centralized technique. In this technical report, the authors explain the complexity of Architecture Engineering Construction, Supply chain and integration model. Using a data-driven approach, they proposed technology-based *Building Information Modelling (BIM)* to manage Complex organizational behaviour. They proposed two assumptions "imaginary application of *BIM*" and "central role of the architect". They integrate the organizational behaviour, product and process using a graph-based model [55].

2.2.2 Classification based on Reinforcement Learning

Regarding applying the *RL* in the manufacturing system, we can consider it a complex and dynamic system, and many decisions should be made. For simplicity, in the first instance, we choose another typology of the production process: Shop environment, a type of manufacturing where similar production devices are grouped in closed units, and most of the products produced require a unique set-up and sequencing of process steps. In a Job Shop, when a machine becomes idle, and several jobs are waiting, we decide which job should be processed next. This optimization problem is a real-time*Job Shop Scheduling Problem (JSSP)*; jobs are assigned to resources at particular times. The sequential decision problem can often be described as an *Mark of decision Process (MDP).*

The application of *MDP* is well studied in [82], where the problem of Real-Time *JSSP* is modelled by using two algorithms, simulation-based value iteration and simulation-based Q-learning. Another real-time sequential decision-making problem is proposed in [83], where real-time batching in job shops occurs; a machine can process several jobs simultaneously, and a neural-fitted Q-learning algorithm is introduced to solve the *MDP*.

A similar approach is proposed in [74] where the optimal scheduling of arriving jobs to machines is solved by using approximate Q-learning techniques and neural networks to bottleneck identification (under the optimal scheduling policy) to improve the throughput till the desired throughput is achieved. Dynamic resource allocation is another problem that could be solved as a Markov Deci-

sion Process with a high-performance asynchronous actor-critic learning algorithm as proposed in [77]. It is a combination of the value-based reinforcement learning method and the strategy-based reinforcement learning method. The authors also used function approximation without making assumptions about the optimization goal or reducing the complexity of resource allocation computing. This needs to be improved in distinguishing its components. However, it is possible to identify this new concept of future manufacturing as autonomous factories; a first example of smart manufacturing is in [78], where a *DRL* strategy to production schedule is used. The authors apply this technique to a complex environment of work centres with different constraints, machines of several types, and multiple products. They optimize the global scheduling of production by using *DQN* agents. Each *DQN* agent optimizes the rules at one work centre while communicating and monitoring the actions of other agents to optimize a global reward.

The present thesis arises in this scenario. It aims to use a new *DRL* algorithm to solve the problem of the assembly line, trying to approach the features that industry 4.0 requires. Several production environments are being studied with the increasing complexity of assembly line manufacturing and refinement of the manufacturing process. The processes expect sophisticated communications systems, material flow plans, and production schedules. To keep the entire system running smoothly, we must demand excellent coordination between the parts of the system and redefine the problem, requiring an environment where an agent can take action and observe the results. Designing such a line is a very complex problem due to manufacturing and design constraints and due to a large number of possible decisions [79]. Significant product quality improvements and business model changes arise if manufacturing becomes automated. Then the enterprises could compete on cost and could compete based on innovation so that assembly line methods could make it possible to produce more with less.

2.2.3 Preliminaries of Learning Techniques

Three fundamental components are required to improve the agent's knowledge about the environment and make the right choice: state space, action and reward.

State

The *state* is the basic source of information about the environment we provide to the agent at every time step; indeed, this plays a crucial role in the learning mechanism of the agent since it needs to capture the correct data to allow the agent to learn. This should provide all the necessary information required to make the best decisions. In literature, we can find several ways in which it is possible to gather information about the junction.

If high-quality cameras are available, also traffic signals can be acquired, improving the state accuracy. Nevertheless, implementing the above techniques requires powerful hardware; another common approach is creating a feature-based value vector representing specific information about each lane. The great advantage is that road sensors may quickly recover the features above.

Action

The *action* is how the agent interacts with the environment; based on the information provided to the agent, this will perform the best action choosing from the available set of actions at a given

time t. In the context of traffic sign control, several standard action selections exist for a single intersection based on their flexibility. A first possibility is that the agent can choose between a defined set of light configurations which last for a fixed amount of time; the following action cannot be selected before this time is expired [23].

The second solution is similar to the first with the difference that the light duration is not fixed, but flexible [38]. Another possibility is choosing an action from a fixed set, which will be performed only under certain conditions [24].

Reward

The *reward* is a feedback mechanism that the agent uses to understand the consequence of the latest action a_t taken in the state s_t. Its role is to express the quality of the action taken at the previous time step $t-1$ by penalizing or awarding the agent. It is usually defined as a function of some performance indicator of the intersection; therefore, through the variation of the *KPI*, the agent can understand if the performed action improved the situation.

2.3 Mathematical Ingredients for Industrial System Design

2.3.1 Mathematical Parameters of Assembly Lines

We introduce some mathematical parameters used in formulating industrial assembly lines. These variables represent the location and associated function in the math metal framework of Assembly lines. A variable $l_{r,k,(w_1,w_2)}$ is used to denote the location of resources in the factory. It will take the value $l_{r,k,(wk,wk)} = 1$ if and only if the resource is at the workstation wk. On the other hand, it will take a value $l_{r,k,(w_1,w_2)} \in \{0,1\}\ \forall (w_1,w_2) \in \epsilon$, with $l_{r,k,(w_1,w_2)} = l > 0$ to indicate that the resources is on path (w_1,w_2) with l the percentage of the path travelled. Tasks, on the other hand, can have three different states, which are $\{i_{t,k}, e_{t,w,k}, f_{t,k}\}$, where $i_{t,k}, e_{t,w,k}, f_{t,k} = 1$ if and only if, respectively, task t is *idle*, *executing* (at w) or *finished* at time k.

Finally, the last key feature is workstations, which have two main variables. $o_{w,k}$ the *occupancy level* (i.e., number and type of tasks in execution) of workstation w at time k. The occupancy level dictates the number of tasks the workstation can accept each time. While $R_{w,k,r}$, the level of the input/output buffer of resources at workstation w at time k, for resource type r. This indicates capital importance, as a task assigned to the workstation can only be performed if the required resources are in place. This dictates the number/type of tasks that can be started at the workstations (since tasks need input resources to execute). The other important part of an optimization problem, as we know, is the constraints that will affect the problem. We will start with the state dynamics introduced. These laws mainly affect the workstations' occupancy, the input/output inventories, the state of the tasks and the already mentioned resource (vehicles in these cases for the Kourou spaceport) location.

- *Workstations' occupancy level* The dynamics of the occupancy level at a workstation are expressed by the authors as:

$$O_{wk,k} = \sum_{tk \in \tau K} c_{tk} e_{tk,wk,k}, \forall wk, k \tag{2.12}$$

Where c_t is the number indicating the property of the tasks, such as the number of tasks, weight or volume, depending on what we are interested in pointing out.

- *Dynamics of the workstation input/output inventories.* The level of resources at a workstation is straightforward: the e level of resources currently located at the workstation minus the resources consumed and the resources generated/freed by the finishing tasks.

- *Dynamics of the vehicles' locations:* This is one of the main features introduced. Some resources may be needed to use the graph of the workstations to transport input/output resources across the workstations to feed the tasks.

  For all $v, k, (w_1, w_2) \in \mathcal{E}$ it is obtained (3.3)

$$\begin{aligned}(\overrightarrow{g}_{v,k,w_1,w_2} + \overleftarrow{g}_{v,k,w_1,w_2})(l_{v,k+1,(w_1,w_2)} + \\ - l_{v,k,(w_1,w_2)}) = Tv_v(\overrightarrow{g}_{v,k,w_1,w_2} - \overleftarrow{g}_{v,k,w_1,w_2}),\end{aligned} \tag{2.13}$$

  where k is the sampling time, and v_v is the velocity of the resources. The equation is effective only on the links (w_1, w_2) for which $\overrightarrow{g}_{v,k,w_1,w_2} + \overleftarrow{g}_{v,k,w_1,w_2} = 1$, where it is $l_{v,k+1,(w_1,w_2)} = l_{v,k,(w_1,w_2)} + Tv_v(\overrightarrow{g}_{v,k,w_1,w_2} - \overleftarrow{g}_{v,k,w_1,w_2})$. This is a non-linear (quadratic) equation, which can be exactly linearized (as shown in [72]) to have faster solving times. In the previous expression, $\overrightarrow{g}_{v,k,w_w,w_2}$ is a boolean variable equal to one if and only if the vehicle moves from $w1$ to $w2$ at instant k and $\overrightarrow{g}_{v,k,w_1,w_2}$ $(goto)$, a Boolean variable equal to one if and only if vehicle v moves from w_2 to w_1 at k.

- *Dynamics of the task state.* Task dynamics are specified by the equations that dictate the transition from one state to another based on the decision variables. As we mentioned before, the authors use three states for the tasks: Idle, executing and finished, and the transition between them is dependent on facts such as the availability of resources, times left to complete the tasks and starting task times.

For example, to change the state from *idle to execution* when a task is assigned a to a workstation, and it has started, $s = 1$ for some wk and time k:

$$a_{tk,wk,k} <= e_{tk,wk,k}, \forall tk, wk, k \tag{2.14}$$

Apart from these constraints, we can face some others that can be called more classic, s they are pretty standard such as the unique task assignment to the workstation, unique workstation execution and task starting time, meaning that a task can be performed only once per product, a task can be in only one state, and a maximum and minimum occupancy and resource inventory constraints on the workstations. However, there are some last constraints worth mentioning that are the following ones:

- *Task time dependencies constraints* It underlines that task activation or completion also depends on another task status. Tasks can be in one of the following temporal dependence relations: Finish to Start, Start to Start, Finish to Finish or Start to Finish.

- *Vehicles localization constraints* Although not implicitly mentioned, the existence of a set of constraints to enforce the adequate displacement of

- *Resources localization constraints.* This constraint is similar to the previous one, and its existence is based on the fact that they need to be attached to a workstation and, therefore, they cannot move, or their location can change if they are placed in a vehicle.

2.4 Traffic signal from Control Perspective

One of the earliest calculation methods to minimize the average delay per vehicle was proposed in 1958; this built a foundation for traffic signal control. After that, many isolated intersection control approaches were proposed using various optimization methods such as dynamic programming, neural networks, and Model-Based approaches.

In the last 20 years many studies on *TSC* have been conducted with *MPC*, whose usage is motivated by several reasons [80]:

- Provides the designer with many degrees of freedom, allowing him to satisfy different performance requirements at the same time
- Uses real-time feedback information measured from a real-world traffic network making the best current control decisions
- The model can be easily updated or replaced by another, keeping the maintenance costs reduced

Despite the advantages and the effort devoted to improving this technique for this kind of problem, *MPC* still has to deal with its applicability in real-time scenarios due to its computational complexity. The latest technology development in the field of *AI* specifically *RL* in which agents learn as human beings from their interactions with the environment are helpful to address such complex issues. Therefore, combining *RL* and deep neural networks help solve complex control problems, particularly for high-dimensional systems.

The following section will present several literature techniques for the *MPC* approach and *DRL*.

2.4.1 Model Predictive Control: Optimization based and Waiting Time Modelling

In [65], an adaptive traffic management algorithm is presented; this provides the specific sequence of the phases and the green light time based on the waiting time of the vehicles. The junction consists of four main lanes with relative directions and a table indicating with one the feasible turn a vehicle can perform and 0 the one he cannot (See Figure 2.4). The data collected by the sensors are evaluated by a score $S = \frac{(W_u+W_v)^\alpha}{(Q_u+Q_v)^\beta}$ useful to choose the next phase that will have the green light. The values of α and β are chosen according to which objective privilege, instead of the waiting times W_u and W_v or the number of queued vehicles Q_u Q_v for the two movements u and v. The vehicle generation is modelled through a Poisson distribution as in this thesis. The dynamics of the queue length for each lane are quickly captured by considering the vehicles arriving and leaving in the two cases in which the light is red or green; in our work, instead, we introduce an arrival and escape rate, making things closer to real-world ad consider the possibility that the vehicle can pass or not the junction at the next step increasing the accuracy drastically. Also, the waiting time computation differs from ours; they consider two cases: if no vehicles are waiting from the last green

light, the waiting time is the time the first vehicle arrived during this red time, whereas if there are still vehicles from the last green light, the waiting time is the current red light time. The present work proposes a more precise formulation, considering each vehicle's waiting time at the junction. Finally, choosing the lanes with the most significant score provides the green light time required.

The last step of the optimization procedure is to minimize the sum of these functions from all directions. At the same time, in our work, the minimization is done considering the sum of all the waiting times of each vehicle in the junction, which provides a better estimate. Abbracciavento et al. propose in [2] an *MPC* formulation using a receding-horizon approach, aiming to minimise the overall number of vehicles in the queue at the traffic lights on each road. As in the present work, the idea is to design an *MPC* controller to dynamically update the light traffic states based on the measured real-time traffic information. Therefore, in this case, a linear discrete-time model is presented to describe the queue dynamics in a single intersection, which can be easily extended to all the roads. The queue dynamics are described by the following discrete-time model for each road $j = 1,...,n$:

$$x_j(t+1) = x_j(t) - \alpha_j \delta_j(t) \tag{2.15}$$

Where $x_j(t)$ is the queue length state variable, $\delta_j(t) \in (0,1)$ is a binary variable representing if the state of the traffic light (red or green) and $\alpha_j \geq 0$ is a sort of escape rate when the light is green. Despite the introduction of binary variables for describing the system dynamics, as in the present work, this model is not as precise as ours in terms of accuracy due, for example, to the absence of the yellow light, which is fundamental in real-life and that we consider in our formulation instead. The cost function adopted is the sum of two terms:

$$\min_{\delta_j} J = \min_{\delta_j} \sum_{i=1}^{T} \sum_{j=1}^{n} x_j(t+1) + w_\delta \sum_{i=0}^{T-1} \sum_{j=1}^{n} |\delta_j(t+i) - \delta_j(t+i-1)| \tag{2.16}$$

The former is to minimize the sum of queue lengths of all the lanes over the prediction horizon, and the latter is specified given, minimizing the number of changes in traffic signals over the prediction horizon. Similarly to this labour, the constraints on the state variable ensure the correct lightning sequence leading to a *Multi Integer Linear Programming (MILP)* problem, which provides the control sequence to perform at every time step. The proposed traffic control strategy has been validated, precisely as we did, through a realistic simulation environment, by employing SUMO.

The above work was also extended in [3] where the authors propose a *decentralized* MPC *strategy* in a multi-intersection road network. In particular, the objective is to expand the above reasoning to a global network; by keeping the above dynamics and considering each lane $k \in \{1, ..., K\}$

	WN	WE	SN	SW	EW	ES	NS	NE
WN		1	0	0	0	1	0	0
WE	1		0	0	1	0	0	0
SN	0	0		1	0	0	1	0
SW	0	0	1		0	0	0	1
EW	0	1	0	0		1	0	0
ES	1	0	0	0	1		0	0
NS	0	0	1	0	0	0		1
NE	0	0	0	1	0	0	1	

Figure 2.4: Intersection Model

independent from the other, the idea is to divide the computations of a single decentralizes controller into k controllers.

The *MPC* framework in [31] proposes a stable predictive controller whose state-space dynamics are used to estimate the number of vehicles at an isolated intersection and the queue length. The environment is an intersection with four legs; each vehicle entering a leg will exit from the relative exit leg.

The long queue is one of the most important parameters of the traffic flow, and they are modelled as follows:

$$Q_i(n+1) = Q_i(n) + q_i(n) - d_i(n)s_i(n) \tag{2.17}$$

where $Q_(n)$ is the length of the queue, $q_i(n)$ the vehicles entering and $d_i(n)$ the number of vehicles leaving the queue according to s_i. Similarly to the present proposal, in this work, auxiliary variables denoting the traffic light status are introduced; $s_i \in [0, 1]$ is one of the TL is green, 0 otherwise. Unlike our proposal, however, the yellow light is not considered, making the present formulation closer to reality. Even in this case, it is computed as an average for the waiting time, not considering each vehicle as we do. Its dynamics are described as follows:

$$W_i(n+1) = W_i + TQ_i(n) + q_i(n) - d_i(n)s_i(n)1/2q_i(n) \tag{2.18}$$

Instead, the control strategy is based on the duration of the green-light phase, aiming to reduce traffic volume in terms of queue length and waiting time, exactly as in this thesis. The cost function is defined as follows:

$$J = X^T(n+1)QX(n+1)\Delta S^T(n)W\Delta S(n) \tag{2.19}$$

In [81], Zammit proposes a novel adaptive control system that can self-tune to respond to changing traffic conditions, leading towards an automatic system. This uses an *MPC*-based approach that can autonomously tune the controller to ensure good performance. To enable the controller to look ahead and therefore take the best action, a prediction model is set up; this is represented in a state-space form representing each arm of a junction which mainly focuses on the state and parameter estimation problem. The state variables considered are the queue length and the inflow of vehicles entering the junction. The control inputs represent the proportion of green traffic signals in a cycle. Instead, our system is represented through a dynamical system, which explicitly expresses the variables in which we are interested and whose control inputs are provided from the optimization problem to reduce the waiting of each vehicle. Unlike ours, this article proposes several norms to reduce the queue length for all or just some arms of a multiple-arm intersection. In contrast, we focus on the total waiting time.

2.5 Traffic signal from Learning Perspective

This section presents summarize the number of attempts made by the researcher to address the traffic control issue.

2.5.1 Deep Reinforcement Learning

The authors in [40] propose a deep reinforcement learning model to control the traffic lights; this, as also done in the present thesis, incorporates multiple optimization elements such as experience replay and the target network. The work aims to optimize the efficiency of the intersection by dynamically changing the duration of the traffic light phase. The traffic scenario is modelled by expressing the whole intersection into a grid; each grid cell is a two-value vector expressing a binary value of whether a vehicle is there and its speed. The action space instead is defined by how to update the duration of each phase at each cycle; in particular, a fixed minimum duration of 5 seconds is set, as in the presented work, and at each time step, this is reduced or augmented of extra time based on the information arriving by the state space. The maximum duration is 60 seconds, while if a phase needs to be skipped, it is set to 0 seconds, which is also the minimum duration. The above concept is resumed in Figure 2.5

Also, here the yellow sign is taken into account, and in particular, its duration is defined by:

$$T_{yellow} = \frac{v_{max}}{a_{dec}} \tag{2.20}$$

Where v_{max} is the maximum speed and a_{dec} is the most common deceleration profile. In small words, this determines the length of time necessary to stop at the intersection. Since, exactly as in the proposed work, the main goal is to minimize the cumulative waiting time, the following same reward was chosen:

$$r_t = W_t - W_{t+1} \tag{2.21}$$

Where W_t is the sum of the waiting times of the vehicles at *th* cycle.

Finally, the results are validated using the traffic simulator SUMO.

A similar approach was implemented in [23] in which an adaptive traffic signal control is proposed. In this case, the algorithm's stability is improved thanks to experience replay and the target network.

As usual, also in this case, the environment consists of a four-way intersection discretized into cells indicating position and velocity; nevertheless, in this formulation, an extra two-row information on the status of the traffic light is introduced. Concerning the action, these are chosen from a fixed

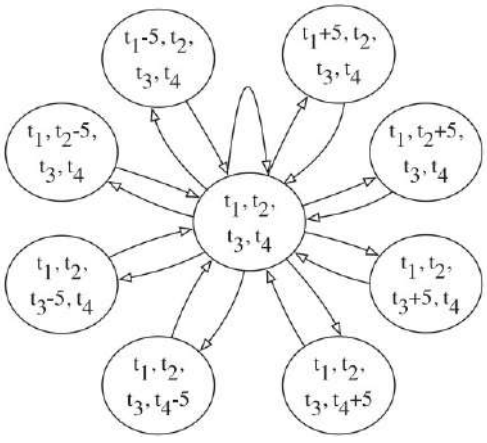

Figure 2.5: Actions Machine State

action space where each action has a fixed time interval length; similarly to the present formulation, if the action chosen at t+1 is equal to the one chosen at t, then the same phase is kept. Otherwise, the yellow configuration is set before a new action can be selected again. The most reasonable *KPI* for traffic congestion reduction is the cumulative waiting time; therefore, also in this case, Equation 2.21 is used as a reward.

In [16], the authors propose a reward function that considers the total cumulative delay of two consecutive actions and the traffic flow. The former is defined as:

$$r_t^{(1)} = D_{t-1} - D_t \tag{2.22}$$

Where D_{t-1} and D_t are total cumulative delays between the current and previous time steps. The total cumulative delay at time t is the sum of the cumulative delays of the vehicles in the simulation from t=0 until the current time; therefore, a positive reward value means that the performed actions up to that time decreased the delay. The novelty of the work is either the introduction of the traffic flow as a positive reward by looking at the occupancy of the lane; in particular, the halting vehicles in a lane are seen as a discouraging factor. The traffic flow can positively be rewarded as follows:

$$r_t^{(2)} = D_{t-1} - D_t + \frac{Occupancy}{NumberOfHaltingVehicles + c} \tag{2.23}$$

Intuitively, we are giving a positive reward to a lane fullness if the vehicles on it are not halted to encourage traffic flow. For what concerns the state representation, the usual *Discrete Traffic State Encoding (DTSE)* was used considering only the car position. The action is chosen within a set of all possible legal actions; however, to ensure safety at each transition, the controllers consider safety constraints. For this reason, the agent needs to consider intermediate traffic signals as shown in 2.6

Where the left side of the figure shows the current action configuration, the right side illustrates all possible intermediate actions that can be considered from the current action before selecting an action from the action set. Also, the authors of [29] dealt with the problem of overestimation and instability during the learning process, which they issued by combining novel optimization techniques.

The states are defined by the observation of the road every 10 second from which the agent recovers the information on the road and the lanes; in this case, a 24-dimensional vector is provided containing a part from position and velocity, information also on the number of vehicles and average velocity. The actions, uniformly to the present work, are defined as eight non-conflict phases of the duration of 10 seconds; between every transmission, there will be 5 seconds all-red period for cars in intersections to pass safely. Instead, the chosen reward is the minus value of pressure in the intersection; if the value of all the output vehicles minus all the input ones is maximized, it will mean that the traffic efficiency is high and the chosen signal phase is proper for the current traffic condition.

Ma et al., in [44], propose an adaptive optimization framework for traffic signal timing aiming to issue the limited input dimensions and enhance the performance of the reward function. In particular, to improve the decision-making execution, the input dimensions are increased by providing information on the vehicle's and road's state; even in this work, this is done by using *DTSE*,

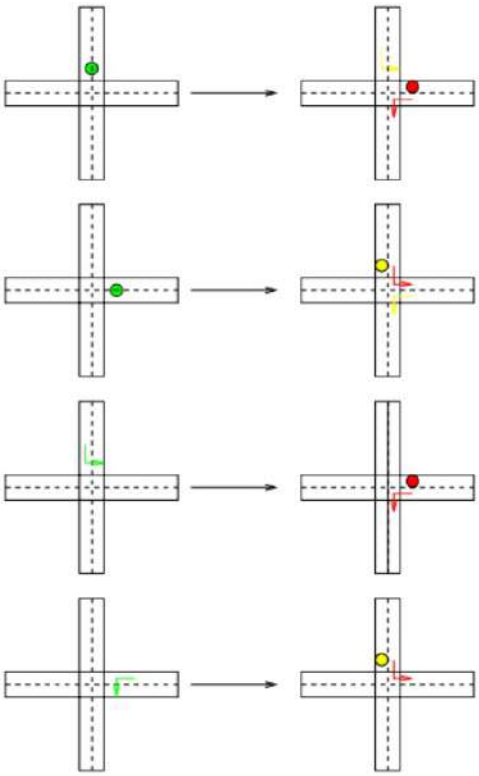

Figure 2.6: Safety Control guarantee

i.e., summarizing the traffic information by dividing the lanes into a grid. However, unlike the above works, here, more information is provided to the agent as the vehicle's direction, traffic light duration and lane density.

Based on the state information gathered from the environment, the agent has to choose an action; the action space of this study takes into account two factors: jumping to any green phase based on the traffic scenario and dynamically adjusting its duration according to the length of the queued vehicles. However, to avoid changing phases frequently, the duration of the green time must be within the range $\phi \in (T_{mingreentime}, T_{maxgreentime})$. Even in this work, a yellow phase is introduced when the current action is different from the previous one; the following formula gives this:

$$T = t + \frac{s_{85}}{2a + (19.6xG)'} \tag{2.24}$$

Where t is the driver's manoeuvre time, s_{85} is 85% of the speed limit of the intersection, a is the average deceleration, and G is the slope of the entrance lane of the intersection. The definition of the reward equation is measured from two dimensions; the first is a combination of the following equations:

$$r_{t1} = W_t - W_{t+1} \tag{2.25}$$

Where the meaning of W_t is explained below.

$$W_t = \sum_{t=0}^{N} w_{s,\epsilon} \tag{2.26}$$

Where ϵ is the number of vehicles queuing at the intersection, the greater the change in the accumulated waiting time before and after the action is performed, the greater the reward value. The second contribution is aimed at avoiding long green time through a penalty term:

$$r_{t2} = -\alpha max[(T_t - T_{maxgreentime}), 0] \tag{2.27}$$

Where T_t represents the duration of the corresponding green time at time step t and α is the control coefficient which weighs how much to punish or favour the performed action through the reward. When multiple green phases occur in succession and exceed the set value, a penalty will be given to avoid traffic flow unbalance in all directions at the intersection. The final reward function will therefore be the summation of the above function, which leads to the following:

$$R_t = r_{t1} + r_{t2} = (W_t - W_{t+1}) - max[(T_t - \alpha T_{maxgreentime}), 0] \tag{2.28}$$

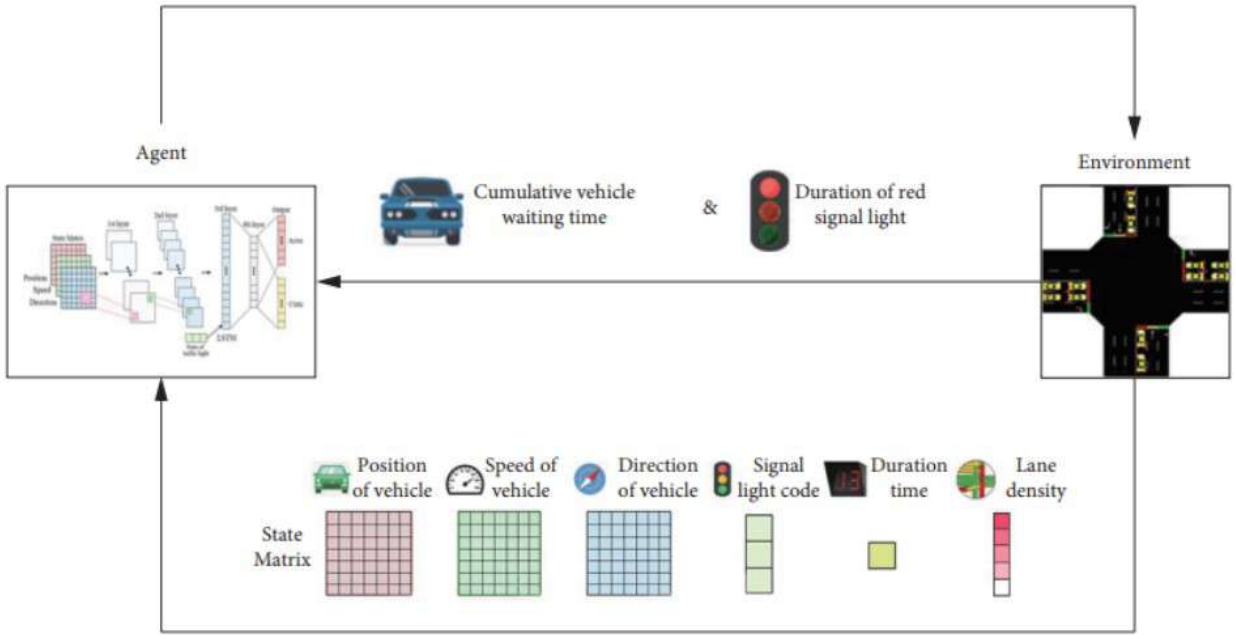

Figure 2.7: Intelligent traffic light system framework based on RL

Chapter 3

Control Framework Modeling

This chapter presents a comprehensive mathematical formulation of both use cases from the perspective of control and *DRL*. Firstly, we model the manufacturing and city logistics use case in terms of *MPC*. The modelling is helpful for the implementation and real-time deployment of the developed software suite. Secondly, both use cases are formulated in terms of *Mark of decision Process (MDP)* that, 's is a common framework for *RL* problem formulation. Further, the training and mechanism of *DRL* agents and respective *Neural Network (NN)* structure are present. The validation of the developed framework is tested in different scenarios, and results and compression are presented in the numerical results chapter.

3.1 Smart Factory Use Case Description and Modelling in terms of MPC

This section presents the mathematical modelling of smart factory use cases, specifically Industrial assembly lines.

3.1.1 Introduction

The objective of this use case is to maximize the production capabilities of the European spaceport in Kourou; to achieve this goal, We consider the problem of optimal scheduling and then controlling in real-time the execution of a given set of tasks in a given assembly line. The task characteristics (duration, needed input resources and resulting output resources) are known inputs of the problem. Also, the layout and the characteristics of the assembly line are known, for example, in terms of available workstations, technical specifications of the workstations, tools and personnel available. Further, we first model the objects in the problem (tasks, workstations, vehicles, etc.) and then outline the mathematical formulation of the proposed optimal task control problem.

3.1.2 Factory Layout and Common Parameters

We model the overall factory as a graph $\mathcal{G} = \{\mathcal{W}, \mathcal{E}\}$, where the set of the graph's nodes $\mathcal{W}$ is the set of workstations and $\mathcal{E}$ is the set of transportation paths connecting the workstations. By convention, we let $(w_1, w_2) \in \mathcal{E}$, only if $w_1 < w_2$.

Task

Tasks $\mathcal{T}$ is the set of all tasks to be planned/controlled. Tasks are characterised by a duration d_t, the earliest allowed starting time, S_t, and the latest allowed finish time, F_t. Each task is unique, even though it may have a different assigned duration depending on the product model. For each task, there will also be defined two new sets: Predecessor $prec_{tk}$ and succeeding tasks $succ_{tk}$. Being only the direct preceding and succeeding ones. As the tasks must follow an order to produce the item, a Precedence Graph is required to show the task precedence relations. Task states are implemented to show the status of each model task at a given time.

- *Waiting:* is used to indicate that a task has not started because the precedent tasks are not completed
- *Active:* when all previous tasks are completed, in the next time step, the current task will start, and so, it will become active
- *Finished:* this state indicates that all actions to perform the task have been done, also represented as the time left being zero. This allows the succeeding tasks to change from "Waiting" to "Active" states.
- *Reorganizing:* this state shows the necessity for a task to be stopped whenever it has started, and at the same time, in a given time instant, that task is moved from the original workstation to a new one. After this has been done, the task will return to the previous state until completion.

Whenever the initial task state is either Waiting or Reorganizing, we have to check two more conditions:

- There are no previous tasks to the task we are checking or all its precedent tasks have been completed and are in Finished status. The other condition is that the time left to complete the task exceeds 0. If all these conditions are fulfilled, it is the moment for the task to become Active.
- If the task was in Active status and the time left to complete it is 0, it means it has been properly done, and the new task's status must be Finished.
- *Task state variables.* $\{i_{t,k}, e_{t,w,k}, f_{t,k}\}$, where $i_{t,k}, e_{t,w,k}, f_{t,k} = 1$ if and only if, respectively, task t is *idle*, *executing* (at w) or *finished* at time k.

Workstations

Workstations $\mathcal{W}$ are the specific machines and/or sectors of the assembly line where the tasks are executed. $\mathcal{W}_t$ denotes the set of workstations that can work task $t \in \mathcal{T}$. We model the workstation as a system composed of a processing unit with an internal capacity (which limits the maximum number of tasks that can be worked in parallel) and a buffer for storing the input and output resources. W_k are the physical places to perform the tasks. Each of them has some limitations, like the maximum number of tasks that can be assigned. As the workstations are a limited resource, we need to know if they have already been assigned a task and if they are doing any labour. For

this reason, states were introduced to show the delay that may appear when a rescheduling has to be made:

- *Waiting:* as in the previous case, it indicates that currently, there are not any active tasks in it
- *Active:* is a straightforward state that is closely related to the task state Active, as if any of its assigned tasks is at that state, the workstation will be too to indicate that there is labour being performed.
- *Not operative:* this state indicates that there has been a malfunction at the workstation and that it is out of service, which will lead to rescheduling to obtain the new optimal task placement of the tasks that were assigned to the workstation that is no longer available. It can be applied in the case that predictive maintenance is scheduled.

The way workstations change their status is easier than for tasks.

- As expected, at a time equal to 0, all workstations must be in the Waiting state. Moreover, if, at any given moment, a workstation has one or more tasks assigned, and they are not in Active status, the workstation will be in an idle situation. Therefore, the status must be Waiting for too
- . When a fault is detected, or there is a problem inside the workstation, it cannot continue performing any task, and its new state must not be operative until it has been solved. Any workstation with this status will not be eligible at the optimizer to be assigned tasks.
- When a task has been assigned to the workstation, and it is in an Active status, it means that the workstation is doing any labour and consequently it is Active

A particular workstation type is the source and storage points of material where the resources are stored.

1. *Workstation state variables.*
   (a) $o_{w,k}$ the *occupancy level* (i.e., number and type of tasks in execution) of workstation w at time k. The occupancy level dictates the number of tasks that can be accepted at each given time by the workstation;
   (b) $R_{w,k,r}$, the input/output buffer level of resources at workstation w at time k, for resource type r. This dictates the number/type of tasks that can be started at the workstations (since tasks need input resources to execute).

Resources

$\mathcal{R}$ are the objects or operators of the factory procedure that can be assigned to the workstations to perform the tasks. Each task must require one or more resources to be carried out. Not only the straightforward items that come to mind are designed as Resources, such as screws, screwdrivers, drillers, and robots, but also we can point out that human operators can be counted as resources. This derives from the fact that they are a production system component abstractly. Another

perspective to consider, and the one applied later, is assigning a set of the previously mentioned tools as a resource. This is because if a workplace is specialized in a series of tasks, the tools are grouped into a set.

$\mathcal{R}$ is the set of all the resources (tools, materials, people, etc.) available in the assembly line. Let $\mathcal{R}^t$ denote the set of resource types (two or more resources can be the same type). Examples of resource types are $\mathcal{C}$ consumables, i.e., resources which are consumed by the tasks (e.g., water, power, chemicals, parts to be assembled, etc.), $\mathcal{I}$, instruments/tools/operators needed to execute a task (freed after the task ends), $\mathcal{V}$, the set of vehicles, etc. Tasks consume and produce resources. We denote with R_t^i the input resources required by task t and with R_t^o the output resources/products produced by task t.

Resources Location. Variable $l_{r,k,(w_1,w_2)}$ denotes the location of resources in the factory (i.e., over the graph $\mathcal{G}$). It is $l_{r,k,(w,w)} \in \{0,1\}$ (i.e., *Boolean* variable), $\forall w \in \mathcal{W}$, with $l_{r,k,(w,w)} = 1$ if and only if the resource is at the workstation w. It is instead $l_{r,k,(w_1,w_2)} \in [0,1]$ (i.e., *continuous* variable) $\forall (w_1, w_2) \in \mathcal{E}$, with $l_{r,k,(w_1,w_2)} = l > 0$ meaning that the resource is on path (w_1, w_2), with l the percentage of the path travelled.

3.1.3 Definition of Variables and Constrains

The control variables are:

1. $a_{t,w,k}$ ("assign"), a Boolean variable equal to one if and only if task t is assigned to workstation w at time k. A second task control variable is $s_{t,k}$, equal to one if and only if task t is started at time k. It is:
$$s_{t,k} = \sum_{w \in \mathcal{W}_t} a_{t,w,k}, \ \forall t, k. \tag{3.1}$$

2. $p_{r,v,k}$ ("place"), a Boolean variable equal to one if and only if resource $r \in \mathcal{R}$ (e.g., a driver, a tool, a semi-finished component) is placed on the vehicle v at time k.

3. $\overrightarrow{g}_{v,k,w_1,w_2}$ ("goto"), a Boolean variable equal to one if and only if vehicle v moves from w_1 to w_2 at k;

4. $\overleftarrow{g}_{v,k,w_1,w_2}$ ("goto"), a Boolean variable equal to one if and only if vehicle v moves from w_2 to w_1 at k.

Definition of the State Dynamics

We specify in the following the laws according to which the system's state evolves in time.

1. *Workstations' occupancy level.* The following equation captures the dynamics of the occupancy level at a workstation at a given time:
$$o_{w,k} = \sum_{t \in \mathcal{T}} c_t e_{t,w,k}, \ \forall w, k. \tag{3.2}$$

   The occupancy level can capture different aspects depending on the task and the workstation types. For example, it can be simply the number of tasks executing at the workstation (in which case, it is $c_t = 1$), or the volume occupied, or the weight (in which case c_t is a volume or weight, etc.).

2. *Dynamics of the workstation input/output inventories.* The level of resources at a workstation is equal to the level of resources currently located at the workstation, minus the resources consumed by starting tasks plus the resources generated/freed by the finishing tasks;

3. *Dynamics of the vehicles' locations*: Vehicles move on the graph of the workstations to transport input/output resources across the workstations to feed the tasks. The general equation of their dynamics is, for all $v, k, (w_1, w_2) \in \mathcal{E}$

$$\begin{aligned}(\overrightarrow{g}_{v,k,w_1,w_2} + \overleftarrow{g}_{v,k,w_1,w_2})(l_{v,k+1,(w_1,w_2)} + \\ - l_{v,k,(w_1,w_2)}) = Tv_v(\overrightarrow{g}_{v,k,w_1,w_2} - \overleftarrow{g}_{v,k,w_1,w_2}),\end{aligned} \tag{3.3}$$

where T is the sampling time, v_v the velocity of the vehicle. The equation is effective only on the links (w_1, w_2) for which $\overrightarrow{g}_{v,k,w_1,w_2} + \overleftarrow{g}_{v,k,w_1,w_2} = 1$, where it is $l_{v,k+1,(w_1,w_2)} = l_{v,k,(w_1,w_2)} + Tv_v(\overrightarrow{g}_{v,k,w_1,w_2} - \overleftarrow{g}_{v,k,w_1,w_2})$. This is a non-linear (quadratic) equation, which can be exactly linearized (as shown in [72]) to have faster solving times.

4. *Dynamics of the task state.* Task dynamics is specified by equations that dictate the transition from one state to another based on the decision variables. For example, the state transits to "execution" when $s_{t,k} = 1$, i.e., when $a_{t,w,k} = 1$ for some w:

$$a_{t,w,k} \leq e_{t,w,k}, \ \forall t, w, k. \tag{3.4}$$

Then, let $d_{t,k}$ denote the number of time steps left to complete task t at time k. If a task is in the execution state, i.e., $e_{t,w,k} = 1$ for some w, then it shall remain in the execution state as long as $d_{t,k} > 0$. If $d_{t,k} = 0$, then the task shall no longer be in the execution state, i.e.,

$$e_{t,w,k} \leq d_{t,k}, \ \forall t, w, k. \tag{3.5}$$

And will go to the "finished" state. Finally, it is not possible to go back from the finished state:

$$f_{t,k} \leq f_{t,k+1}, \ \forall t, k. \tag{3.6}$$

Other Problem Constraints

The remaining problem constraints are as follows:

1. *Unique task starting time.* Only one starting time can be chosen for every task, between the earliest allowed starting time and the latest feasible one

$$\sum_{k=S_t}^{F_t - d_t} s_{t,k} = 1, \ \forall t. \tag{3.7}$$

2. *Unique workstation of execution.* A task can be in execution at most at one workstation

$$\sum_{w \in \mathcal{W}} e_{t,w,k} \leq 1, \ \forall t, k. \tag{3.8}$$

3. *Unique task assignment to workstation.* A task can be assigned to one and only one workstation, only once.

$$\sum_{k=S_t}^{F_t-d_t} \sum_{w \in \mathcal{W}} a_{t,w,k} = 1, \ \forall t. \tag{3.9}$$

4. *Task time dependencies constraints.* Tasks can be in one of the following temporal dependence relations:

   (a) *Finish to Start*: every task t', predecessor of task t, must finish before t can start:

$$d_{t'} + \sum_{k=S_{t'}}^{F_{t'}-d_{t'}} ks_{t',k} \leq \sum_{k=S_t}^{F_t-d_t} ks_{t,k}. \tag{3.10}$$

   Note that the left-hand side represents the finish time of task t', while the right-hand side is the start time of task t;

   (b) *Start to Start*: every task t', predecessor of task t, must start before t can start:

$$\sum_{k=S_{t'}}^{F_{t'}-d_{t'}} ks_{t',k} \leq \sum_{k=S_t}^{F_t-d_t} ks_{t,k}; \tag{3.11}$$

   (c) *Finish to Finish*: every task t', predecessor of task t, must finish before t can finish:

$$d_{t'} + \sum_{k=S_{t'}}^{F_{t'}-d_{t'}} ks_{t',k} \leq d_t + \sum_{k=S_t}^{F_t-d_t} ks_{t,k}; \tag{3.12}$$

   (d) *Start to Finish*: every task t', predecessor of task t, must start before t can finish:

$$\sum_{k=S_{t'}}^{F_{t'}-d_{t'}} ks_{t',k} \leq d_t + \sum_{k=S_t}^{F_t-d_t} ks_{t,k}. \tag{3.13}$$

5. *Task state constraints.* A task can be in one and only one state at every given time.

$$i_{t,k} + \sum_{w \in \mathcal{W}_t} e_{t,w,k} + f_{t,k} = 1, \ \forall t, k. \tag{3.14}$$

6. *Workstations occupancy constraints.*

$$o_w^{min} \leq o_{w,k} \leq o_w^{max}, \ \forall w, k. \tag{3.15}$$

7. *Workstation resource inventory constraints.*

$$R_{w,r}^{min} \leq R_{w,k,r} \leq R_{w,r}^{max}, \ \forall w, k, r \in \mathcal{R}^t. \tag{3.16}$$

The above constraints also ensure that a task is assigned to a workstation only if there are enough input resources and enough “space" to store the output resources.

8. *Vehicles localization constraints.* A complex set of constraints is included to enforce the proper movement of vehicles along the paths connecting the workstations.

9. *Resources localization constraints.* Another set of constraints is needed to model the movement of resources across the assembly line. In short, resources cannot move if not associated with a vehicle. On the contrary, when they are associated with a vehicle, their location variables will coincide with the location variables of the vehicle they are associated with.

3.1.4 Objective Function

The objective function is written as the convex combination of several terms, reflecting different optimization goals:

$$V = \sum_{i=1} \alpha_i V_i, \tag{3.17}$$

with $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$.

Possible relevant terms are discussed in the following.

1. V_1 (*minimization of task completion time*). We pose:

$$V_1 = \tau \tag{3.18}$$

Where τ is an auxiliary variable greater or equal to all the finish times.

$$\tau \geq d_t + \sum_{k=S_t}^{F_t - d_t} k s_{t,k} \quad \forall t \tag{3.19}$$

Hence, when minimized, τ represents the latest task finish time, and the inclusion of V_1 minimizes it;

2. V_2 (*minimization and balancing of the input/output resource inventory*).

$$V_2 = \sum_{w,k,r} \left(R_{w,k,r} - R_{w,r}^{ref} \right)^2 \tag{3.20}$$

where $R_{w,k}^{ref}$ is the reference level of the resources inventories (if put to zero, inventory is minimized);

3. Cost minimization. Cost minimization can be easily integrated, for example, to consider the operational cost of utilizing workstations, the cost of storing inventory, the penalty cost of early or late delivery, etc.

In the following simulations, we will consider only the minimization of the overall task completion time (i.e., the cycle time).

3.2 Smart Transportation Use Case Description and Modelling in terms of MPC

This section presents the mathematical modelling of the smart transportation use case, specifically Traffic lights.

3.2.1 Introduction

The objective of the considered use case is to reduce the vehicle waiting time and, consequently, queue balancing at junctions via optimal traffic signal scheduling. In [41], the problem of reducing the number of cars halting in a junction was considered. However, this method is unsuitable for small queue lengths of vehicles as the controller prioritizes long queues at every time step. We consider a more accurate mathematical model formulation is presented in which the waiting time of each car at the junction is also taken into account, ensuring a better control strategy and optimizing the new *Key Performance Indicator (KPI).*

3.2.2 Mathematical Modeling of Traffic System and Components

The following section will describe the variables, constraints, traffic queue lengths, waiting for time vehicle dynamics and objective function to optimize.

3.2.3 Variable Definition

Simple definitions of variables further which are further used in system definition as follows:

$$\mathcal{I} = \{(i,j) : i = k, ..., k+N-1, j = 1, ..., L\}, \tag{3.21}$$

$$\mathcal{I}' = \{(i,j) : i = k, ..., k+N-2, j = 1, ..., L\}, \tag{3.22}$$

$$\begin{aligned}\mathcal{J} = \{(i,j,v) : i = k, ..., k+N-1, j = 1, ..., L,\\ v \in V_j(k)\},\end{aligned} \tag{3.23}$$

$$\begin{aligned}\mathcal{J}' = \{(i,j,v) : i = k, ..., k+N-2, j = 1, ..., L,\\ v \in V_j(k)\},\end{aligned} \tag{3.24}$$

where $V_j(k)$ denotes the set of vehicles at queue j at time k. The index $j \in [1, 2, ..., L]$ denotes a generic TL of the road junction, while the index i is used to denote a generic time step in the prediction horizon N.

The state of the generic TL, j at time i is given by the triple Boolean variables $g_{i,j}, y_{i,j}, r_{i,j}$, green, yellow and red status, respectively. The yellow light in between the green and red states requires extra variables; in particular, for safety reasons, we need to introduce a minimum duration time t_i^y. This indicates the duration of the current yellow cycle preventing continuous and repeated light changes, which may lead to collisions. The Boolean variable $y_{i,j}^{\uparrow}$ highlights the changing of the light from green to yellow, namely, if the TL passes from green to yellow, $y_{i,j}^{\uparrow}$ is equal to one, zero otherwise. To keep the formulation linear, the nonlinear term $t_{i,j}^y y_{i,j}^{\uparrow}$ is equivalently written in a linear form by including the auxiliary variable $\hat{t}_{i,j}^y$ and a set of constraints which will be explained later.

Concerning the modelling of the queue, the dynamics of the number of vehicles halting at time i at TL j is represented by the following:

$$n_{i+1,j} = n_{i,j} + T[v^{in}_{i,j} - v^{in}_{i,j}\check{g}_{i,j} - v^{esc}_{i,j}\hat{g}_{i,j}] + s_{i,j}, \quad \forall (i,j) \in \mathcal{I}'. \tag{3.25}$$

Where v^{in} and v^{esc} are the arrival and departure rates of the vehicles, and T is the sampling time. Since the number of vehicles halting $n_{i,j}$ must be greater or equal to zero, the slack variable $s_{i,j}$ was added to prevent that infeasibility arises due to $n_{i+1,j}$ becoming negative in specific conditions; e.g., when $n_{i,j} = 1$, $v^{in}_{i,j} - v^{esc}_{i,j} \leq -1$ vehicles/second and $T \geq 1$ second. The objective function will include a dedicated term to ensure that $s_{i,j}$ is greater than zero only when needed. Moreover, if there is no queue $n_{i,j} = 0$ and TL is green $g_{i,j} = 1$. The arrival and departure rate would be equal $v^{out}_{i,j} = v^{in}_{i,j}$; on the other hand, when $n_{i,j} \geq 0$ ant the TL is green, $v^{out}_{i,j}$ will be equal to an "escape rate" which will depend on the characteristics of the vehicle, the road etc. This rate can be estimated based on measurements (See Section Future Work). To model the above behaviour related to $v^{out}_{i,j}$, the Boolean variable $\bar{n}_{i,j}$ is introduced to indicate if $n_{i,j}$ is zero or greater than zero.

The variables $\hat{g}_{i,j} := \bar{n}_{i,j} g_{i,j}$ and $\check{g}_{i,j} := (1 - \bar{n}_{i,j}) g_{i,j}$ in 3.25 are helpful, through some additional constraints explained afterwards, to describe the traffic light situation when a vehicle is approaching the junction. Let $w_{i,j,v}$ denote the cumulative waiting time of vehicle v in queue j at time i. The dynamics of the waiting time are:

$$w_{i+1,j,v} = (w_{i,j,v} + T) n^{+}_{i+1,j,v}, \quad \forall (i,j,v) \in \mathcal{J}'. \tag{3.26}$$

where $V_{k,j}$ is the set of vehicles at the queue j at time k. This is updated at every instant of time with the vehicles leaving and the new vehicles arriving. Moreover, the set $V_{i,j}$ needs to be updated as well within the control window $(i,j) \in \mathcal{I}$ considering the vehicles that are predicted to leave and arrive at the junction during the control window. The former update is based on real measurements, the latter on estimated arrival and departure rates. For the latter, it holds:

$$V_{i+1,j} = \{v \in V_{i,j} : n^{+}_{i+1,j,v} = 1\} \cup A_{i+1,j}, \quad \forall (i,j,v) \in \mathcal{J}'. \tag{3.27}$$

where $A_{i+1,j}$ is the set of vehicles expected to arrive at time i+1. Equation 3.26 states that the waiting time of a certain vehicle v increase as long as the vehicle is in the queue while, when it leaves, then $w_{i+1,j,v}$ is set to zero. This is made possible thanks to the Boolean indicator variable $n^{+}_{i,j,v}$ in Equation 3.26, which is the Boolean indicator variable of $\delta_{i,j,v}$; it is equal to 1 if $\delta_{i,j,v} > 0$ and equal to zero otherwise. The dynamics of $\delta i, j, v$ is:

$$\delta_{i+1,j,v} = \delta_{i,j,v} - T v^{esc}_{j} g_{i,j}, \quad \forall (i,j,v) \in \mathcal{J}. \tag{3.28}$$

The variable $n^{+}_{i,j,v}$ in fact, will be equal to one if $n^{q}_{i,j,v}$ is positive and zero otherwise; its dynamics can be expressed as:

$$n^{+}_{i+1,j,v} = n^{+}_{i,j,v}(y_{i,j} + r_{i,j} + g_{i,j}\bar{p}_{i,j,v}), \quad \forall (i,j,v) \in \mathcal{J}'. \tag{3.29}$$

Where the Boolean auxiliary variable $\bar{p}_{i,j,v}$ captures if vehicle v manages to pass the junction or not; in particular, it will be equal to one if the vehicle **does not** manage to pass the junction and

equal to zero otherwise.

3.2.4 Constraint Definition

The model captures all the constraints that must be respected to enforce the correct functioning of the road junction and the TLs over the prediction horizon.

Single State Constraint

Firstly, we want our traffic lights to be in just one state at a time, and therefore it must be:

$$g_{i,j} + y_{i,j} + r_{i,j} = 1, \quad \forall i, j \tag{3.30}$$

It is straightforward to see that only one variable between $g_{i,j}, y_{i,j}$ and $r_{i,j}$ can be equal to one.

Light Transition Order

The lights must follow the right order of transition, hence, avoiding going from green to red, from red to yellow and from yellow to green; this is ensured by:

$$\begin{aligned} r_{i+1,j} &\leq 1 - g_{i,j}, \quad \forall i, j, \\ y_{i+1,j} &\leq 1 - r_{i,j}, \quad \forall i, j, \\ g_{i+1,j} &\leq 1 - y_{i,j}, \quad \forall i, j. \end{aligned} \tag{3.31}$$

It is straightforward to see, for example, from 3.31 that if $g_{i,j} = 1$, then it must be $r_{i+1,j} = 0$.

Yellow Time Minimum Duration

For safety reasons, a minimum duration time for the yellow is set; this is useful to avoid the lights changing too fast, preventing the vehicles from starting and stopping continuously. The auxiliary variable that indicates the duration of the current yellow cycle up to time i is:

$$t^y_{i+1} = t^y_{i,j} + y_{i,j} - t^y_{i,j} y^{\uparrow}_{i,j}, \quad \forall (i, j) \in \mathcal{I}'. \tag{3.32}$$

It is evident to see that the yellow time will increase as long $y_{i,j} = 1$ and will return to zero as soon $y^{\uparrow}_{i,j} = 1$, i.e., as soon the light goes from green to yellow, in order to start again counting. The behaviour of $y^{\uparrow}_{i,j}$ is achieved via the inclusion of the following constraint:

$$y_{i+1,j} - y_{i,j} \leq (1 + \epsilon) y^{\uparrow}_{i+1,j} \leq y_{i+1,j} - y_{i,j} + 1, \quad \forall (i, j) \in \mathcal{I}'. \tag{3.33}$$

with ϵ an arbitrarily small positive constant. When there is a transition from green to yellow, i.e., when $y_{i+1,j} - y_{i,j} = 1$, Equation 3.33 forces $y^{\uparrow}_{i+1,j}$ to one. On the other hand, when $y_{i+1,j} - y_{i,j} = 0$ (i.e., when the yellow sign does not vary) or when $y_{i+1,j} - y_{i,j} = -1$ (i.e., the sign goes from yellow to red), then (3.33) forces $y^{\uparrow}_{i+1,j}$ to zero, which is exactly the behaviour sought for $y^{\uparrow}_{i+1,j}$.

As anticipated in the previous section, the nonlinear term $t^y_{i,j} y^{\uparrow}_{i,j}$ in 3.32 can be rewritten in a linear form by including the auxiliary variable $\hat{t}^y_{i,j}$ and the constraint:

$$\begin{cases} 0 \leq \hat{t}^y_{i,j} \leq t^{y,max}_{i,j} y^{\uparrow}_{i,j} \\ t^y_{i,j} - (1 - y^{\uparrow}_{i,j}) t^{y,max}_{i,j} \leq \hat{t}^y_{i,j} \leq t^y_{i,j}, \forall i, j \end{cases} \tag{3.34}$$

It can be shown then, through the above constraint that $\hat{t}^y_{i,j} = t^y_{i,j}$ when $y^{\uparrow}_{i,j} = 1$, and $\hat{t}^y_{i,j} = 0$ when $y^{\uparrow}_{i,j} = 0$. Indeed, Equation 3.32 is substituted with:

$$t^y_{i+1,j} = t^y_{i,j} + y_{i,j} - \hat{t}^y_{i,j}, \quad \forall (i,j) \in \mathcal{I}'. \tag{3.35}$$

Nevertheless, the minimum duration of the yellow light is ensured by the following constraint:

$$1 - \frac{t^y_{i,j}}{t^{y,min}_{i,j}} \leq y_{i,j}, \quad \forall i, j \tag{3.36}$$

where $t^{y,min}_{i,j}$ denotes the minimum duration of the yellow sign. Equation (3.36) will in fact force $y_{i,j}$ as long as $t^y_{i,j} \leq t^{y,min}_{i,j}$ to ensure that the yellow sign lasts at least a certain time.

Queue at a Traffic Light

Concerning the modelling of the queue at a TL, starting from the following approximated dynamics, we will recover the appropriate constraints to ensure the correct behaviour.

$$n_{i+1,j} = n_{i,j} + T(v^{in}_{i,j} - v^{out}_{i,j} g_{i,j}), \quad \forall (i,j) \in \mathcal{I}'. \tag{3.37}$$

The two arrival and departure rates are equal when there is no queue and the TL is green; whereas, if $n_{i,j} \geq 0$ and $g_{i,j} = 1$, $v^{out}_{i,j}$ will be equal to an "escape rate" depending on the features of the junction. This behaviour is modelled through the below simple constraints:

$$n_{i,j} \leq M \bar{n}_{i,j}, \quad \bar{n}_{i,j} \leq M n_{i,j}, \quad \forall i, j \tag{3.38}$$

where M is a large positive number (larger than the maximum value that $n_{i,j}$ can take). In such a way, if $\bar{n}_{i,j} = 0$ (i.e. there is no queue) and $g_{i,j} = 1$, then $v^{out}_{i,j} = v^{in}_{i,j}$, while if $\bar{n}_{i,j} = 1$ and $g_{i,j}$ (i.e. there is a queue and the light is green) we have $v^{out}_{i,j} = v^{esc}_{i,j}$. In order to capture these two functioning $\hat{g}_{i,j} := \bar{n}_{i,j} g_{i,j}$ and $\check{g}_{i,j} := (1 - \bar{n}_{i,j}) g_{i,j}$ the following constraints are introduced:

$$\hat{g}_{i,j} := \bar{n}_{i,j} g_{i,j} \ \textit{ is equivalent to } \begin{cases} \hat{g}_{i,j} \leq \bar{n}_{i,j}, \\ \hat{g}_{i,j} \leq g_{i,j}, \\ \hat{g}_{i,j} \geq \bar{n}_{i,j} + g_{i,j} - 1, \forall i, j \end{cases} \tag{3.39}$$

$$\check{g}_{i,j} := (1 - \bar{n}_{i,j}) g_{i,j} \ \textit{ is equivalent to } \begin{cases} \check{g}_{i,j} \leq 1 - \bar{n}_{i,j}, \\ \check{g}_{i,j} \leq g_{i,j}, \\ \check{g}_{i,j} \geq -\bar{n}_{i,j} + g_{i,j}, \forall i, j \end{cases} \tag{3.40}$$

Moreover, these also allow rewriting (3.37) as (3.25).

Waiting Time Constraints

To update the queue, it is fundamental to know the vehicles arriving at the TL and in particular, the ones leaving; for this matter, the following constraints are introduced. From Equation 3.29 it is seen that if $n^+_{i,j,v} = 0$ (i.e. the vehicle has passed the junction), then $n^+_{i+1,j,v} = 0$ as expected. This is easily modelled by:

$$n^+_{i+1,j,v} \leq M n^+_{i,j,v} \tag{3.41}$$

Furthermore, if $n^+_{i,j,v} = 1$ and the status of the TL is yellow or red, then it must be $n^+_{i+1,j,v} = 1$ which is imposed by the following:

$$n^+_{i,j,v} + y_{i,j} + r_{i,j} \leq 1 + n^+_{i+1,j,v} \tag{3.42}$$

In fact if $n^+_{i,j,v} = 1$ and $y_{i,j}$ or $r_{i,j}$ then $n^+_{>} i+1, j, v$ is forced to be equal to 1. Finally, if $\delta_{i,j,v} = 1$ and the status of the TL is green, the status of $\delta_{i+1,j,v}$ will depend on whether or not the vehicle will manage to pass the intersection from time i to the next time $i+1$ (i.e., in T seconds, the sampling period). This will depend on the current position of the vehicle in the queue (i.e., on $n_{i,j,v}$), and on how fast the vehicles pass the intersection (i.e., on $v^{out}_{i,j}$). Then, with a good approximation, if $n_{i,j,v} \geq T v^{out}_{i,j}$ (i.e., $n_{i,j,v} - T v^{out}_{i,j} \geq 0$) the vehicle will not manage to pass the intersection from time i to time $i+1$, and $\delta_{i+1,j,v}$ will be still equal to one. The following constraint captures this:

$$\frac{n_{i,j,v} - T v^{out}_{i,j}}{M} \leq \delta_{i+1,j,v}, \forall (i,j,v) \in \mathcal{J}'. \tag{3.43}$$

If instead $n_{i,j,v} \leq T v^{out}_{i,j}$ and the TL is green ($g_{i,j} = 1$), the vehicle will manage to pass the intersection, and it will be $\delta_{i+1,j,v} = 0$. The following constraint can model this:

$$\frac{T v^{out}_{i,j} - n_{i,j,v}}{M} \leq 2 - \delta_{i+1,j,v} - g_{i,j}, \forall (i,j,v) \in \mathcal{J}'. \tag{3.44}$$

Conflict Avoidance Constraints

Usually, in a junction, the lanes may intersect, and therefore potential collisions are possible; in the crossroad of the present work depicted in Figure 3.1, there are three conflicting set lanes $C_i \ \forall i \in N$: $C_1 = \{1, 3\}$, $C_2 = \{1, 2, 4\}$ and $C_3 = \{2, 5\}$. To avoid any collisions, we impose that a maximum of one TL per conflicting set C_i can be green or yellow at any time through the following constraint:

$$\sum_{j \in C_i} (g_{i,j} + y_{i,j}) \leq 1 \ \ \forall (i,j) \in \mathcal{I}. \tag{3.45}$$

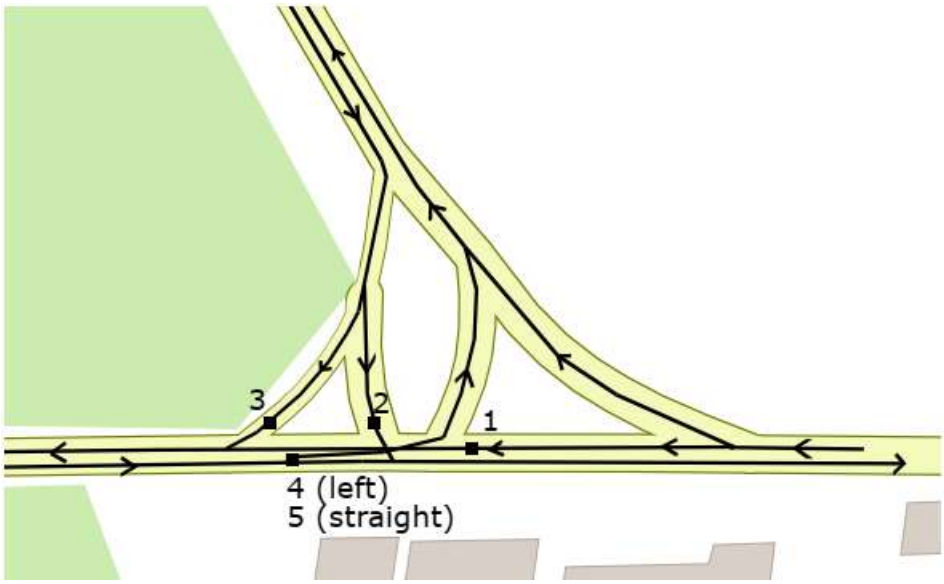

Figure 3.1: Intersection

3.2.5 Objective Function

The proposed TL control aims to minimize and balance the drivers' waiting time at the TL. The following choice of target function reflects this goal:

$$J_N = \sum_{i=0}^{N-1} \sum_{j=1}^{L} (M s_{i,j}) + \sum_{i=0}^{N-1} \sum_{j=1}^{L} \sum_{v \in V_{i,j}} (w_{i,j,v}^2), \tag{3.46}$$

where the term $M s_{i,j}$ is to minimize the slack variable of constraint (number of vehicles) and M is a large positive number. The above is a mixed-integer quadratic programming problem (quadratic objective and linear constraints) for which efficient solution algorithms and solver packages (both commercial and free) are available nowadays.

As previously discussed, one of the most critical components of the *MPC*-based approach is the model used to predict the future dynamics of the traffic flow. One of the main features of the present method is the model used in the *MPC* framework. This is an accurate discrete non-linear system describing the queue dynamics in a junction; however, adding additional auxiliary variables allows the model to keep a **linear formulation**, leading to a *MILP* problem.

3.3 Smart Factory Use Case Modeling in terms of Deep Reinforcement Learning

This section presents the mathematical modelling of smart factory use cases, specifically Industrial assembly lines, in the model-free *RL* approach. The main motivation for adopting the *RL* approaches are follows:

- It's model free; we don't need the exact model of the system as we developed in 3.1, its time taking process.
- To take advantage of *Industrial Data Space (IDS)*, to train the model, and when it is trained, it's deployable in the cross-discipline environment to make it generalised.

3.3.1 Introduction

The *ALBP* instance considered in this work consists in assigning a given set of tasks $\tau \in \mathcal{T}$ and resources $r \in \mathcal{R}$ to a given set of workstations $w \in \mathcal{W}$. The objective of the proposed control algorithm is to minimize the total makespan (i.e., the time required to finish all the tasks) while respecting all the constraints.

3.3.2 Common Terminologies and Description

This section describes the mathematical definition of tasks, resources, constraints, state space, action space and reward function of the *RL*.

Tasks, Resources and Workstation descriptions

1. Each task $\tau_j \in \mathcal{T}$ is characterized in terms of:
   - the first discrete time instant in which it can be processed S_{τ_j}
   - the latest possible discrete time instant in which it must be completed F_{τ_j}
   - the number of discrete time steps required to process it d_{τ_j}.
2. The set of resources required to execute a given task τ_j is denoted with R_{τ_j}:
   - Resources can either be *consumables* (e.g., energy, water, chemicals, raw materials)
   - *instruments* (e.g., tools, operators, workers with specific).
3. Each workstation $w_i \in$ is characterized in terms of its processing capabilities:
   - in terms of the maximum number of tasks that can process in parallel $\bar{W}_i$

Constraints

The considered constraints can be grouped into three classes.

1. *Task constraints*, $\mathcal{T}_\mathcal{C}$, these types of constraints are referred to as Finish to Start (FS), Start to Finish (SF), Finish to Finish (FF) and Start to Finish (SF) constraints

- the latest allowed start time S_{τ_j} and finish time F_{τ_j} for each task τ_j
- dependence relations between tasks
- precedence constraints allow capturing the fact that a given task τ_{j_1} must be finished before a given task τ_{j_2} can be processed
- it must start before a given task τ_{j_2} begins or that (iii) it must be finished before a given task τ_{j_2} can be finished
- it must start before a given task τ_{j_2} can be finished

2. *Workstation constraints*:

   - specifying, for each workstation $w_i \in \mathcal{W}$, (i) which tasks and (ii) the maximum number of tasks that can be processed in parallel

3. *Resource constraints*

   - specifies the type and quantity of resources needed to execute a given task τ_j.

3.3.3 Environment Modeling in terms of Markov Decision Process

MDP represent a practical mathematical framework for formalizing *RL* problems. Indeed, *MDP* model decision processes in which part of the underlying dynamics is stochastic while part depends on the control actions taken by an agent. More precisely, *MDP* are defined using the tuple $\langle \mathcal{S}, \mathcal{A}, P_a, R_a, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P_a(s, s')$ are the transition probabilities specifying the probability of ending in-state s' when, in the state, s, action a is selected, $R_a(s, s')$ is the expected reward for taking action a and going from state s to state s' and, finally, $\gamma \in [0, 1]$ is the discount factor weighting future and past rewards.

The objective of MDPs is to identify the optimal policy π^*, i.e., a function mapping states to actions, maximizing the cumulative reward function R_a. Given the *MDP* framework, defining an *RL* problem is straightforward. Indeed, at each discrete time instant k, the RL agent observes the environment's state $s[k] \in \mathcal{S}$. Based on such observation, the agent interacts with the environment by acting $a[k] \in \mathcal{A}$. As an effect of action $a[k]$, the environment evolves in state $s'[k] \in \mathcal{S}$. The agent can evaluate the effects of said action and state transition through the reward function $R_a(s, s')$. The goal of the *RL* agent is thus to learn the optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ maximizing the cumulative reward

$$R[k] = \sum_{k=0}^{\infty} \gamma^k r[k] \tag{3.47}$$

The considered *ALBP* instance envisages the presence of several hard constraints. As further described in the following sections, a feasibility check addresses this issue before implementing the control actions. If the control action $a[k]$ proposed by the agent at time k is unfeasible (i.e., if it violates at least one constraint $c \in \{\mathcal{T}_C; \mathcal{W}_C; \mathcal{R}_C\}$), the agent obtains a negative reward and must provide a different action. The number of tentative actions the agent can propose is limited to avoid infinite loops. The following subsections will formalise the considered *ALBP* instance as a *MDP*. Thus, all the tuple $\langle \mathcal{S}, \mathcal{A}, P_a, R_a, \gamma \rangle$ will be instantiated for the considered scenario.

3.3.4 State, Action Spaces, Reward and Policy Definition

State space

To capture the dynamics of an assembly line, the state of the environment is defined as

$$\mathcal{S}[k] = \mathcal{S}^{\mathcal{W}}[k] \times \mathcal{S}^{\mathcal{T}}[k] \times \mathcal{S}^{\mathcal{R}}[k] \tag{3.48}$$

where

- $\mathcal{S}^{\mathcal{W}}[k]$ captures workstations' states: the generic element $s_i^{\mathcal{W}}[k] \in \mathcal{S}^{\mathcal{W}}[k]$ represents the state of the i-th workstation defined as the number of tasks which are in execution at time k;
- $\mathcal{S}^{\mathcal{T}}[k]$ captures tasks' states: the generic element $s_j^{\mathcal{T}}[k] \in \mathcal{S}^{\mathcal{T}}[k]$ represents the state of the j-th task defined as the number of discrete time instants $d_j[k]$ left for the task to be finished;
- $\mathcal{S}^{\mathcal{R}}[k]$ captures resources' states: the generic element $s_{i,r}^{\mathcal{R}}[k] \in \mathcal{S}^{\mathcal{R}}[k]$ represents the state of resource of type r concerning the i-th workstation which is defined as the number of resource units available at the workstation at time k.

Action space

The agent's control actions concern assigning tasks and resources to workstations. Hence, the action space $\mathcal{A}$ can be defined as

$$\mathcal{A}[k] = \mathcal{A}^{\mathcal{T}}[k] \times \mathcal{A}^{\mathcal{R}}[k], \tag{3.49}$$

where

- $\mathcal{A}^{\mathcal{T}}[k]$ represents tasks control actions: the generic element $a_{i,j}^{\mathcal{T}}[k] \in \mathcal{A}^{\mathcal{T}}$ specifies if task τ_j has been assigned to workstation i at time k ($a_{i,j}^{\mathcal{T}}[k] = 1$), or not ($a_{i,j}^{\mathcal{T}}[k] = 0$), and
- $\mathcal{A}^{\mathcal{R}}[k]$ represents resources control actions: the generic element $a_{i,r}^{\mathcal{R}}[k] \in \mathcal{A}^{\mathcal{R}}$ specifies if a unit of resource of type r has been assigned to workstation i at time k ($a_{i,r}^{\mathcal{R}}[k] = 1$), or not ($a_{i,r}^{\mathcal{R}}[k] = 0$).

Reward Function

The proposed $ALBP$ algorithm's primary goal is to minimise the total makespan C_{max}, i.e., the time required to finish all tasks $\tau_j \in$. To achieve this goal, we selected the following reward function:

$$R_a(s, s') = \begin{cases} 2, & \text{if } a \text{ is feasible and } \sum_{i=1}^{|\mathcal{W}|} \sum_{j=1}^{|\mathcal{T}|} a_{i,j}^{\mathcal{T}}[k] > 0 \\ 0, & \text{if } a \text{ is feasible and } \sum_{i=1}^{|\mathcal{W}|} \sum_{j=1}^{|\mathcal{T}|} a_{i,j}^{\mathcal{T}}[k] = 0 \\ -2, & \text{if } a \text{ violates at least one constraint} \end{cases} \tag{3.50}$$

Note that the agent is encouraged to assign tasks to workstations with this modelling choice. Indeed, $\sum_{i=1}^{|\mathcal{W}|} \sum_{j=1}^{|\mathcal{T}|} a_{i,j}^{\mathcal{T}}[k] > 0$ if and only if at least one task has been assigned. A zero reward is returned to the agent when no task is assigned. Finally, when the selected action violates constraints, a negative reward is given to discourage this behaviour. The behaviour induced by this reward function aligns with the main goal we aim to achieve, i.e., minimising the total time to complete

the tasks. The total completion time is minimized when the workstations work tasks as much as possible. It will be verified via simulations that this reward function leads to the desired objective.

Policy Function

A typical choice for *RL* applications is an ϵ-greedy policy based on the Q-value Function. Said function allows to evaluate of the quality of state-action pairs under a given policy π and is defined as:

$$Q_\pi(s,a) = \left[\sum_{t=0}^{\infty} \gamma^t r_{k+t+1} | s[k] = s, a[k] = a\right] \tag{3.51}$$

Where γ is the discount factor mentioned in the previous subsection. In other words, the Q-value function measures the rewards obtained when k the state is $s[k]$; the selected action is $a[k]$, and the adopted policy is π. As further detailed in the next section, an Artificial Neural Network (ANN) will approximate the Q-value function in this work. The ϵ-greedy policy exploiting the Q-value function can be defined in the following way:

$$\pi(s,a) = \begin{cases} Q^\pi(s,a), \text{ with probability } 1-\epsilon \\ \text{Random action, with probability } \epsilon. \end{cases} \tag{3.52}$$

ϵ-greedy policies allow modulating exploration (of unknown state-action pairs) and exploitation (of acquired knowledge about given state-action pairs). ϵ-greedy policies allow modulating exploration (of unknown state-action pairs) and exploitation (of acquired knowledge about given state-action pairs) phases. Balancing these two phases is of paramount relevance for achieving good performance. Ideally, exploration should be encouraged at the beginning of the training phase since the agent received only feedback concerning a few state-action pairs. To guarantee exploration, the ϵ value should be high. On the contrary, when a significant amount of interactions between the agent and environment have occurred, it is reasonable to have small values for ϵ to exploit the acquired knowledge. It is common to let ϵ change according to the number of training episodes to achieve these conflicting objectives. Let E be the total number of training episodes, then ϵ can be defined as

$$\epsilon(i) = e^{\frac{i}{\alpha E}}. \tag{3.53}$$

where α is a positive constant and $i = 1, ..., E$.

3.3.5 Proposed DRL Control Algorithm and Agents' Training

DRL techniques adopt the adoption of *NN* as function approximators of value and/or policy functions. In this work, a *DQN* algorithm [46] and a parallel training approach similar to what was done in [66] have been adopted. Training is performed over a given number of episodes. An episode is the simulation of the complete mapping of all the tasks. During every episode, the agents take actions which modify the environment, resulting in rewards for them (different agents receive, in general, different rewards, depending on the actions they take). The rewards are stored in a database, and at the end of each episode, every agent is trained on the database of training samples collected during the episode, with weights of the *NN* of agents updated. The agent selects a random mini-batch

from memory, and the new episode is started again.

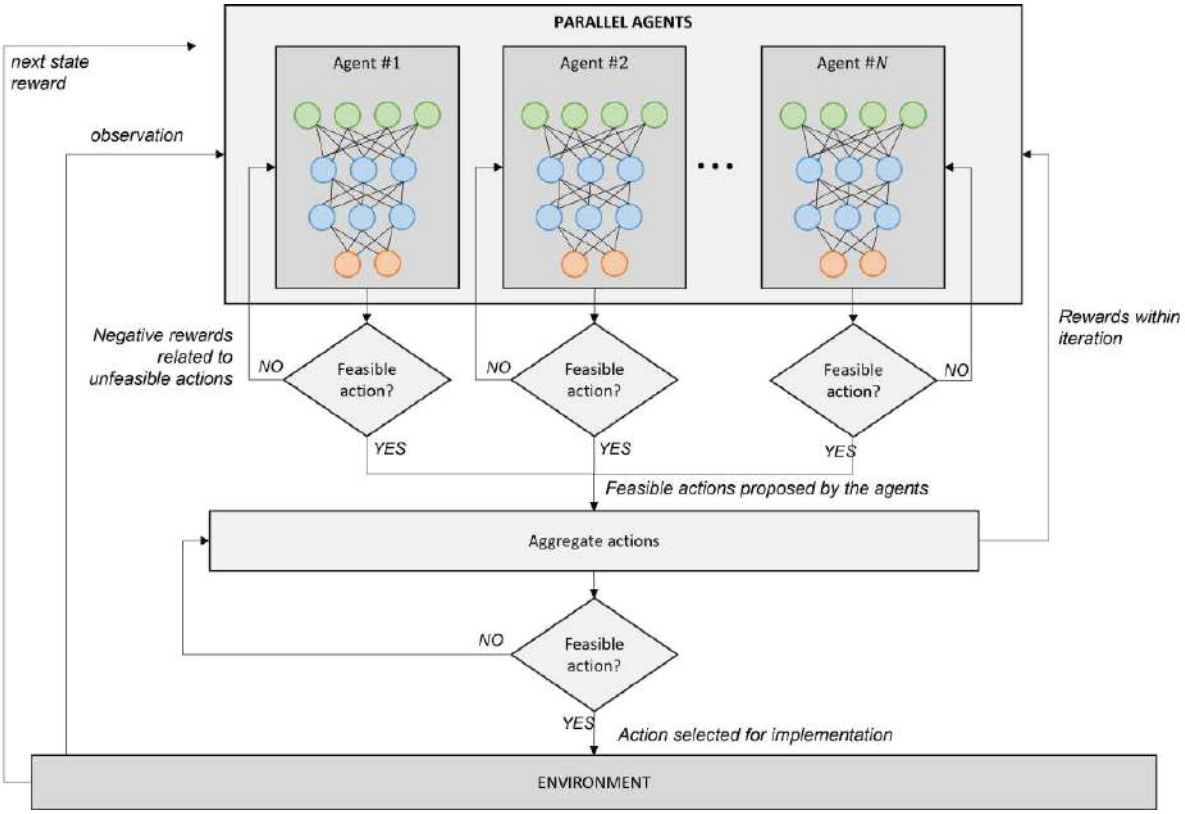

Figure 3.2: Parallel DRL with hard constraints' management.

As depicted in Figure 3.2, at each discrete time k in a given episode, a set of N agents receive an observation of the environment $o[k]$. Based on said observation, each agent proposes a control action $a_i^*[k]$ by following an ϵ-greedy policy as described in equation (3.52). To guarantee that only feasible actions are implemented, each action $a_i^*[k]$ is checked against the set of constraints $\langle \mathcal{T}_C; \mathcal{W}_C; \mathcal{R}_C \rangle$. If such a feasibility check fails, the agents receive a negative reward $\tilde{r}_i$ and suggest a different action. To avoid infinite loops, the number of attempts an agent can perform is limited; if an agent fails to find a feasible solution, then its control action is forced to be *do nothing* (i.e., do not assign tasks nor resources). After each agent has computed its control action $a_i^*[k]$, a central control logic combines them. For this purpose, it is checked if some of the control actions are in conflict. Indeed, two agents may assign the same task to different workstations. In this case, based on an ϵ-greedy policy, the central control logic decides which one of the two control actions can be implemented. Based on the decision of the central control logic, the agents receive an additional reward $\bar{R}$ for encouraging non-conflicting actions. After this second set of feasibility checks, the joint control action $A^*[k]$ is implemented, and a standard reward $r[k+1]$ is provided to the agents.

In the following subsections, the structure of the agents' neural networks is detailed.

Agents' Neural Network Structure

At the generic time k, the state of the environment (defined as in (3.48)) is observed, and the observations are stored in a vector, $S_k^{obs} \in \mathbb{R}^{W+T+WR}$, which is the input of the agents' neural

networks.

$$S_k^{obs} = col(s_k^{w,obs}, s_k^{t,obs}, s_k^{r,obs}) \tag{3.54}$$

Where $s_k^{w,obs}$, $s_k^{t,obs}$, and $s_k^{r,obs}$ are, respectively, the vectors of the current (at time k) observations of the state of the workstations, the state of the tasks, and the state of the resources ($col(\cdot)$ is the operation of the vertical concatenation of vectors).

The vector of the observations of the current state of the workstations is structured as follows:

$$s_k^{w,obs} = col(s_{k,1}^{w,obs}, ..., s_{k,W}^{w,obs}), \tag{3.55}$$

where the generic element $s_{k,w}^{w,obs}$, $w = 1, ..., W$, is a binary vector of length T. The generic element of position t in $s_{k,w}^{w,obs}$ is equal to one if and only if, from the observation, it is determined that task t is currently running at workstation w.

The vector of the observations of the current state of the tasks coincides with $\mathcal{S}^{\mathcal{T}}[k]$, as defined in Section 3.3.4 ($\mathcal{S}^{\mathcal{T}}[k]$ is already a vector).

The vector of the observations of the current state of the resources is structured as:

$$s_k^{r,obs} = col(s_{k,1}^{r,obs}, ..., s_{k,W}^{r,obs}), \tag{3.56}$$

where the generic element $s_{k,w}^{r,obs}$, $w = 1, ..., W$, is a binary vector of length R. The generic element of position r in $s_{k,w}^{r,obs}$ is equal to the amount of resource type r observed at workstation w at time k. In conclusion, the dimension of the input layer (i.e., the number of nodes composing it) is:

$$|S_k^{obs}| = |s_k^{w,obs}| + |s_k^{t,obs}| + |s_k^{r,obs}| = WT + T + WR \tag{3.57}$$

The *NN* output layer contains the actions the *DQN* agent decides. The output layer is structured into two parts, one concerning assigning tasks to workstations and the other concerning assigning resources to workstations. Therefore, the output vector, $A_k^{out} \in \mathbb{R}^{W(T+R)}$, is:

$$A_k^{out} = col(a_k^{t,out}, a_k^{r,out}), \tag{3.58}$$

Where the vector $a_k^{t,out}$ concerning the actions of the current state of the workstations is structured as:

$$a_k^{t,out} = col(a_{k,1}^{t,out}, ..., a_{k,T}^{t,out}), \tag{3.59}$$

where the generic element $a_{k,t}^{t,out}$, $t = 1, ..., T$, is a vector of length W, whose generic entry w relates to the assignment of task t to workstation w. Specifically, the value of the position element w in $s_{k,t}^{w,obs}$ represents the expected reward that the actor gets taking that action at time k and then following the policy in the following times.

Element $a_k^{r,out}$, in (3.58) is built similarly and concerns assigning resources to workstations. Element $a_k^{r,out}$ is structured as:

$$a_k^{r,out} = col(a_{k,1}^{r,out}, ..., a_{k,R}^{r,out}), \tag{3.60}$$

where the generic element $a_{k,r}^{r,out}$, $r = 1, ..., R$, is a vector of length W. The value of the element of position w in $s_{k,r}^{w,obs}$ represents the expected reward that the agent receives if it assigns resource r

to workstation w, and after that follows the policy.

In the next section, we discuss the proposed *DQN* control strategy used to select, at every time k, the actual actions to be implemented from examining the values returned by the *NN* in the output layer.

In conclusion, the dimension of the output layer is:

$$|\mathcal{A}_k^{out}| = |a_k^{t,out}| + |a_k^{r,out}| = WT + WR. \tag{3.61}$$

3.4 Smart Transportation Use Case Modeling in terms of Deep Reinforcement Learning

This section presents the mathematical modelling of smart transportation use cases, specifically traffic signals, in the model-free *RL* approach. The main motivation for adopting the *RL* approaches are follows:

- It's model free; we don't need the exact model of the system as we developed in 3.1, its time taking process.
- To take advantage of *IDS*, to train the model, and when trained, it's deployable in the cross-discipline environment to make it generalised.

3.4.1 Introduction

The present work aims to improve an intersection's performance in traffic flow, average waiting time and emissions by controlling the traffic lights via *DRL*. The learning mechanism used to train the neural network is the*DQN*; therefore, at each instant of time t, the agent will receive a state s_t and a reward r_t, and according to these, it chooses the action a_t to perform at the next time step. The information received about the state s_t as a consequence of the action a_{t-1} taken at s_{t-1}, together with the reward r_t are the knowledge which allows the neural network to learn about how to deal with similar scenarios.

3.4.2 Environment Modeling in terms of Mark of Decision Process

The environment used for simulation in the present work is in Figure *control architecture*. It consists of a multiple-way intersection with five primary arms where each lane has its destination, and it is managed by a traffic light allowing or not the vehicles to pass the junction. Indeed, if a vehicle leaves a lane, this will reach its relatively appropriate destination lane. For example, if the vehicle arrives from lane X, this will undoubtedly reach lane Y. In the present environment, three traffic lights regulate all the junctions; this was just a design approach since each lane has a traffic light. These are indicated with colour at each time step which also defines its state and must follow the correct fixed order: green, yellow, red, green, yellow. All the TL work together according to the junction's rules to avoid collisions; therefore, the TL system may set only a fixed number of configurations. Each of these configurations may be set at any instant except for the yellow state, which must be there for at least a certain amount of time.

Vehicle Generation

The generation of the vehicles is modelled following a Poisson Distribution, a discrete distribution providing the probability of occurrence of an event in a given period. From a view inspection, an arrival rate, vehicle per second, was assigned to each lane. Moreover, three different traffic intensities are considered; hence, three arrival rates are assigned at each lane, one for each traffic intensity condition.

3.4.3 State, Action Spaces, Reward and Policy definition

State Space

State space is a primary source of information for the agent, and plenty of sensors are available to capture meaningful information from state space that needs to be provided to improve its knowledge. We use a realistic simulation framework *Simulation of Urban Mobility (SUMO)* for each lane to gather information at each time step which consists of real numbers. The number of halting cars and cumulative waiting time provide information on the current intersection status, and the average arrival rate should help the network to know also about future scenarios. The state S_t is represented at each time step as follows:

$$S_t = N_{hv}^t \times CW^t \times R_{av}^t \tag{3.62}$$

Where N_{hv}^t is the number of halting cars at the TL, CW^t cumulative waiting time of the junction and R_{av}^t average arrival rate of each lane, the collected information is passed as the input to the neural networks. Agents select future actions from the output layer of the neural network following some policies.

Action Space

RL agent interacts with the environment by selecting an action from the action space by following policy π. The present action space is a combination of traffic lights with changing patterns. The yellow light is considered safe; every green light will be followed by a yellow light lasting for 5 seconds to allow the agent to choose only feasible actions. We divided the set of actions into two groups $1, 2, 3$ gree and 4,5,6 are yellow

- Green configurations and Yellow configurations :
    1. TL1 and TL5 green, TL2, TL3 and TL4 red
    2. TL2 and TL3 green, TL1, TL4 and TL5 red
    3. TL3, TL4 and TL5 green, TL1 and TL2 red
    4. TL1 and TL5 yellow, TL2, TL3 and TL4 red
    5. TL2 and TL3 yellow, TL1, TL4 and TL5 red
    6. TL3, TL4 and TL5 yellow, TL1 and TL2 red

At each time step t, the agent chooses an action a_t at given state s_t.

1. if a_t is is same as a_{t-1} at time step $t-1$, then the yellow configuration will remain same

2. if a_t differs from a_{t-1}, the appropriate action within the yellow group will be performed for 5 seconds

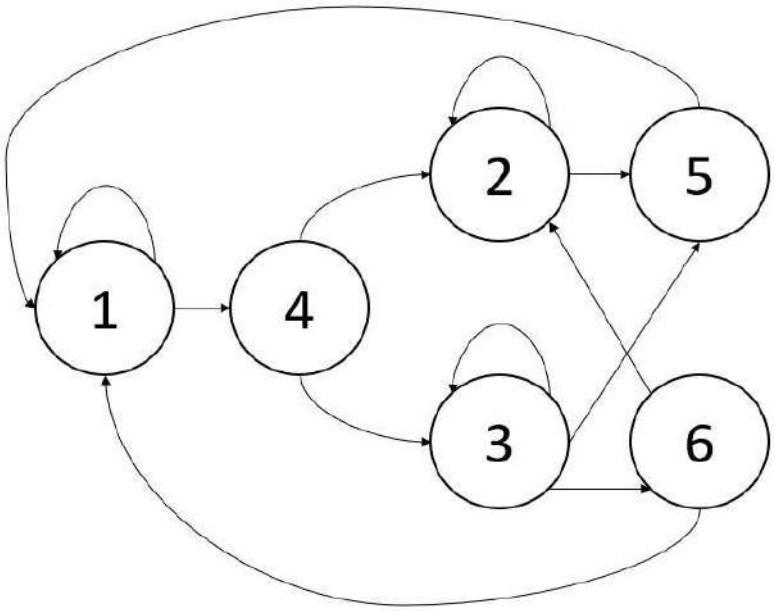

Figure 3.3: Graphically representation of action selection

These actions are executed in the environment, and their effectiveness is measured regarding reward collection.

Reward Formulation

Reward formulation in *RL* is a challenging task, and, most of the time, *KPI* and constraints of the problem are intuitive for its design. In the present problem, the objective is to reduce the number of vehicles in the junction. The waiting time is one of the significant parameters, and the sum of the *Cumulative Waiting Time (CWT)* of each vehicle in the junction is used as a reward formulation which is computed as:

$$CWT = \sum_{v=1}^{N} WT(v) \tag{3.63}$$

The difference between two computed CWT at time step t and $t-1$ and based on its value, positive or negative reward is assigned as follows:

$$R_a(s, s') = \begin{cases} +ve, & \text{if } CWT(t-1) - CWT(t) > 0 \\ -ve, & \text{if } CWT(t-1) - CWT(t) < 0 \end{cases} \tag{3.64}$$

Where $CWT(t)$ and $CWT(t-1)$ are the cumulative waiting time of each vehicle v at the current and previous time steps, respectively.

- If $CWT(t-1) - CWT(t) > 0$, it means that $CWT(t) < CWT(t-1)$ and cumulative waiting time at the next time step t has reduced, and reward is positive.
- If $CWT(t-1) - CWT(t) < 0$, it means that $CWT(t) > CWT(t-1)$ and so that the performed action was not optimal. Indeed, the reward will be negative and used to penalize the agent for future actions.

To conclude, in this section, we define the essential components of *RL*, and the next phase is dedicated to the training cycle of the framework.

3.4.4 Agent training

This section describes how above mention components work together. In particular, at each time step t, the agent first retrieves the environment state regarding the time delay at time steps t and $t-1$ and computes the reward associated with the action performed at $t-1$. All this information is saved into a memory used in the experience replay technique and helpful to the agent for setting the new action. We trained the agent through a traffic simulator, which recovers the state information from the simulation environment at each step. The training intensity may change based on the sampling strategy used; the first strategy is more straightforward and consists in initiating the training phase at every *time step*; a second possibility, and also the most intensive, is of initiating the training phase at every *simulation step* instead. In this work, only the latter is used to obtain more stable results.

The following section gives a detailed explanation of the training process.

Training Cycle

The following steps are performed every time a training instance of the agent is initiated:

- A sample $m = s_t, a_t, r_{t+1}, s_{t+1}$ is added to memory

- A fixed number of samples, depending on the sampling strategy used, are picked randomly from memory, constituting the batch B

Every sample $b_k \in B$ holds the starting state s_t, the action performed at a_t, the consequent reward r_t received after taking action a_t from state s_t and the following state s_{t+1}. Now, the following operations are performed for each of these samples:

1. Once the row vector representing s_t is provided to the neural network and therefore the predicted Q-value, relative to action a_t, are obtained, the Q-value $Q(s_t, a_t)$ is computed

2. One the row vector representing s_{t+1} is provided to the neural network. Therefore, the predicted Q-values, relative to action a_{t+1} are obtained, the Q-value $Q'(s_{t+1}, a_{t+1})$ is computed.

3. The Q-value is updated through Equation 3.65. Among all the possible Q-values computed at the previous step, the best one is selected through the $max_a Q'$ term, representing the *maximum* expected future reward. Furthermore, this term is discounted by a factor γ, which gives more or less importance to the immediate reward.

$$Q(s_t, a_t) = r_{t+1} + \gamma max_a Q'(s_{t+1}, a_{t+1}) \tag{3.65}$$

4. At this point, the neural network is trained for a single sample. As shown in the Figure above, the neural network's output will be the desired Q-value $Q(s_t, a_t)$, which is now the maximum expected future reward thanks to Equation 3.65.

The departure rate follows the car model used in the simulation.

Chapter 4

Numerical Results

This chapter presents the number of experiments and related results to check the effectiveness of the developed framework. At first, the description of smart manufacturing, specifically *ALBP* uses-cases from superficial to complex scenarios, is presented. Further results are compared with baseline control and learning-based techniques considering the complexity, scalability, and real-time deployability factor. The second half presents descriptions of the city's logistics, specifically *TSC* Use-case. The effectiveness of the Control and learning-based module of the developed framework is tested against a real-time Junction in Rome, Italy. Finally, a comprehensive comparison of the results of the proposed framework with baseline controller, heuristics and learning-based methods is discussed.

4.1 Smart Manufacturing Use-case Testing by Control Technique

This section tested the *MPC* based unit of developed control and data-driven framework. The key goal is to make sure that the model developed can correctly capture and represent an assembly problem, with its complexity given by the interaction of actors, facilities, resources, scheduling constraints, etc.

4.1.1 Use-case Description

In the following section, three simulation scenarios are presented. At first, a limited number of workstations, tasks and resources are considered. This low-complexity simulation aims to show that the algorithms work as expected. A complex simulation involves many workstations and hundreds of tasks, spanning a long time horizon and based on real scenarios and data from the industrial partner. This simulation aims to test the algorithm on highly complex scenarios and have a first check of the scalability in terms of memory allocation and time complexity; Finally, a third scenario is complex and aims to capture a realistic scenario of assembly operations at the Kourou Launch base, focusing on the Vega mobile gantry.

The simulation scenario is as follows. We consider an assembly line with three workstations and a job of two tasks. Figure 4.2 reports the topology of the paths linking the workstations and the initial state of the workstations' resource buffers (i.e., the distribution of the resources in the plant). Figure 4.1 reports the task dependency graph, the inputs needed to execute each task, and the outputs produced by the tasks.

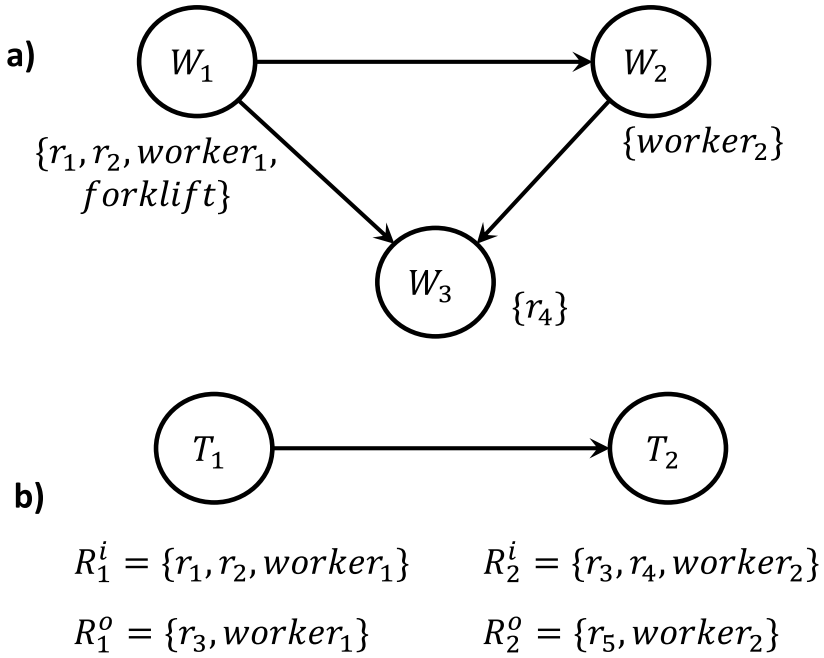

Figure 4.1: Workstations (a) and tasks (b).

4.1.2 Experimental Setup

Simulations are performed using the open-source technical computing language Julia, version 1.3.1 (https://julialang.org/). The optimization problem is written using the Julia JuMP modelling package [19] and solved using Gurobi optimizer [26] on an Intel I7, 8GB RAM machine running Windows 10.

4.1.3 Algorithmic Validation

The aim is to ensure the algorithms work as expected from the problem conception and modelling stage. A key goal is to make sure that the model developed can correctly capture and represent an assembly problem, with its complexity given by the interaction of actors, facilities, resources, scheduling constraints, etc. Accurate checks and simple proof of concept simulations were used for debugging the algorithm and code developed for the simulations and ensuring it worked as expected.

The following simulations concern a straightforward artificial scenario with minimal workstations, tasks and resources spanning a concise time horizon. They are meant to provide a first validation of the complete algorithm; on a scenario whose complexity is easily manageable. This is very important to check that the algorithm is correct and behaves as expected. This is the first step before scaling the validation via simulation on a more complex scenario. These simulations have been presented in an accepted conference paper [42].

4.1.4 Simulation Results

The simulation is on a time horizon of 20-time steps. The tasks last 3-time steps and can start at any time or workstation (provided that the input resources are available at that time/workstation). One vehicle is present in the scenario. A constant velocity is assumed for it. Figure 4.3 reports the tasks' state: task 1 is immediately executed since it has all the inputs available at the workstation.

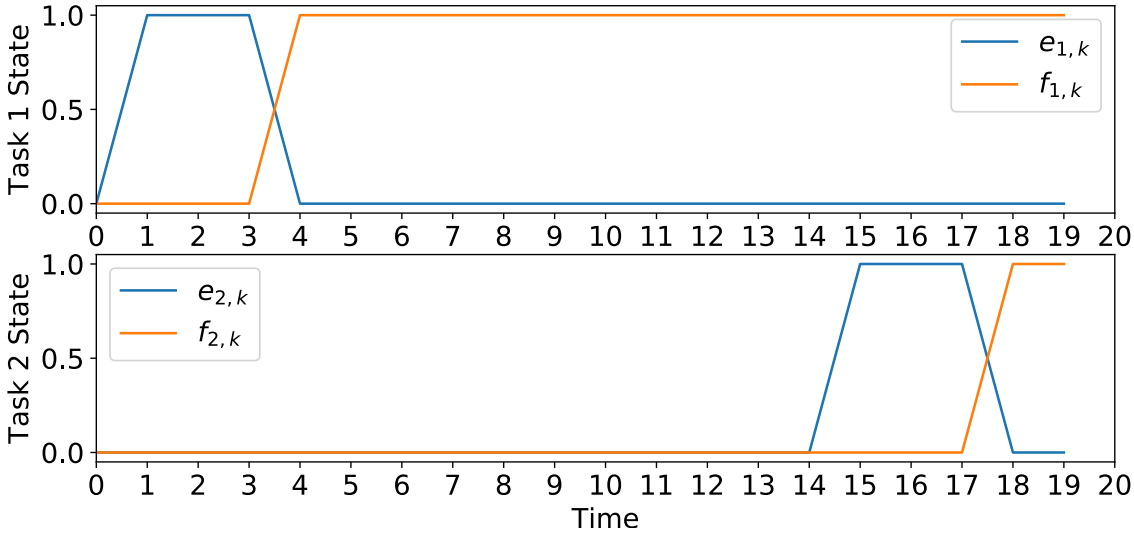

Figure 4.2: Tasks' state.

Task 2 instead needs resource 3 (generated at workstation 1 as the output of task 1), worker 2, who is at workstation 2, and r4, which is at workstation 3.

We present a simple simulation to provide proof of the concept. The analysis of a more complex scenario is shown in the next section. The simulation scenario is as follows. We consider an assembly line with three workstations and a job of two tasks, as illustrated in Fig. 4.1. Figure 4.1.a report on the topology of the paths linking the workstations and the initial state of the workstations' resource buffers (i.e., the distribution of the resources in the plant). Figure 4.1.b reports the task dependency graph, the inputs needed to execute each task, and the outputs produced by the tasks.

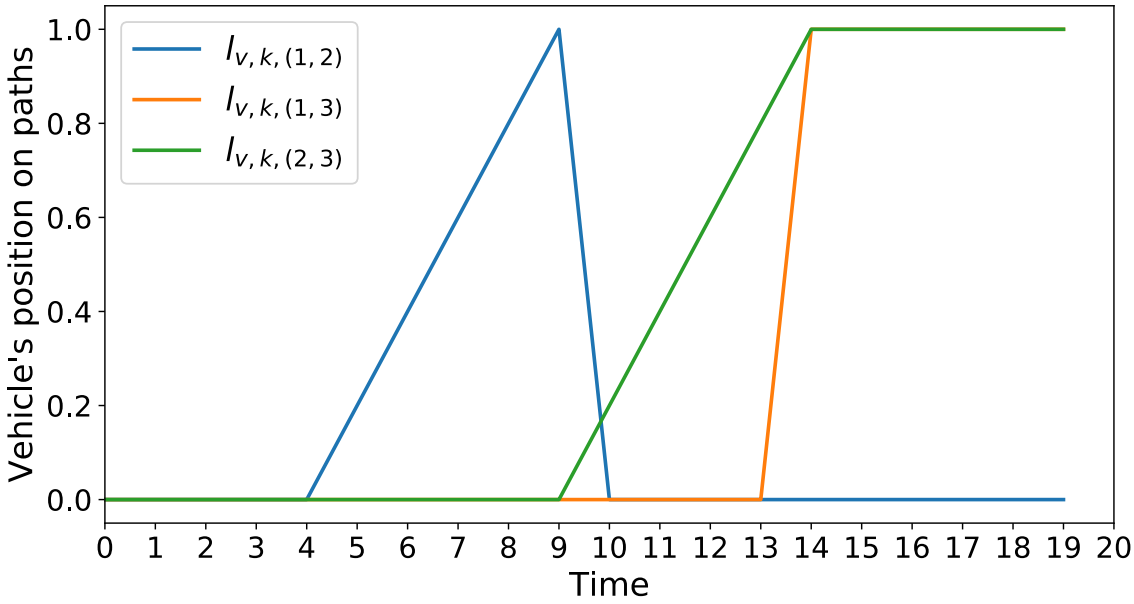

Figure 4.3: Vehicle's movements.

The simulation is on a time horizon of 20 time steps. The task's last 3 time steps can start at any time and at any workstation (provided that the input resources are available at that time/-workstation). One vehicle is present in the scenario. A constant velocity is assumed for it. Figure 4.2 reports the tasks' state: task 1 is immediately executed since it has available at w_1 all the needed inputs. Task 2 needs r_3 (generated at w_1 as the output of task 1), worker 2, which is at w_2, and r_4, which is at w_3. Figure 4.3 reports the vehicle's movements (i.e., its location along the paths connecting the workstations). After r_3 is generated, the vehicle carries it from w_1 to w_2 (i.e., $l_{v,k,(1,2)}$ grows from 0 to 1). Once at w_2, the vehicle also carries worker two and brings both r_3 and worker two at w_3, where all the resources are available to start task 2. Notice that when the vehicle is at a workstation, it is also at the end of all the paths in coming into that workstation (e.g., when the vehicle is at w_3, both $l_{v,k,(1,3)}$ and $l_{v,k,(2,3)}$ are equal to one). Figure 4.4 reports the location of resources in time (w_{bar} is a fictitious workstation that stores the resources still to be created and the consumed ones). It is seen that the location of resources is consistent with that of the vehicle and with task execution. Notice, for example, that r_3 is created at time 4 at w_1, then at time 9 it reaches w_2, then at time 14 it finally arrives at w_3, where it is consumed by task 2. The solution found was the one minimizing the job finish time. Simulation time was 10.75 seconds. There are 3043 variables in the model.

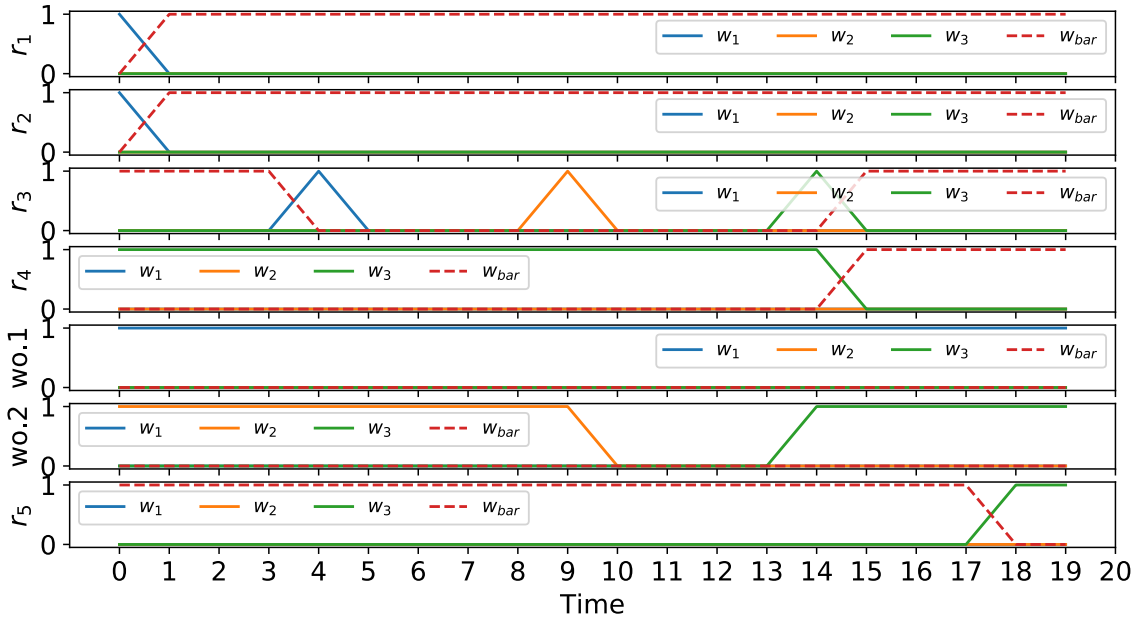

Figure 4.4: Resources' movements.

4.2 Smart manufacturing Use-case Testing by Data-driven Technique

This section tested the *DRL* based unit of developed control and data-driven framework. The key goal is to make sure that the model developed can correctly capture and represent an assembly problem, with its complexity given by the interaction of actors, facilities, resources, scheduling constraints, etc.

4.2.1 Use-case Description

In this section, we present numerical simulations to validate the proposed method. We propose a set of simulations of increasing complexity as follows:

1. Scenario 1. We evaluate the performance of the algorithm for the problem of scheduling a set of tasks on a given number of workstations, considering the following constraints: the precedence constraints among the tasks must be respected, all tasks must finish before a given deadline, and workstations can work any of the tasks, but only one at any given time;

2. Scenario 2. It is the same as Scenario 1, where in addition also, constraints on resources needed by the tasks to execute are considered. In addition, the algorithm needs to provide an optimized schedule of resource assignment to the workstations, aiming at minimizing the assignment of resources (i.e., the algorithm should assign only the needed resources when needed).

In every scenario, we consider a batch of 15 tasks scheduled on two workstations. The precedence constraints among the tasks are displayed in Fig. 4.5 (tasks not displayed do not have precedence constraints). Without loss of generality, we consider all dependencies to be of the finish-to-start type, the most common one. Hence, in the graph, an arrow from a generic task τ_i to a generic task τ_j means that τ_i must finish before τ_j can start.

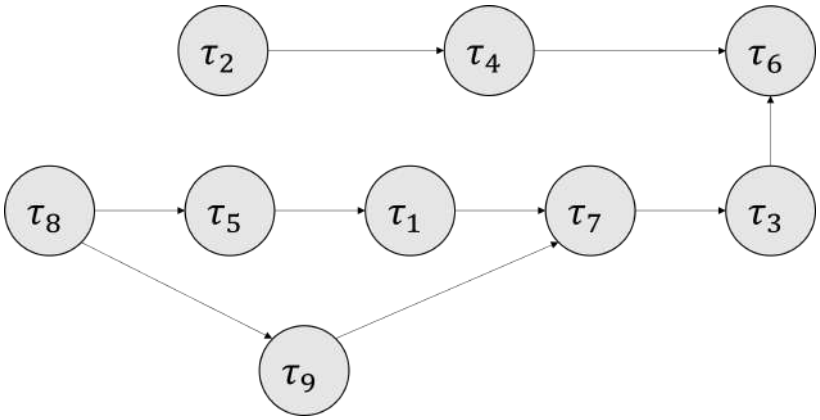

Figure 4.5: Precedence constraints for task execution.

4.2.2 Experimental Setup

Simulations are performed on an Intel(R) Core(TM) i5-10210U CPU @ 2.11 GHz, 16.0 GB RAM, and an x64-based processor at Windows 10. The simulation environment has been coded in Python, using PyTorch libraries to implement the deep *RL* agent. The tool openAI gym is used to develop the assembly line environment [51]. For the deep *RL* agents, we considered the following parameters:

- The NN implemented inside each *DQN* agent is a dense two-layer network with 24 neurons in each layer, with re-LU activation functions;
- Adam optimizer is used [52];
- The learning rate is $\alpha = 0.001$;
- The discount factor is $\gamma = 0.9$;
- A memory size of 2000 samples is selected;
- The mini-batch size is 256 samples;
- Learning is performed over 10000 episodes;

4.2.3 Training

Figure 4.6 displays how the parameter ϵ varies episode after episode, following (3.53). In the first episodes, high values of ϵ encourage exploration. As the agent learns the optimal policy, ϵ decreases to favour the exploitation of the acquired knowledge.

4.2.4 Simulation Results

In this section, report the results of individual scenarios as follows:

Scenario 1

Figure 4.7 illustrates the total reward collected from the environment at every episode. The agents have trained over 10,000 episodes; simulation results are described below. From Fig. 4.7, we can

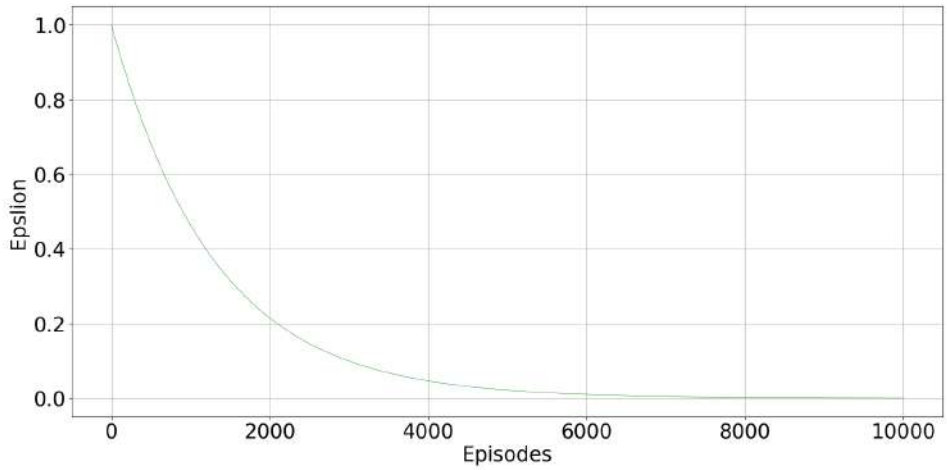

Figure 4.6: Plot of ϵ parameter.

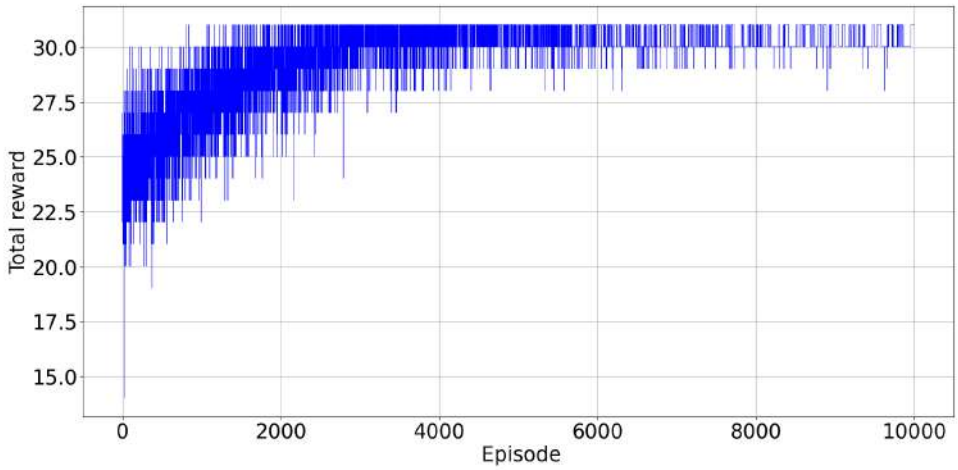

Figure 4.7: Scenario 1: Total observed reward at each training episode.

empirically check that the proposed DQN algorithm converges after about 6000 episodes of training. The training process over 10,000 episodes took 2460 seconds.

Figure 4.8 displays the evolution of the total make-span observed in every episode. As the learning progresses, the make-span decreases and converges to the minimum value of 48-time intervals. In Fig. 4.8, there are spikes in the made-span's observed value at some points when the agents explore random actions. Exploration becomes less and less frequent from one episode to the other because ϵ, the exploration rate, decreases as reported in Fig. 4.6. We can see also from Fig. 4.8 that in some early episodes, the observed make-span is lower than the optimal one found after learning has converged. This is because the stopping criterion on the maximum possible number of iterations within a single episode is reached in these early episodes. The episode is terminated before all the tasks are assigned, so the observed make-span refers only to a portion of the total number of tasks.

The Gantt chart in Fig. 4.9 and Fig. 4.10 illustrate, respectively, the placement of resources, and the scheduling of tasks on a workstation, as suggested by the trained agents.

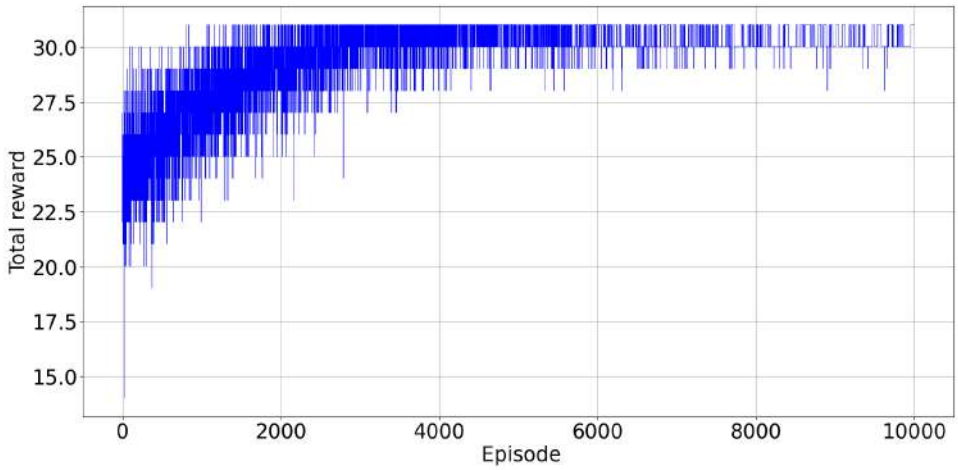

Figure 4.8: Scenario 1: Observed make-span at each training episode.

Figure 4.9: Scenario 1: level of resource at the workstations.

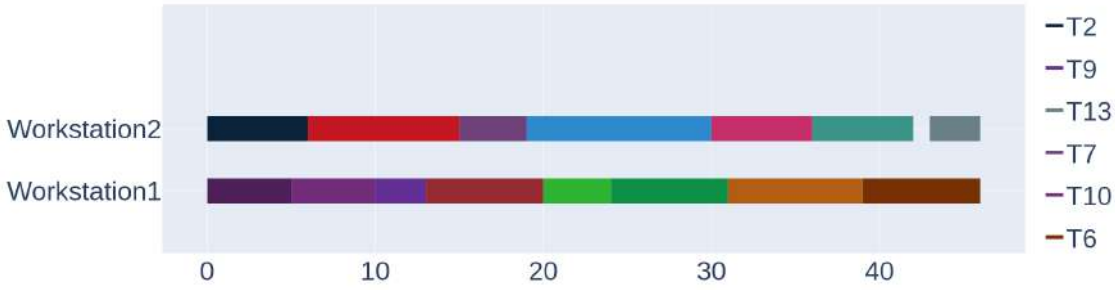

Figure 4.10: Scenario 1: Tasks' assignment - trained agents.

It is possible to check from the Gantt charts that there is no feasible alternative plan of resources and tasks assignment that results in a lower makespan, which confirms that the agents have learned the optimal scheduling policy for the problem.

Scenario 2

In this scenario, also resource constraints are considered, in addition to the ones already considered in Scenario 1. The agents are again trained over 10,000 episodes. Figure 4.11 displays the total reward collected from the environment, and Fig. 4.12 illustrates the total make-span at every episode. In Fig. 4.12, there are spikes in the made-span's observed value at some points when the agents explore random actions. Exploration becomes less and less frequent from one episode to the other because ϵ, the exploration rate, decreases as reported in Fig. 4.6.

We can see in some early episodes the observed make-span is again lower than the optimal one found after learning has converged for the same reasons as above.

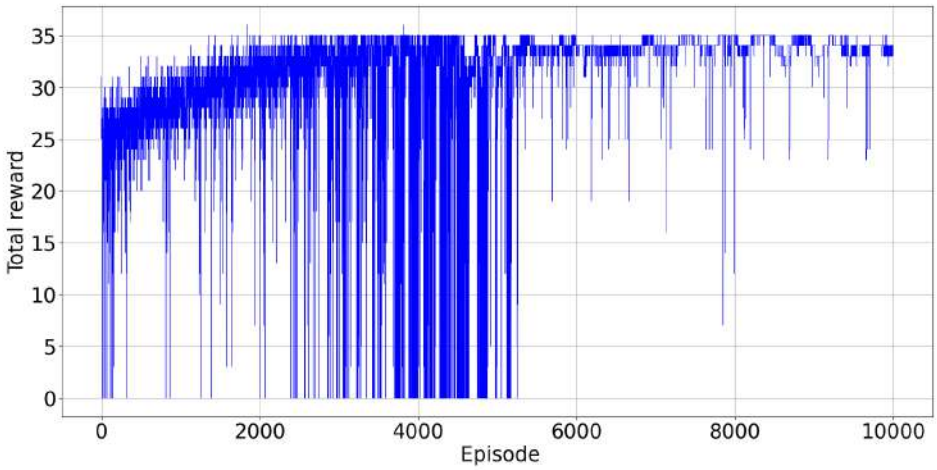

Figure 4.11: Scenario 2: Total observed reward at each training episode.

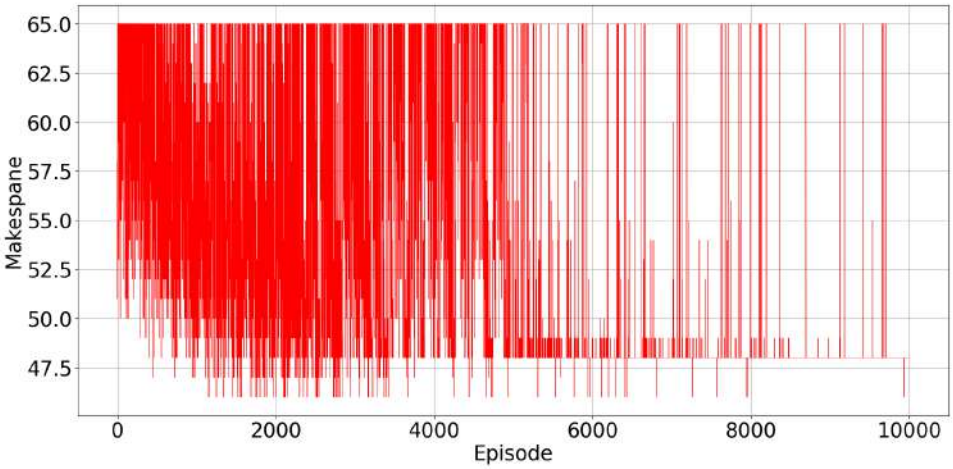

Figure 4.12: Scenario 2: Observed make-span at each training episode.

Figures 4.13 and 4.14 report the optimal assignment of resources and tasks to workstations.

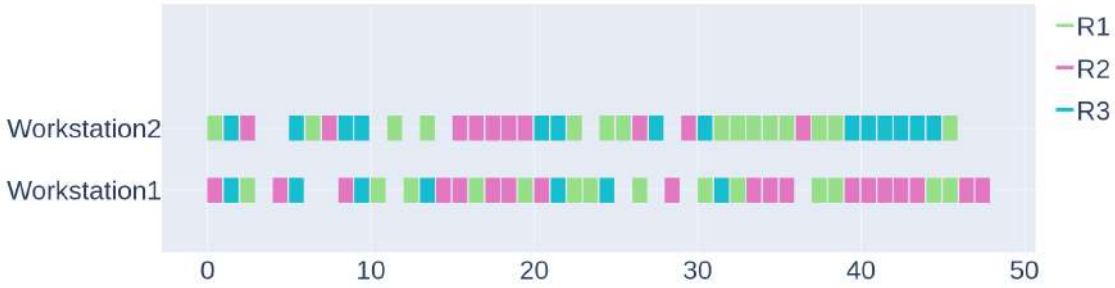

Figure 4.13: Scenario 2: level of resource at the workstations

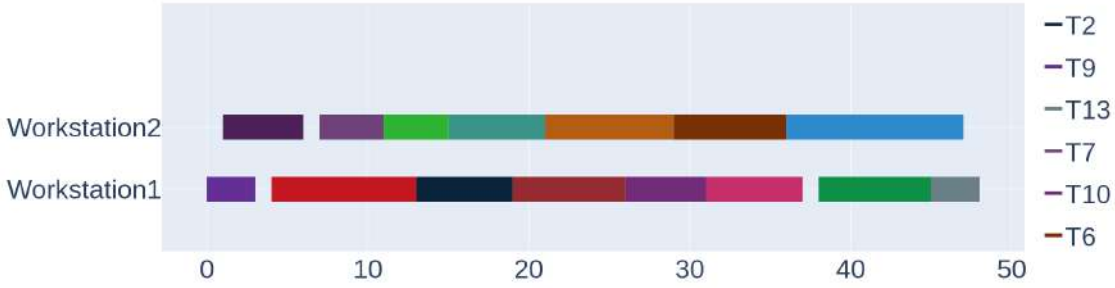

Figure 4.14: Scenario 2: Tasks' assignment - trained agents.

4.3 Comparison with State-of-the-art Approaches

4.3.1 Heuristic Approach's

We compare the proposed algorithm with a heuristic solution for task scheduling and with an optimization-based algorithm that implements a *MPC* scheme for task control. These are two popular solutions in assembly line control. The comparison is performed based on Scenario 2 above. The leading key performance indicators of interest for the comparison are:

- The total time to complete the tasks (cycle time). This gives a measure of how good the algorithm is in optimizing the tasks' execution;
- The execution time of the algorithm. This tells us how fast the algorithm is in computing the task scheduling. This is important, especially when there is the need to update the scheduling in real-time (e.g., due to faults).

4.3.2 Control Approach's

In this section, we present the *MPC* based algorithm proposed in [42], which controls allocating tasks and resources to the workstations in discrete time, with a sampling time of T seconds.

The *MPC* logic foresees that every T seconds, an optimization problem is built and solved to decide the best allocation of resources and tasks to workstations while respecting all the applicable constraints. The target function of the *MPC* algorithm is designed to minimise the make-Span.

The *MPC* algorithm has been implemented in Julia 1.7 (https://julialang.org/), and the optimization problem has been solved using the solver Gurobi (https://www.gurobi.com/). The average detected solving time of the *MPC* algorithm was 21.2 seconds, which compares with an almost instantaneous running time (0.046 seconds) of the trained *DRL* agents. This gap is expected to be much more prominent in more significant scenarios, with more tasks, resources and workstations. Also, the solving time of the *MPC* algorithm increases further when the sampling time is decreased (in this simulation, we considered for the *MPC* algorithm a sampling time $T = 15$ minutes). Due to its almost instantaneous speed, the proposed *DRL* approach is more usable in real-time applications, for example, when there is the need to re-schedule tasks in real-time. The total time to complete the tasks, resulting from the *MPC* algorithm, is 46 time steps (compared to a cycle time of 48 time steps resulting from the proposed *DRL* approach).

4.3.3 Shortest Processing Time Heuristic

In the following, Scenario 2 is solved using the *Shortest Processing Time (SPT)* heuristic, often used to derive simple and effective task scheduling. This heuristic gives precedence in the execution of the tasks having the shortest processing time. The assumptions used in *SPT* are the following ones:

- In the assignment problem, if both workstations are free, priority is given to the workstations' IDs;
- In the assignment problem, if two or more tasks have the same processing time, priority is given to the tasks' IDs.

The rule repeatedly tries to send in execution first the tasks with the shortest execution time after all the constraints are verified to be satisfied.

The heuristic has negligible computing times. Figure 4.15 reports the resulting tasks Gantt chart. The total time to complete the tasks is 52 time steps.

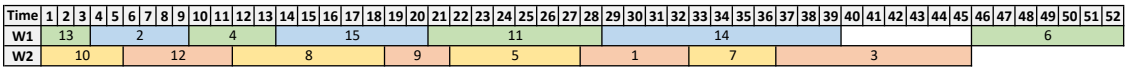

Figure 4.15: Scenario 2: Task scheduling resulting from SPT heuristic.

In conclusion, the *MPC* provides solutions with the minimum total time to execute the tasks. This is achieved at the expense of the computational complexity and the complexity of the control architecture (the approach requires building and solving an optimization problem and detailed modelling of the problem). On the contrary, simple heuristics, like the *SPT*, require no computational complexity and modelling effort at the expense of achieving a sub-optimal solution, often far from the optimal one.

The proposed approach presents the benefits of both the optimization-based approaches and the heuristics, being characterized by low computational complexity to execute in real-time, low complexity of the control architecture and reduced modelling effort, and good optimization of the cycle time (achieving a value that is intermediate between the optimal one, achieved by *MPC*, and

the one achieved by the heuristics). In conclusion, the *RL* approaches like the one presented in this paper, which is made more robust and scalable by the use of neural networks, can be integrated and can benefit from the ongoing Industry 4.0 digitalization trend (especially in the training phase, thanks to the digital twin concept), and appear as an up-and-coming solution for making the assembly lines more agile and able to react optimally, in real-time, to the disruption that could otherwise seriously impair the productivity of the factory.

4.4 City Logistics Use-case Testing by Control Technique

In this section, we validate the developed framework against several experiments, and the results are compared with existing solutions. Firstly, a comprehensive overview of the tested use case is presented. Secondly, the Deep Q-Network training set-up is described with the *MPC*. Later, an outline of the agent's learning performance during the training phase is provided. Finally, a baseline controller is introduced to validate the two proposed approaches.

4.4.1 Use-case Description

In literature, ad hoc intersections are usually modelled for result validation; in this work, a real road junction in Rome, Italy, is considered. An inspection view of the junction recovered all the arrival rates; in particular, each lane's arrival rate was found. Moreover, three different traffic conditions were retrieved: low, medium and high traffic intensity, and therefore, an arrival rate for each traffic condition for each lane was considered. The simulation lasts 3 hours: the first hour with low traffic, the second hour with medium traffic and the third hour with high traffic intensity. Providing various traffic flows and patterns is important in the learning mechanism since it maximizes the agent's performance.

4.4.2 Simulation Scenario

In this section, we define the dynamics of vehicles, traffic lights and intensity of vehicle low, medium and high arrival rates.

Vehicle Generation

The generation of the vehicles was modelled through a Poisson distribution setting the average arrival rate λ for each TL, reported in 4.1, conveniently. The source and destination lanes are fixed.

Traffic Intensity	λ_1	λ_2	λ_3	λ_4	λ_5
Low	0.13	0.086	0.086	0.05	0.2
Medium	0.52	0.35	0.35	0.2	0.8
High	0.975	0.645	0.645	0.375	1.5

Table 4.1: Car Arrival rates at the TLs.

An example of the arrival generation of one episode is depicted in Figure 4.16.

Vehicle model Dynamics

All the experiments are carried out in the open-source traffic simulator SUMO, which uses a car-following model; this means that each vehicle is kept at a safe distance from the car ahead, leaving the vehicles enough time to break in case of emergency stops. The car is modelled in this thesis following the Krauβ car-following model [34]. This consists in setting the following parameters for the car model:

Parameter	Value	Meaning
Length	4	The vehicle's netto-length (length) (in m)
minGap	2	Security Distance between two running vehicles (in m)
Acceleration	2.6	The acceleration ability of vehicles of this type (in m/s^2)
Deceleration	4.5	The deceleration ability of vehicles of this type (in m/s^2)
Max Speed	50	The vehicle's (technical) maximum velocity (in m/s)
Sigma	0.5	Driver Imperfection

Table 4.2: Car-following model parameters

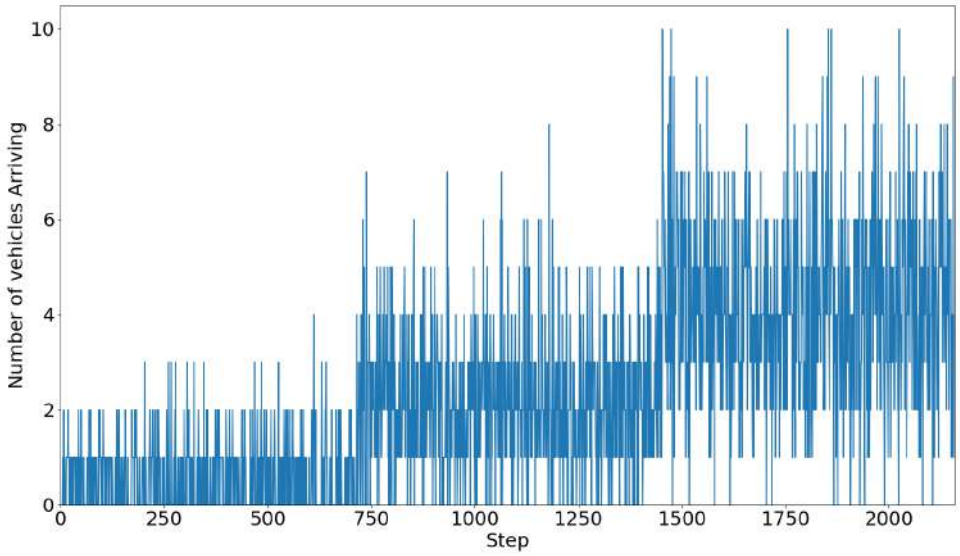

Figure 4.16: Traffic Generation Distribution over one episode

4.4.3 Experimental setting

The implementation and validation of the present work were made possible through the use of several tools and programming languages. The *MPC* environment was implemented and simulated in Julia programming language [9]; nevertheless, for modelling of the optimization problem, a particular Julia library was used for modelling the optimization problem; namely, JuMP [20] and we adopted the Gurobi solver [26]. This lets the user write the variables, constraints and cost function framework straightforwardly, particularly for *MILP* problems. The results are validated in the traffic Simulator SUMO (Simulation of Urban Mobility) [43]. We interface the Julia module with the simulator using the *PyCall* package, which is open-source and freely available.

4.4.4 Simulation results

As in all MPC applications, the larger the prediction horizon N, the closer the MPC performance is to the performance that would be achieved via solving an infinite-horizon optimization problem (i.e. which, however, cannot be solved in practice, as it includes an infinite number of variables, and also assumes in the present the knowledge of all the future relevant parameters and signals entering the problem.

4.4.5 Baseline Controller

This section first presents the baseline controller results needed to compare the agent performance of the proposed controllers. This is a *Static Traffic Light (STL)* which will cycle through every traffic light phase in the same order and with a fixed duration. In particular, the order and duration of each phase are denoted in the following:

- TLs 1 and 5 are green for 40s;
- TL 1 is yellow, and TL 5 is green for 5s;
- TLs 5 and 4 are green for 15 s;
- TLs 5 and 4 are yellow for 5 s;
- TLs 2 and 3 are green for 25 s;
- TLs 2 and 3 are yellow for 5 s, and a new cycle starts.

The controller is tested in a new scenario where the vehicles are generated according to Section *Veh Gen* and considers the three different traffic intensity conditions. The *STL* have very different results depending on the scenario. Validations are made in the traffic simulator SUMO, in which both the generation of the vehicles and the above cycle is implemented. The evaluation of the *STL* produced the following results, which will be the baseline for the agent's performance.

Low Traffic Intensity

In low-traffic conditions, a fixed cycle is not optimal as traffic signals manage only limited vehicles to pass the junction. In this scenario, they may be forced to wait even if there is no halting vehicle to wait for the green light. The figures below represent the number of cars halting and the cumulative waiting time at each time step during the first hour of Low-Traffic Intensity conditions.

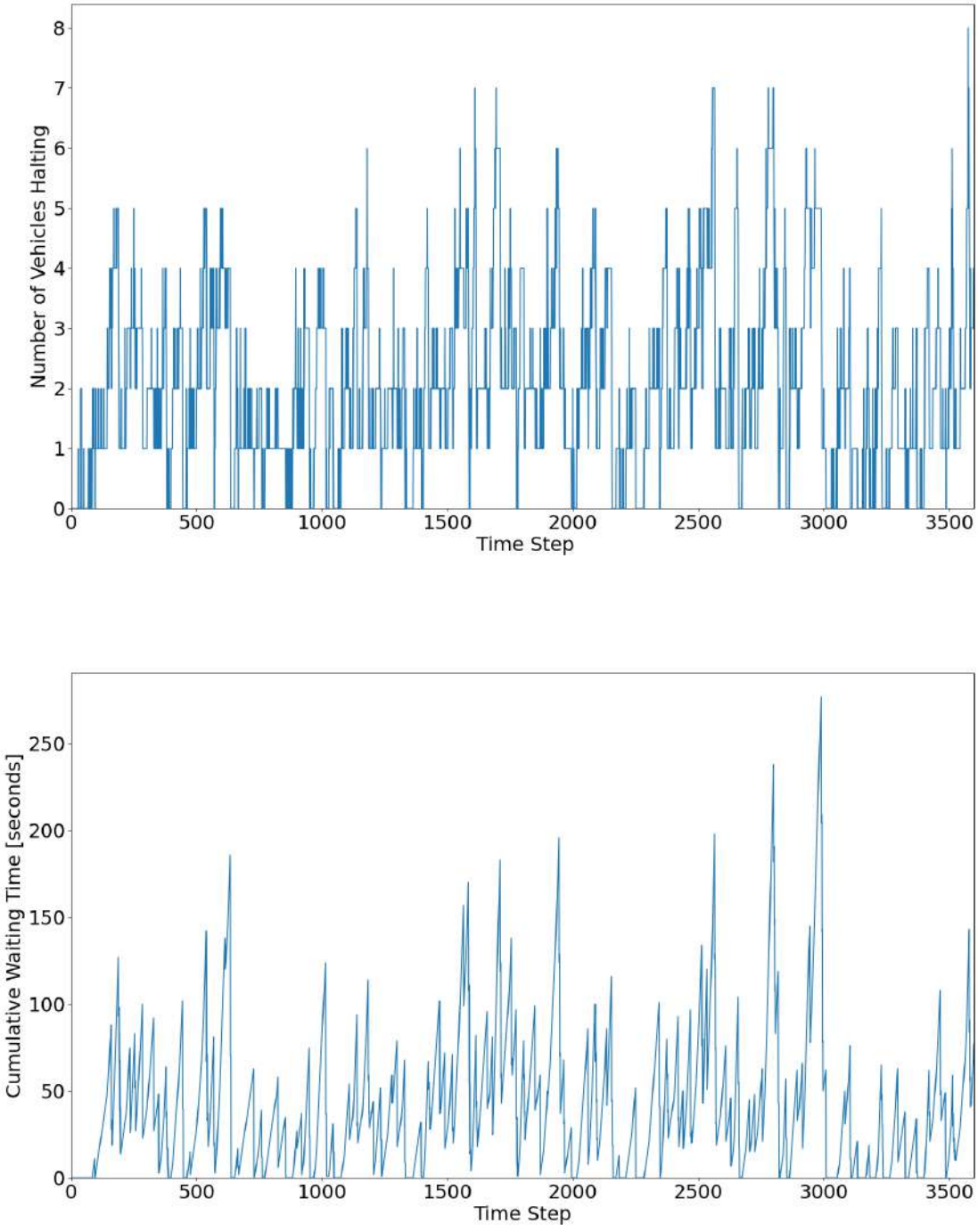

Figure 4.17: Number of Halting Vehicles and Cumulative Waiting Time in Low-Traffic Conditions

Medium Traffic Intensity

In this case, more vehicles approach the junction, leading to longer queues; the current method is arguably the best solution.

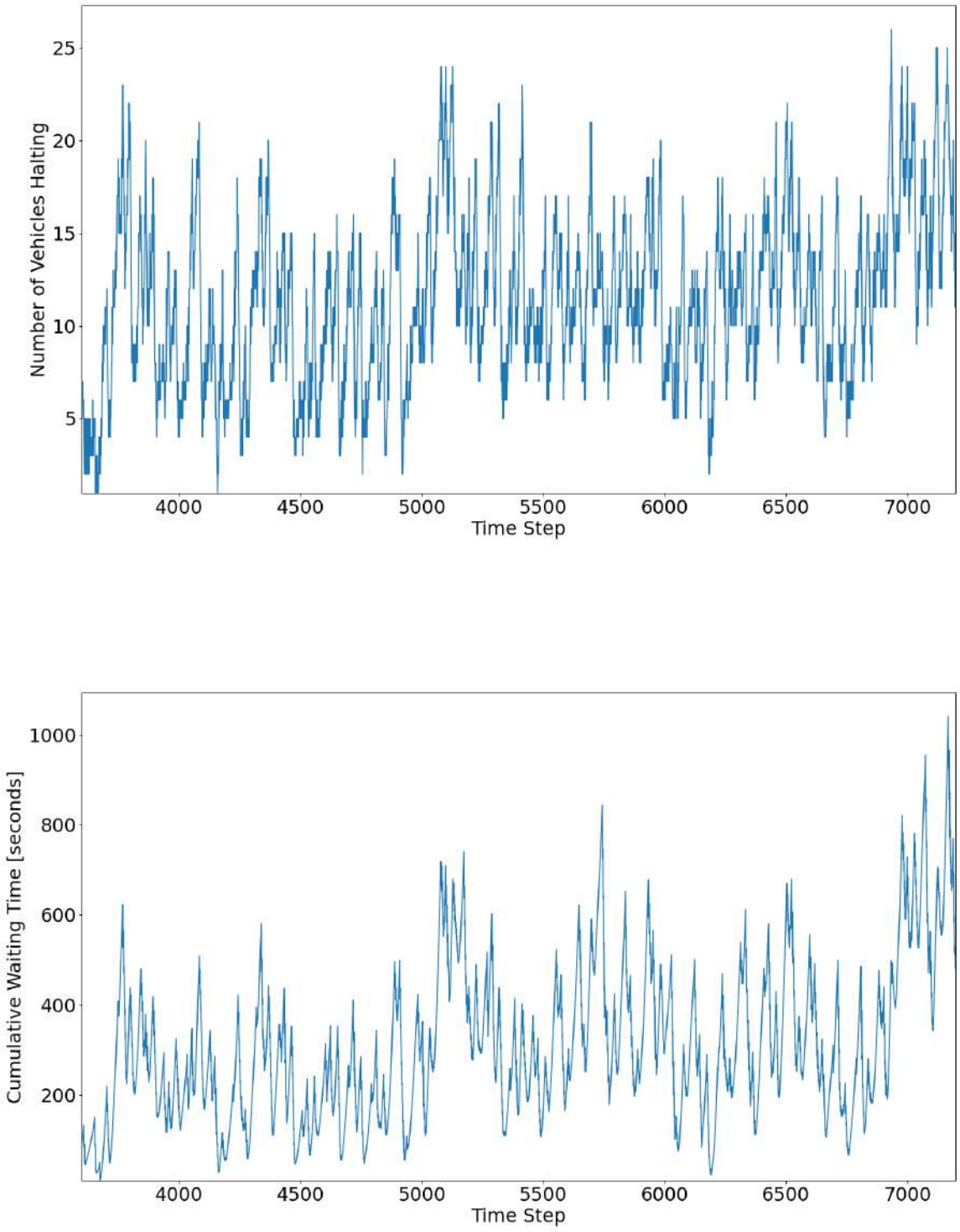

Figure 4.18: Number of Halting Vehicles and Cumulative Waiting Time in Medium-Traffic Conditions

High Traffic Intensity

In a high-traffic situation, many vehicles arrive from all directions, leading to long queues. According to 4.1, lanes 1 and 5 need more green time; hence, not prioritizing them will lead to highly mediocre performance; this is evident from the below figures where both numbers of halting vehicles and cumulative waiting time of the intersection diverge danger.

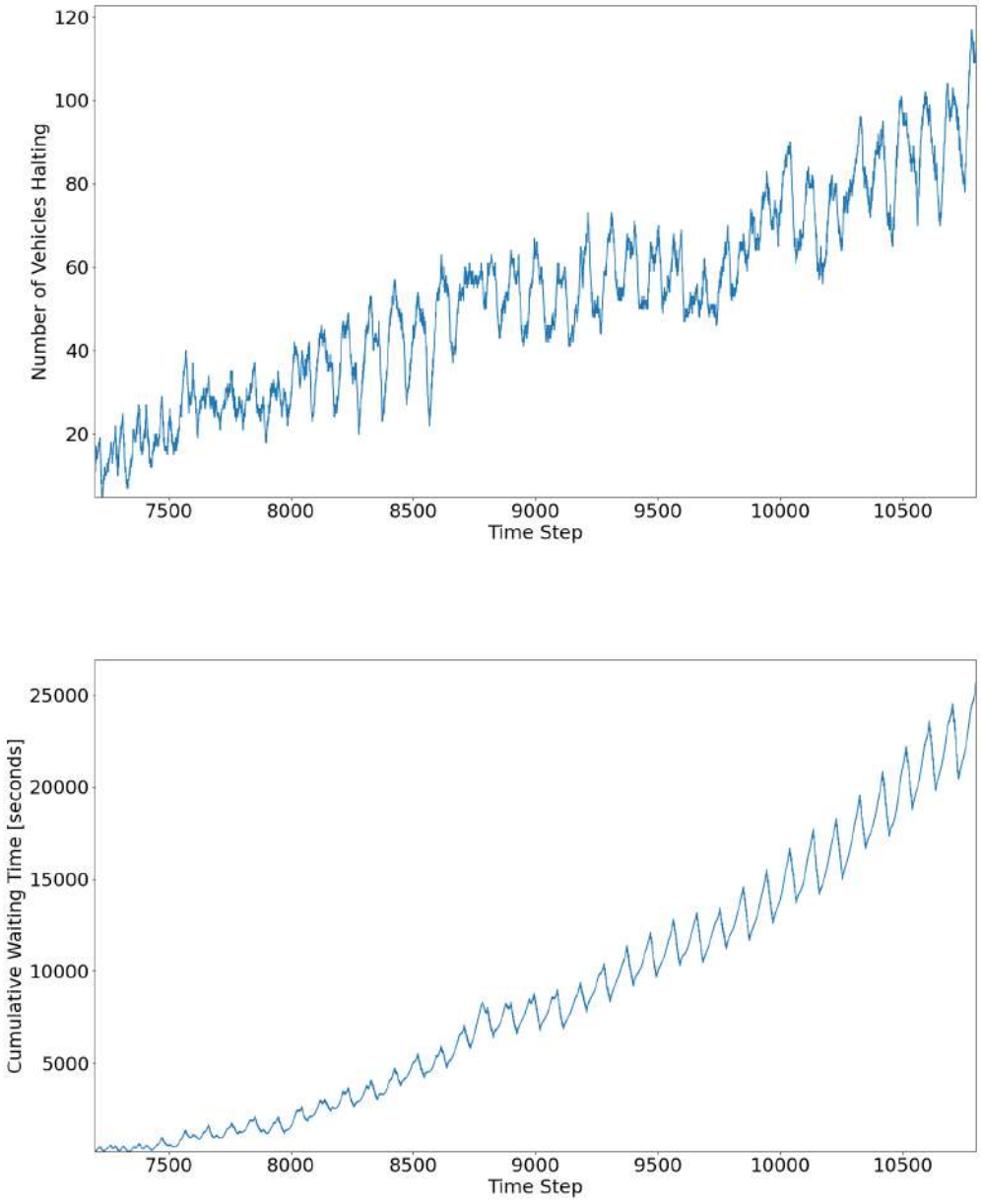

Figure 4.19: Number of Halting Vehicles and Cumulative Waiting Time in High-Traffic Conditions

Regarding the baseline controller as presented in Figures 4.19, waiting time and halting vehicles are exponentially increasing and it's not feasible in a real-time environment. In the next section, *MPC* based solution presented for the same scenario with prediction horizon value $N = 60$ has been chosen with a $T = 1$ second sampling time.

Low Traffic Intensity

The *MPC* approach is highly effective especially in which the vehicles are limit-ted. In fact, from the following figures, the improvements are evident in the *KPI*s.

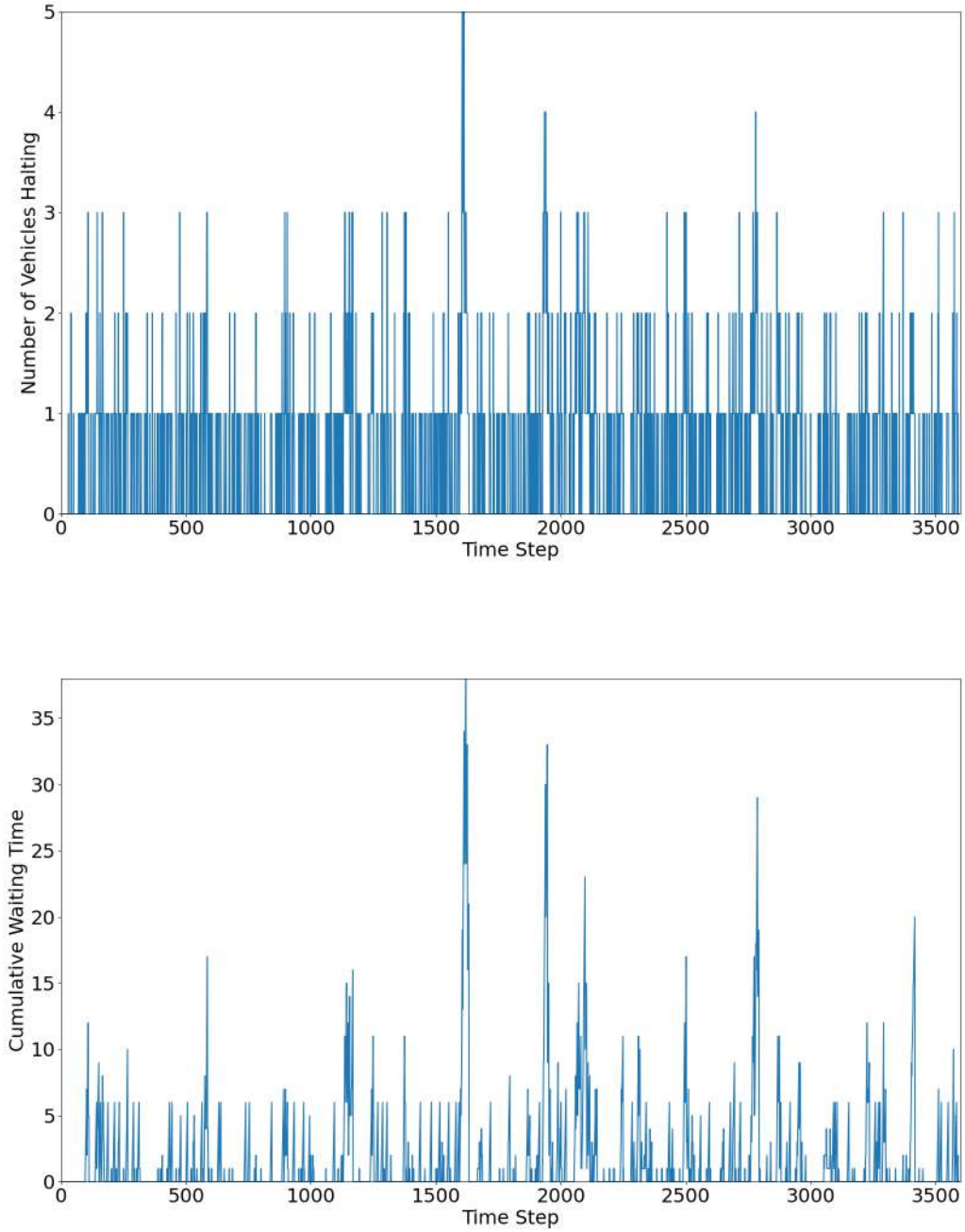

Figure 4.20: Number of Halting Vehicles and Cumulative Waiting Time in Low-Traffic Conditions

Medium Traffic Intensity

In the following Figures 4.18 the behaviour of *MPC* is presented.

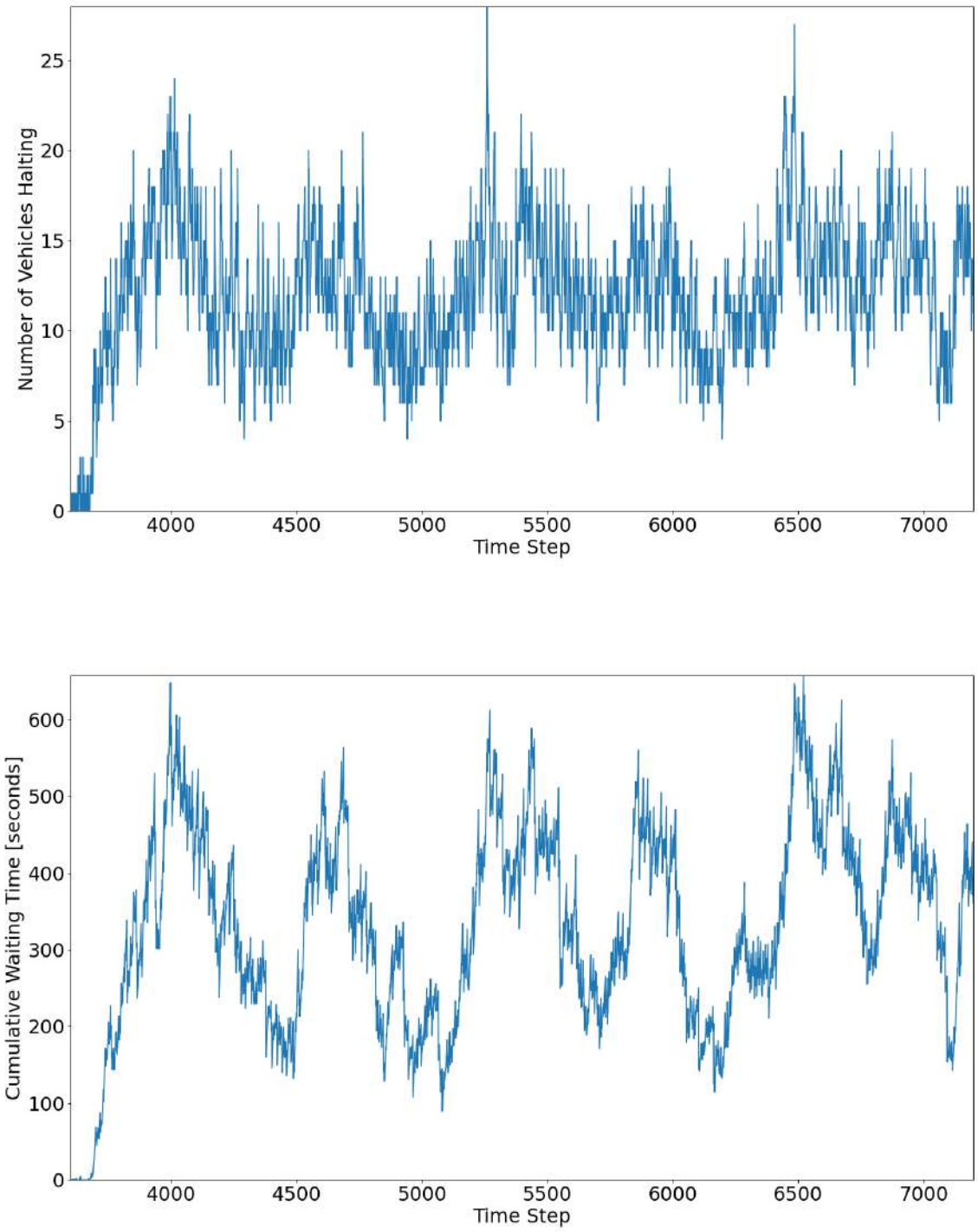

Figure 4.21: Number of Halting Vehicles and Cumulative Waiting Time in Medium-Traffic Conditions

High Traffic Intensity

In this scenario, the arrival rate of vehicles per lane increases substantially, leading to long queues; however, the proposed approach can still manage the situation differently from the baseline controller. This is evident from the following plots compared to Figures 4.19, which diverges badly.

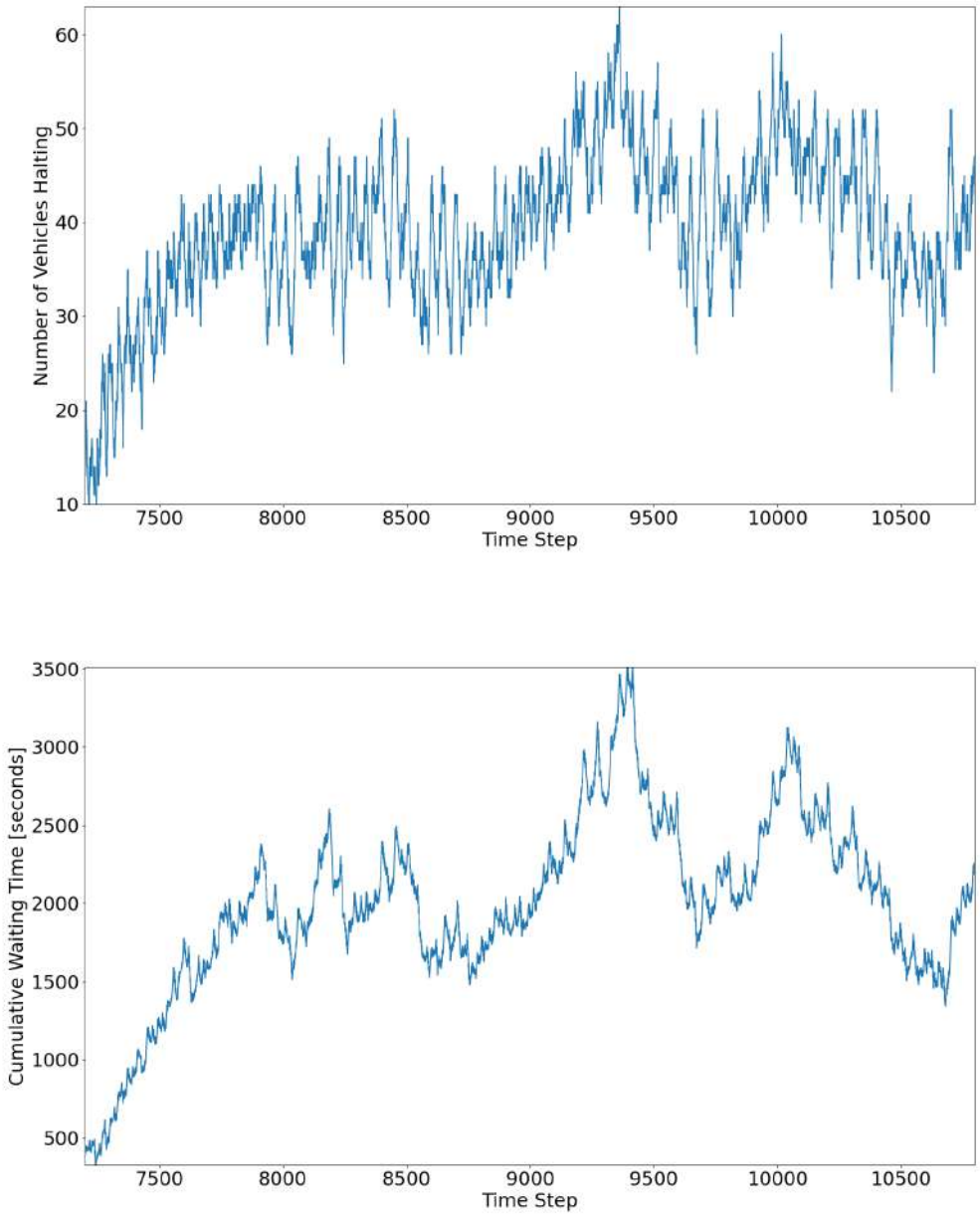

Figure 4.22: Number of Halting Vehicles and Cumulative Waiting Time in High-Traffic Conditions

The results of *MPC* techniques are more optimal than the baseline controller in low-traffic intensity hours. It's not scalable for complex scenarios and is computationally expensive. Further, we proposed a learning-based approach and trained the *DRL* agents to control the traffic signals. The proposed technique is scale-able and computationally feasible.

4.5 City Logistics Use-case Testing by Data-driven Technique

This section uses a scenario similar to the *MPC* section.

4.5.1 Experimental Set-up

In this section, we implemented the proposed *DRL* algorithm and simulated environment by Python language and following deep learning framework TensorFlow [1]. The results are validated using the traffic simulator SUMO (Simulation of Urban Mobility) [43]. Its Python interface Traci (Traffic Control Interface), enables the extraction of values from simulation, which are deplorable in a real environment. The simulation framework [43] was built to facilitate the research community for the high-level implementation of all the junction's features and traffic lights in a real-time fashion. This work modelled the crossroad by connecting nodes and edged precisely as in the real world. Moreover, the traffic lights were arranged as in reality, and the road lengths were respected. All the simulations were carried out on a Windows 11 machine 64-bit, equipped with an Intel i5-1035G1 CPU, 1.00 GHz and 8GB RAM.

4.5.2 Agent Training with Low, Medium and High Discount Rates

The neural networks were trained for 1200 episodes where each episode was set up according to Section 4.4.2; moreover, at each episode, a completely different scenario was provided to present as many situations as possible and therefore improve the learning performance. In such a way, just one trained model can be applied to any intersection and traffic condition.

Two experiments have been conducted for different values of γ to show how the performance changes by applying a forward-looking policy, $\gamma = 0.5$ or a greedy one $\gamma = 0.1$ instead. Both experiments are carried out with the same neural network structure, shown in Figure 4.23, while the other main hyper-parameters are shown in Table 4.3.

	$\gamma = 0.1$	$\gamma = 0.5$
Neural Network Structure	5x800	5x800
Number of Episodes	1200	1200
Sampling Strategy	Intensive	Intensive
Learning Rate	0.00025	0.00025
Batch Size	32	32
Input Dimension	15	15
Output Dimension	3	3
Memory Size	500000	500000

Table 4.3: Hyper-parameters values used for the training

During the training, several *KPI* are considered to prove the presented approach's effectiveness. As the workout progresses, the agent chooses random actions with a decreasing probability of the episode's performance worsening due to the random nature of the action in the beginning. The performance mechanism for agents is convergence in learning that indicates the agent has learned an excellent policy 4.24. Concerning the reward representation, just the negative values are considered; in particular, at each time step, the *CWT* was computed, but this was added to the cumulative reward only if it was negative. The agent's objective will be to maximize the cumulative negative reward.

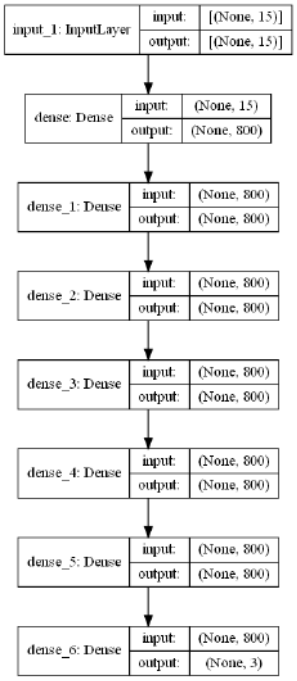

Figure 4.23: Model Structure of the Neural Network

Low Discount Rate

Setting γ at a low value means that the agent will be more likely to choose actions that provide immediate good rewards instead of looking ahead for a less greedy result. The training result in terms of reward and average waiting time is shown in Figure 4.24 and 4.25, which represents the cumulative negative reward and the average *CWT* acquired during all the episodes.

The former explicitly shows how the reward is maximized over the training phase, and at the same time, the latter shows how the average waiting time is proportionally reduced.

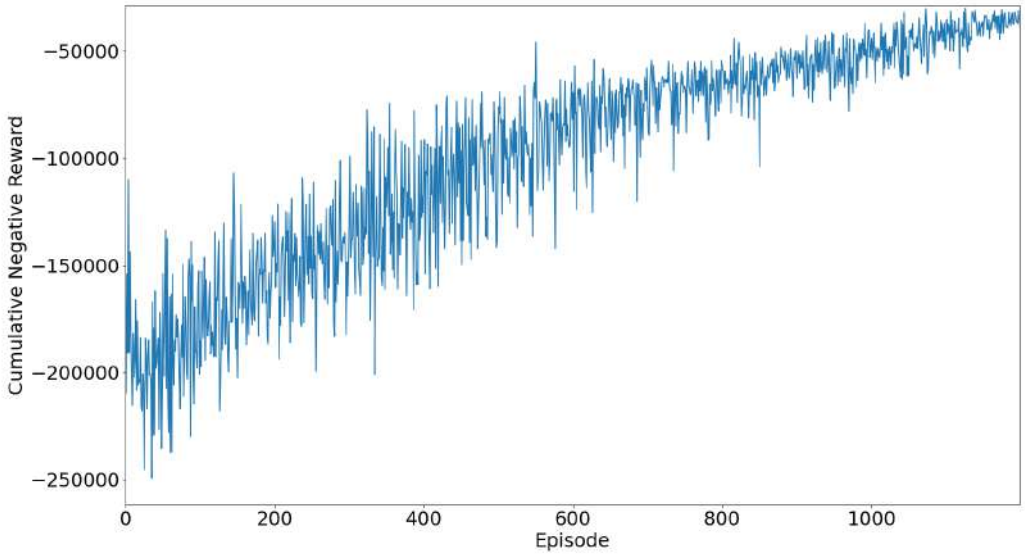

Figure 4.24: Cumulative Negative Reward over all the training phase

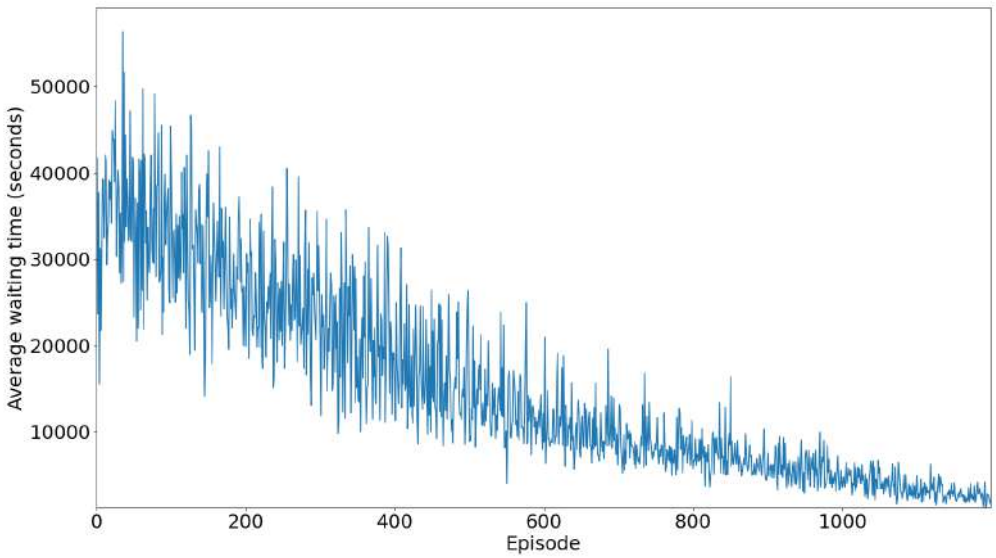

Figure 4.25: Average Waiting Time over all the training phase

The queue length is also another fundamental *KPI* whose course during the training phase is shown in Figure 4.26; as pointed out in Section *Sec Intro* the emission of CO_2 is a significant worldwide problem. The present approach reduces this issue drastically, as shown in Figure 4.27.

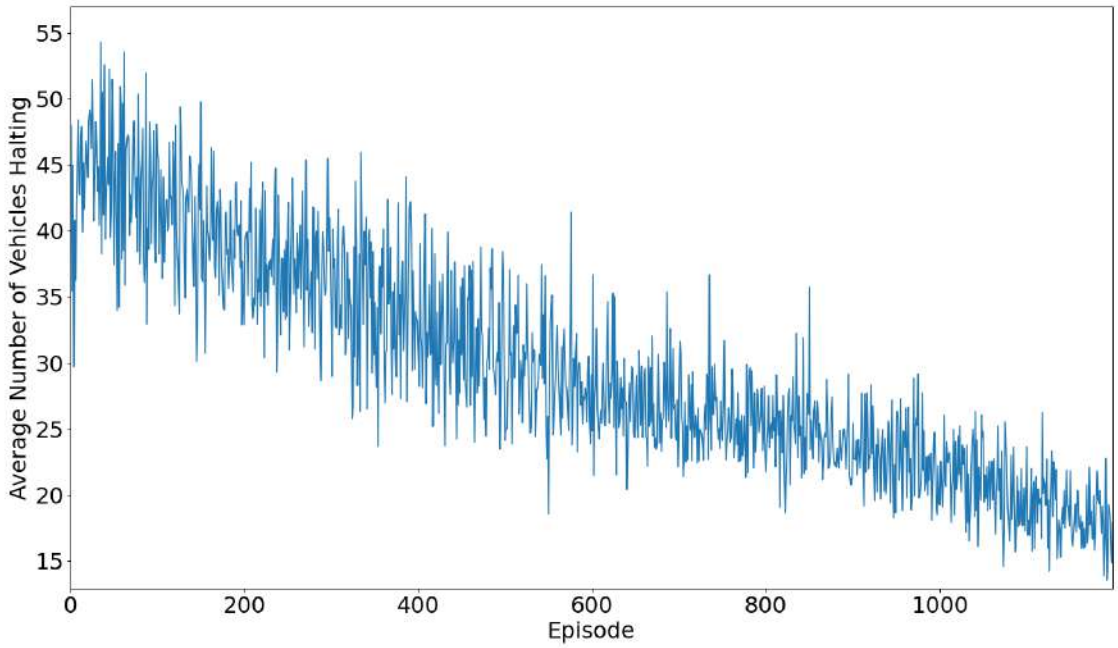

Figure 4.26: Average Number of Vehicles Halting over all the training phase

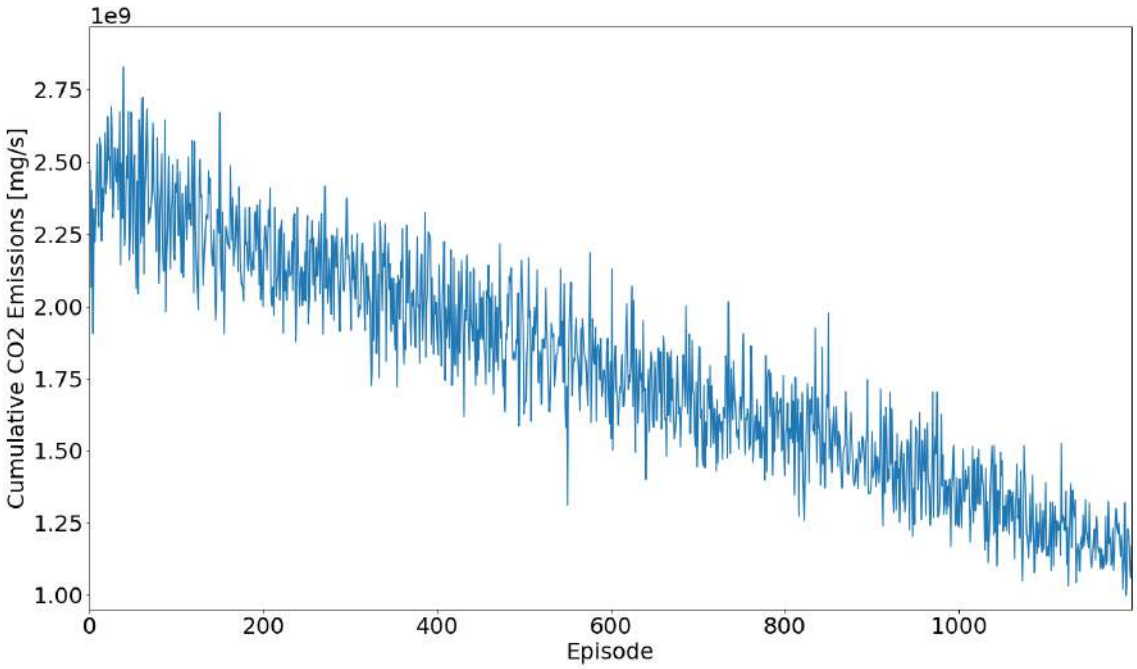

Figure 4.27: Cumulative CO_2 Emissions over all the training phase

High Discount Rate

Setting γ at a high value means that the agent aims to maximize the expected cumulative reward of multiple sequential actions; in particular, at each time step, the agent will first look forward before

choosing the optimal action.

The results in terms of reward and average waiting time are shown in the following:

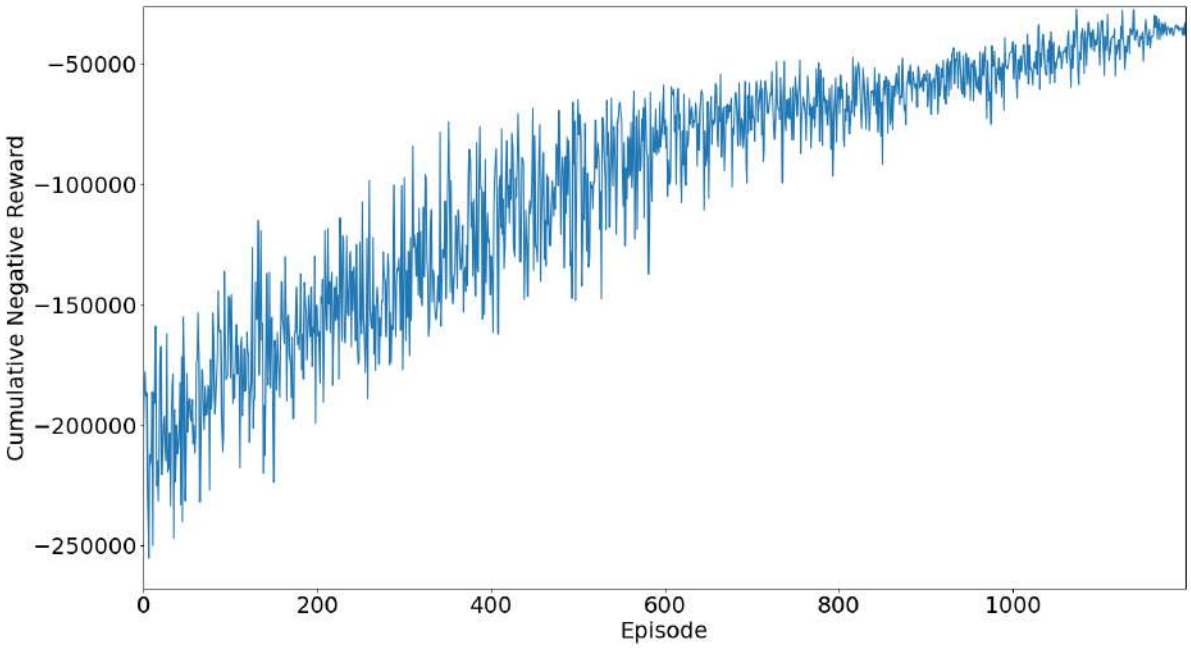

Figure 4.28: Cumulative Negative Reward over all the training phase

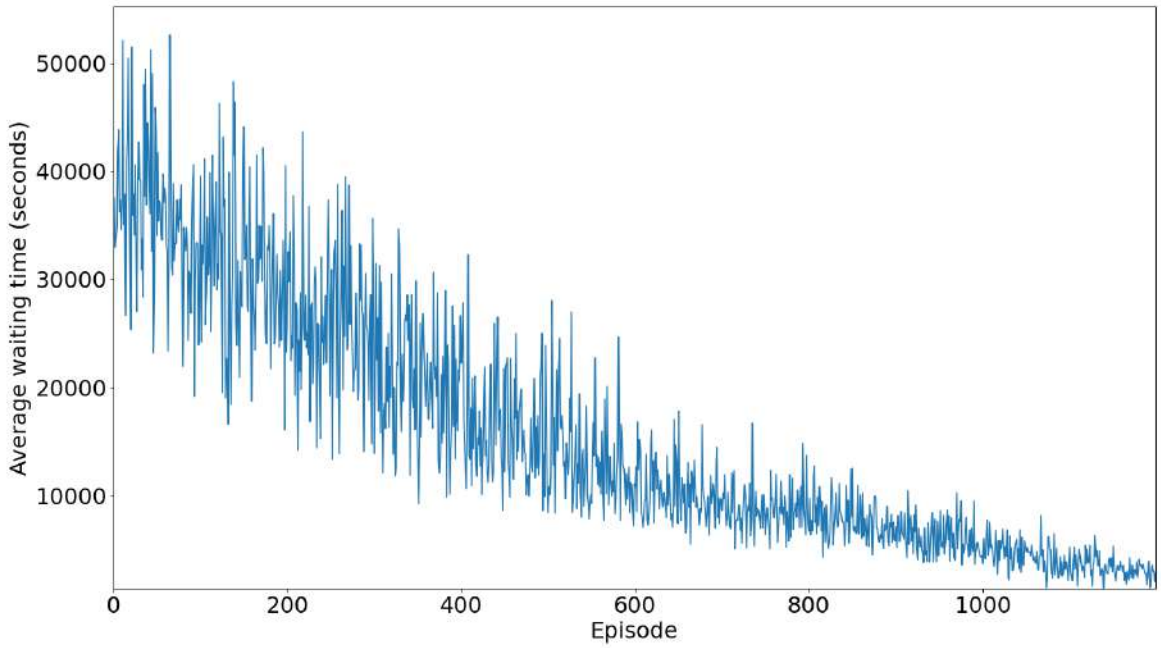

Figure 4.29: Average Waiting time over all the training phase

It is clear, even if slightly, that Figure 4.28 provides a better policy than the one in Figure 4.24;

its stability proves this during the last time steps in which the agent always performs fewer random actions in favour than the optimal ones. Similarly to the above case also f

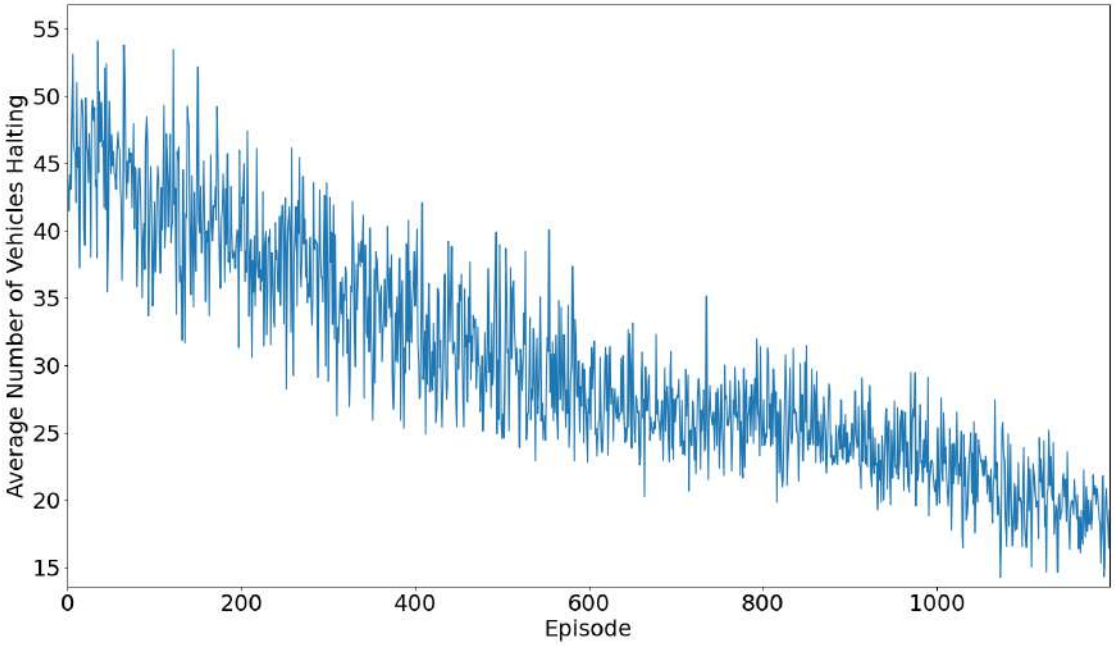

Figure 4.30: Average Number of Vehicles Halting over all the training phase

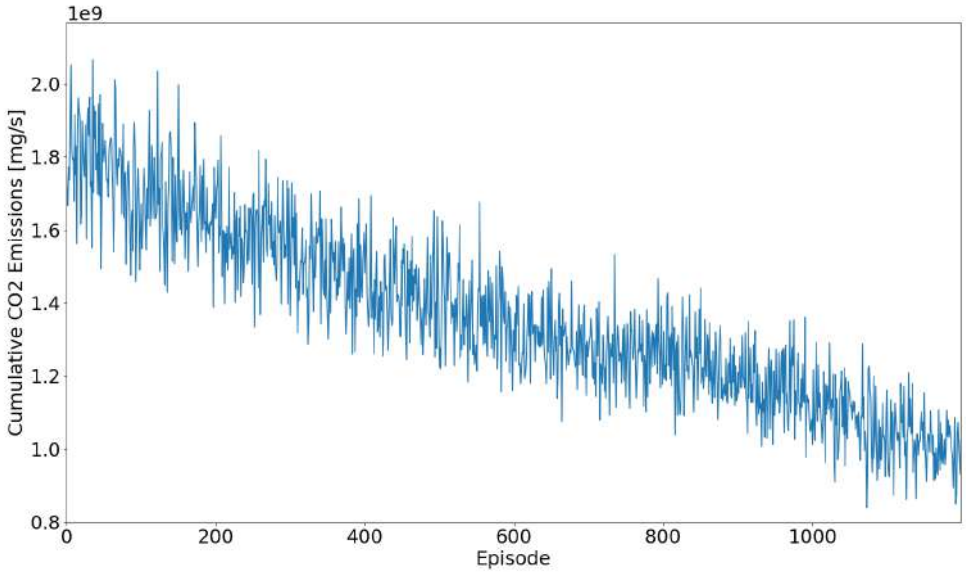

Figure 4.31: Cumulative CO_2 Emissions over all the training phase

4.5.3 Agent Performance checking with Low, Medium and High Traffic Intensity

At the end of the training phase, a trained model was provided; this was then applied to an entirely new scenario from the ones seen during the training phase to prove its effectiveness in the number of halting vehicles and cumulative waiting time. The following shows testing results for both γ values; each traffic intensity condition is described individually. The vehicle generation for the traffic conditions is based on Section *Sec Veh Gen*.

Low Traffic Intensity

In Low-Traffic conditions, the agent performs way better than the baseline controller; the figures below show an apparent reduction in the number of halting vehicles and cumulative waiting time. The following results also show the effectiveness of the trained model for this kind of scenario since one of the objectives was to have a unique model able to manage any traffic condition.

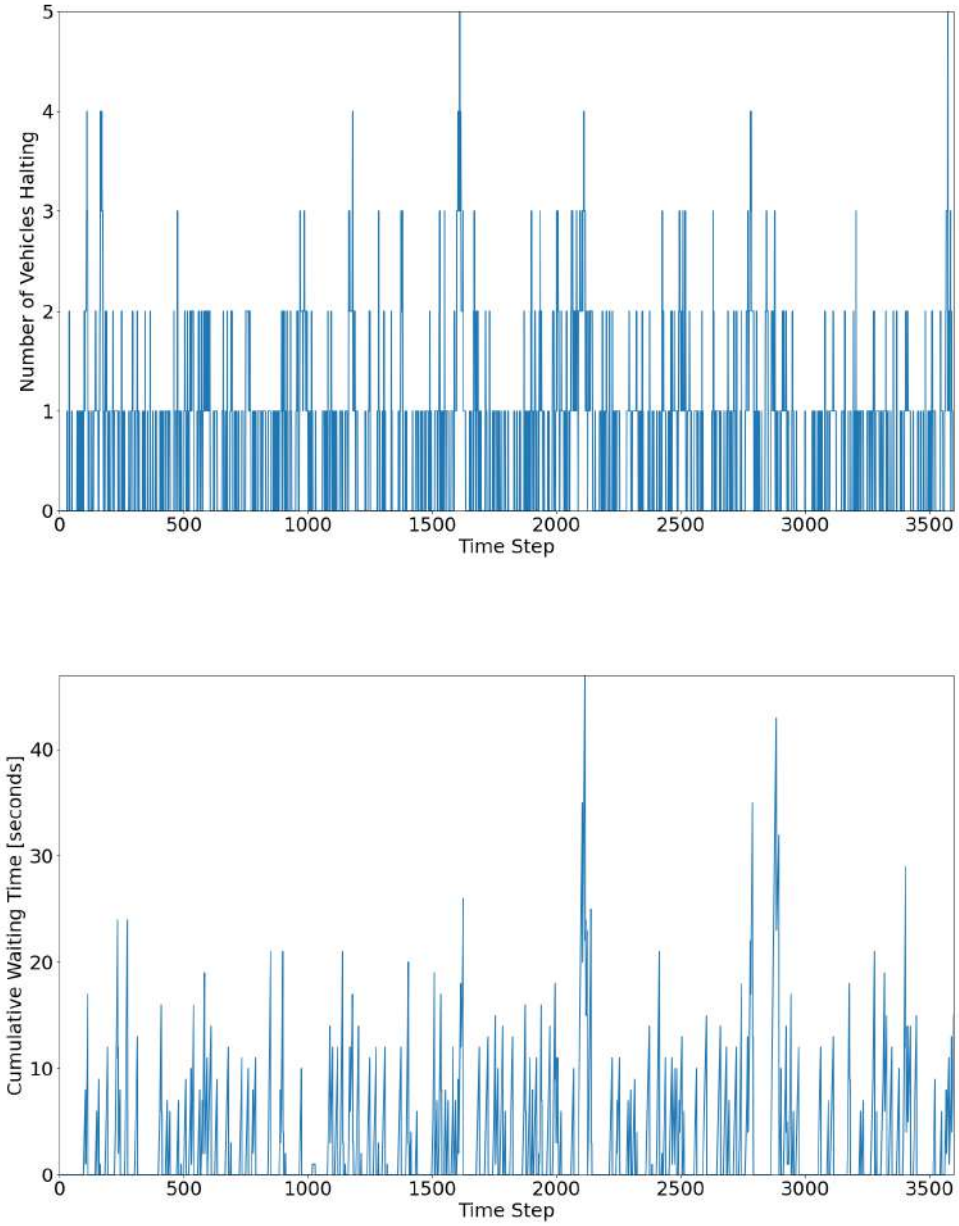

Figure 4.32: Number of Halting Vehicles and Cumulative Waiting Time in Low-Traffic Conditions

Medium Traffic Intensity

As clear from Figures 4.33, the trained model is still enhancing the current control performance; despite a small peak after a few minutes, the controller well manages the traffic while in Figures 4.18 both *KPI*s are exponentially increasing.

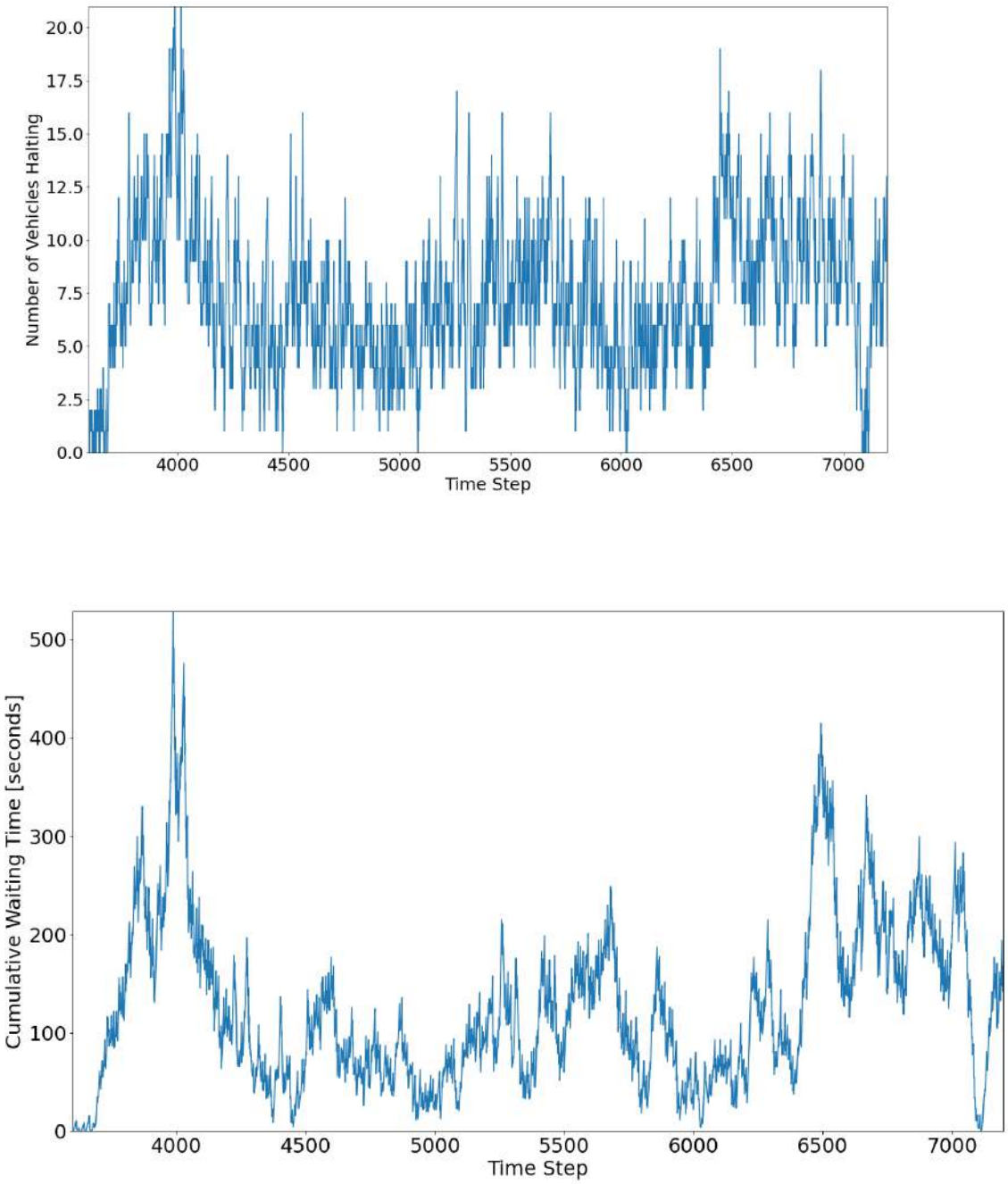

Figure 4.33: Number of Halting Vehicles and Cumulative Waiting Time in Medium-Traffic Conditions

High Traffic Intensity

In this scenario, many vehicles approach the junction, making very long queues of dozens of cars which the proposed controller could still manage; in particular, it shows noticeably better performance concerning the baseline (Figure 4.19). One of the reasons is that with the present strategy, the most trafficked lanes are prioritized, while in the fixed periodic phase case, each lane periodically had a fixed amount of green lights. In this case, lanes with the most cars are penalised, leading to

the situation in which there arrive more cars than the one manageable, leading to instability. The trained network has proven suitable for all traffic conditions, outperforming the baseline controller performance in each scenario.

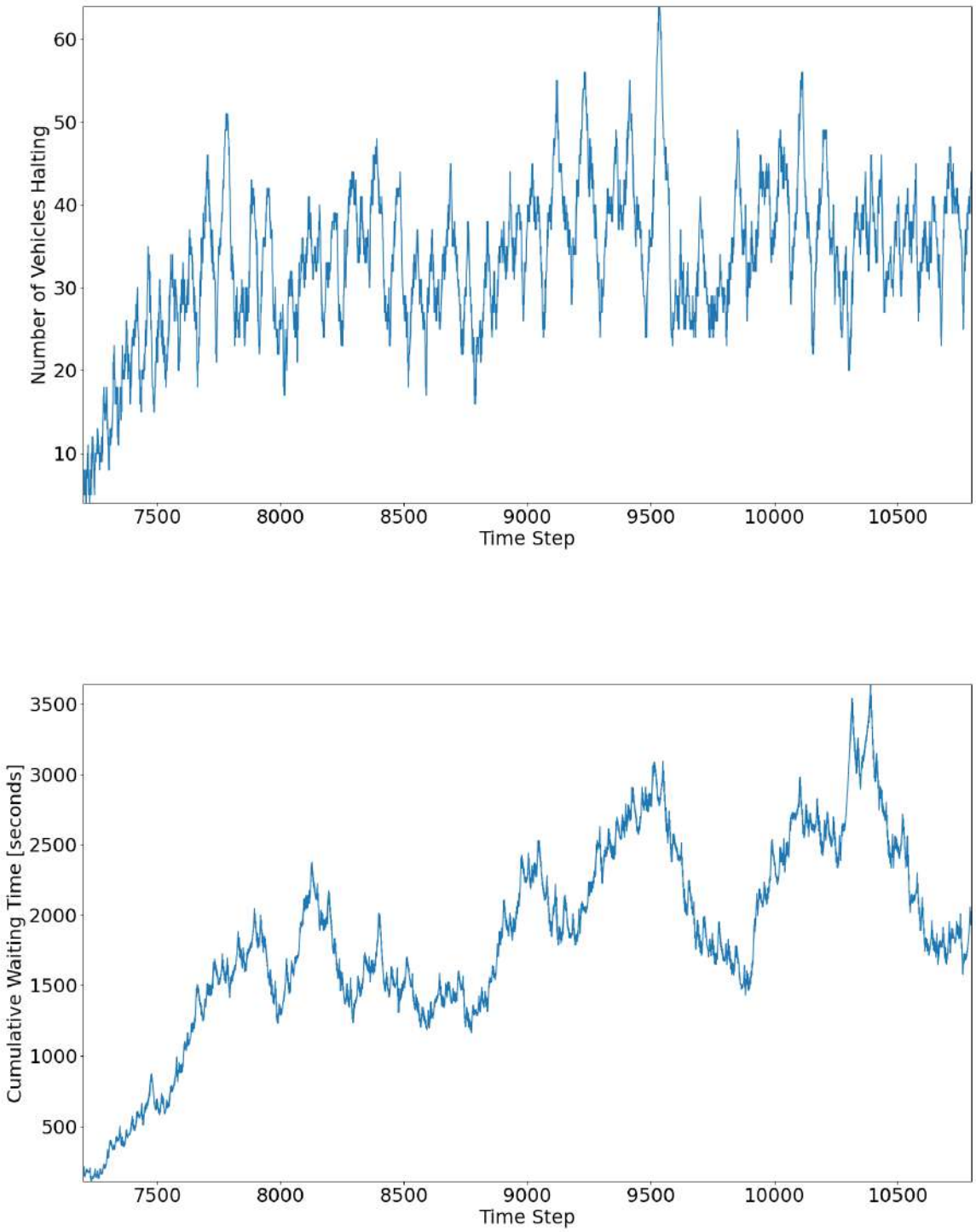

Figure 4.34: Number of Halting Vehicles and Cumulative Waiting Time in High-Traffic Conditions

4.5.4 High Discount Rate

The neural network with the structure according to 4.2 was trained offline with the hyper-parameters of Table 4.3 with $\gamma = 0.5$ and then tested; the results are shown below.

Low Traffic Intensity

The vehicles arriving in Low-Traffic conditions are few and therefore easy to manage; furthermore, predicting the actions more in the future seems the best strategy since it improves the baseline strategy in Figure 4.17.

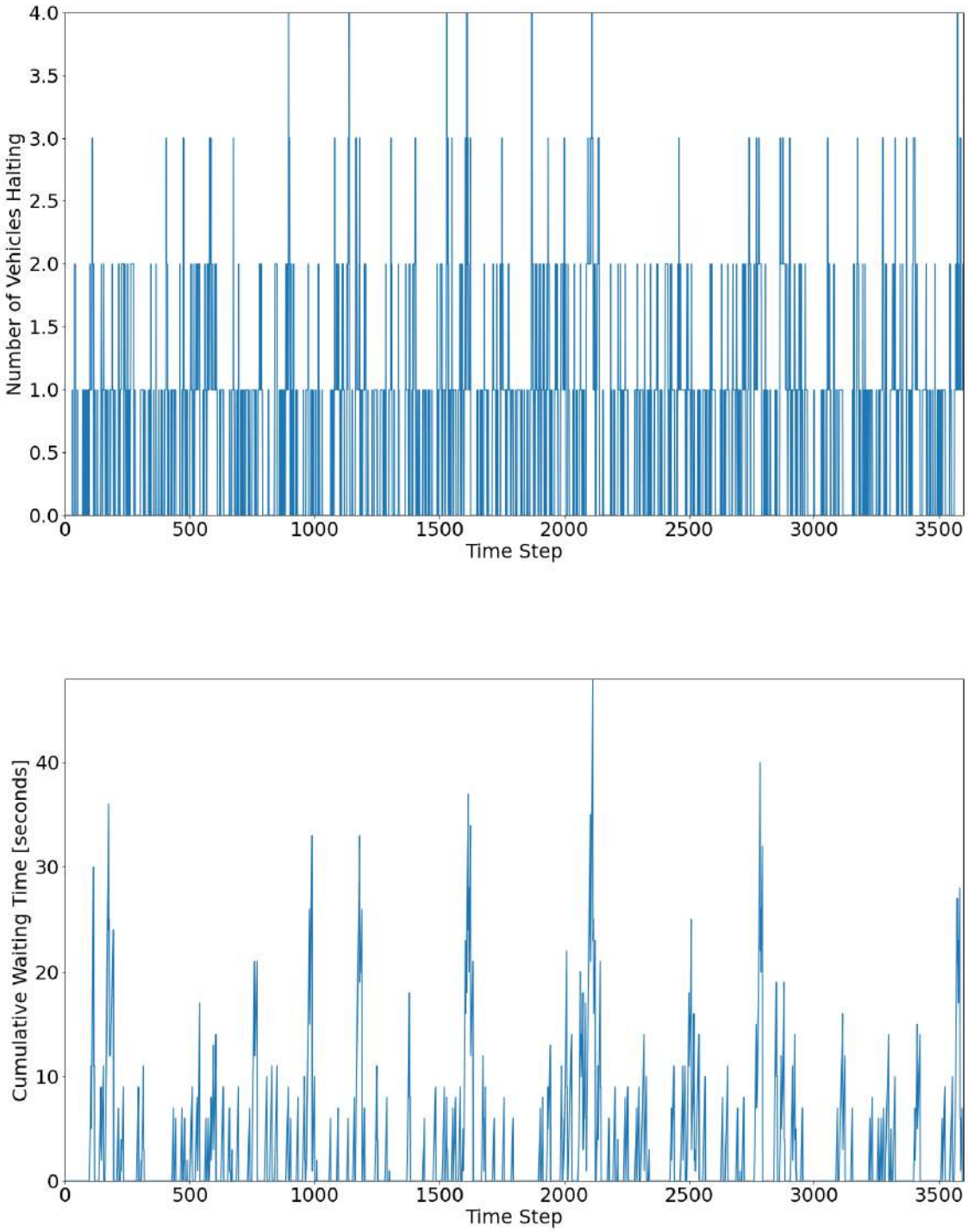

Figure 4.35: Number of Halting Vehicles and Cumulative Waiting Time in Low-Traffic Conditions

Medium Traffic Intensity

In medium intensity, the cars incoming in the intersection increase leading to longer queues, and therefore maximizing the future reward is the best strategy to avoid traffic congestion; this is also proved by the following figures, which clearly show the enhancement of the traffic flow when controlled by the proposed agent. In particular, the present controller outperforms the baseline controller in figures 4.18 and the above neural network with a more greedy policy.

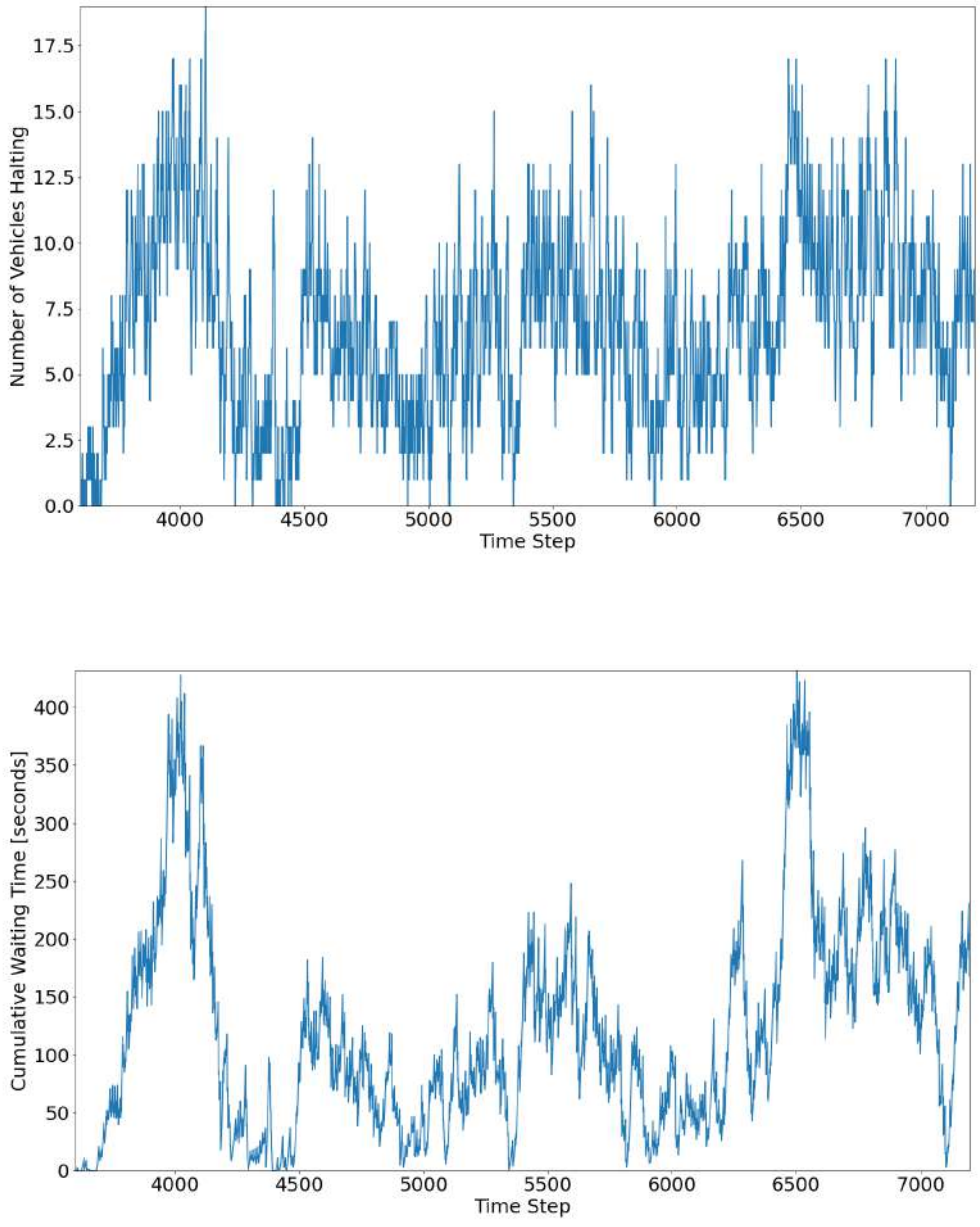

Figure 4.36: Number of Halting Vehicles and Cumulative Waiting Time in Medium-Traffic Conditions

High Traffic Intensity

The present agent outperforms the above in high traffic conditions, as shown in figures 4.37, reducing the number of halting vehicles and the cumulative waiting time even further. Indeed, the proposed agent is efficient in all traffic conditions providing good performance. In conclusion, it was seen how the proposed approach based on an adaptive signal control strategy substantially enhanced the current periodic strategy; moreover, the agent which aimed to maximize long-term reward showed better performance than the greedy one. Finally, it was proved how a unique agent could deal with every traffic condition with better performance concerning the existing ones.

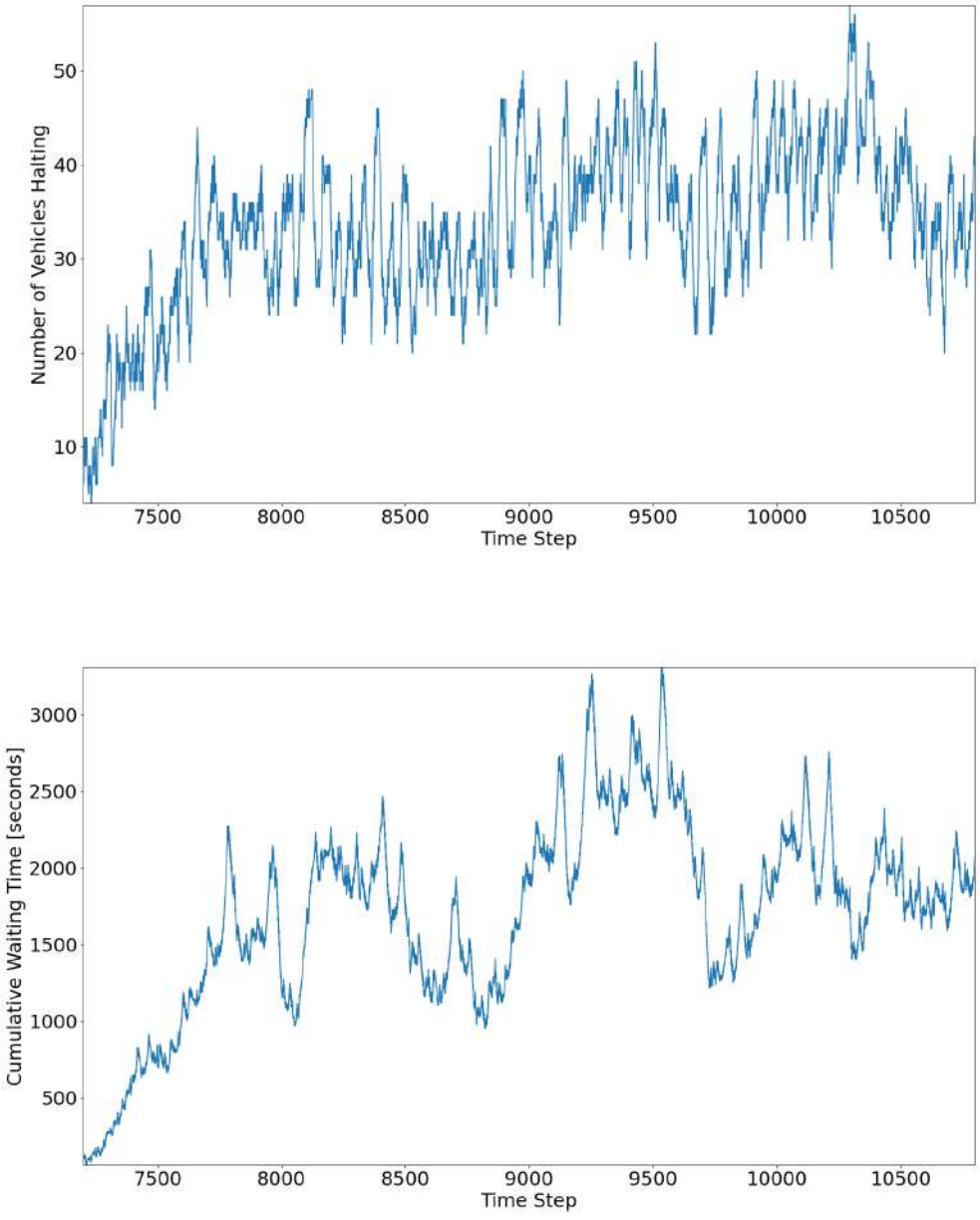

Figure 4.37: Number of Halting Vehicles and Cumulative Waiting Time in High-Traffic Conditions

4.6 Comparison of Control and Learning-based Techniques

The obtained result can finally be used to compare the two proposed controllers: the agent trained with *DQN* algorithm and the *MPC* controller. The comparison is made in terms of both results and computational time; in particular, for the latter, we mean the time spent in finding the optimal solution at each time spent. Concerning the former, the model-based controller provided good results regarding halting vehicles and waiting time of the intersection outperforming the baseline controller; moreover, it gave similar performance to the greedy *DRL* agent while the agent trained with $\gamma = 0.5$ showed the best performance in any scenario. However, the Model Predictive Control agent requires a detailed mathematical model, which is not always easy to implement, and which, for high-dimensional problems, increases the problem complexity drastically, leading to too high computational times.

In particular, the main drawback in this sense is that the *MPC* is solving an optimization time at each time step; this means that unless really power computers are available, real-time control is quite complex to implement, especially for complex systems.

On the other hand, it allows them to manage the environmental constraint very quickly and minimize or maximize any variable through its objective function. Concerning the *DRL*, this does not require any mathematical model describing the system dynamics; nevertheless, the correct state information and way of providing these needs to be found. Furthermore, based on the aim we want to reach, a reward function needs to be modelled appropriately, which is not always straightforward. However, this approach is very suitable for high-dimensional problems thanks to the Deep Neural Network approximations; despite the time it took to train the agent, this is effective in real-time applications.

Chapter 5

Conclusion

In this Thesis, we address two use cases; the first is related to the production capabilities and logistics improvement at the European Spaceport in French Guiana in the context of the SESAME project. Second, Related to city logistics management problems, including traffic lights, a unit of the overall city management system. We propose the *MPC* and *DRL* based approaches to address both use cases. At first, each one is formulated in terms of control and then in terms of learning-based technique and results are compared within and state of the art techniques.

5.1 Smart Manufacturing Use-case Conclusion

The expected outcomes of the control and data-driven software module are to assist in optimal planning of the factory floor activities and provide real-time decision support in the case of adverse events impacting the assembly processes, such as a fault on a workstation/resource. We proposed an optimization-based technique that allows an accurate assembly process model and gives high flexibility in choosing an objective function, making the algorithms useful in real and complex scenarios. To achieve defined goals, Firstly, we formulated the assembly line process in terms of *MPC* to capture all the main constraints and dynamics that need to be considered by the optimization algorithms to build a feasible and implementable schedule. We use the classical Receding Horizon family of *MPC* technique, in which only the first predicted control input is considered at each instant obtained from the system's current state for defined prediction horizon ph. The objective is to reduce the make-span by optimal task assignment at the plant, placing the tasks in adequate workstations considering essential concepts such as precedence constraints, workstation availability and resource usage. We implemented the proposed algorithm in Mixed-Integer Linear Programming (MILP), and simulation results are presented as proof of concept. We proposed several scenarios to prove the overall algorithm's capabilities, and results are published in [42].

The nature of the industrial manufacturing environment is complex and dynamic, and it's challenging to model such environments in real-time scenarios. Especially in the age of technological developments, the *Industrial Internet of Things (IIOT)* and *Industrial Data Space (IDS)*. The proposed methodology must be more scalable and adaptable, but the control-based approach is computationally expensive, domain-dependent and incompatible with energy consumption. Digital technologies offer new solutions to industries with dynamic and resilient tools. The recent *DRL*) breakthrough has gained much attention from the *OR* and Control community. We proposed a *DRL*

approach to overcome these limitations and developed a scale-able *DRL* framework for industrial assembly lines to reduce the overall cycle time, which DeepMind recently cited in the following article[15]. It is scalable and in line with the dynamic nature of the industrial production systems to overcome the domain-dependent heuristics extensively used in the manufacturing industry.

We use a model-free and value-based deep RL-based *DQN* algorithm for optimal scheduling and control of tasks and resources assigned to workstations in an assembly line. The algorithm returns an optimal schedule for allocating tasks and resources to workstations in the presence of constraints. The main objective is to minimize the overall cycle time and the number of resources circulating in the assembly line. In contrast with the standard optimization-based approaches from operational research, the proposed approach can learn the optimal policy during a training process, which can be run on a digital twin model of the assembly line (the simulation environment in this paper was developed by using openAI gym [51]). Furthermore, the proposed method is scalable and, after training is completed, can be used for decision support to the scheduling operators in real-time, with little computation times. This can also be useful in re-planning after an adverse event, like unexpected delays, faults, etc.

The proposed *DRL* approach solves the dimensionality problem that impairs classical, tabular *RL* methods (many states and actions). A parallel training approach for the agents was used, reducing the training times since more than one agent simultaneously takes action and learns from the environment observation. The proposed multi-agent *RL* approach generates promising results in minimising the makespan, demonstrating the feasibility of the proposed method, and results are reported in [76]. Further, we proposed another algorithm *Proximal Policy Optimization (PPO)* to address the same use. The proposed approach showed promising results in terms of depreciation of the **makespan**, which demonstrates the feasibility of the proposed method and also demonstrates that the *PPO* performs better than the standard algorithm since it shows less execution time and has no dimensional problem (linear growth), and performs better than dispatching rules-based *DRL* where the *the degrees of freedom* are reduced.

5.2 City Logistics Use-case Conclusion

This thesis implements two different control strategies in the context of traffic signal control. First, a model-based *MPC* formulation is presented; second, a model-free Deep Reinforcement Learning algorithm is used to design a trained agent. Finally, both control strategies are validated in a real-world traffic simulator. A vehicle waiting time model is provided for the former, optimising the optimization of the *KPI*; an optimization framework imposing the mandatory constraints and a cost function to minimize the junction's total waiting time are presented. Instead, the latter uses the *DQN* algorithm to train the agent; the state representation identifies for each lane: the number of halting vehicles, the cumulative waiting time and the average arrival rate. The action dictionary is defined by legal configurations and their relative yellow phases.

In contrast, the cumulative waiting time was used as a reward to focus on the waiting time *KPI*. The learning approach applied for the learning mechanism is a combination of Q-learning and Deep Neural Network; the Q-learning is used for the update of the action values as the experience of the agent increases, and the neural network is employed for the Q-values prediction and, therefore, the approximation of the state-action function. The SUMO traffic micro-simulator was used to replicate

the considered intersection and to reproduce various traffic intensity conditions. Results indicate that both the proposed controllers outperform the current periodic strategy; moreover, the trained agent can adapt to several situations and traffic intensities. Finally, it was also proved that the proposed approaches could enormously decrease the pollution caused by queued junctions.

Deep Reinforcement Learning had a lot of success and was used in numerous areas; however, it is still a newly developing field. In addition to the present work, alternative techniques may be used, and several parameters may be improved, leading to an always more realistic and precise environment. The state representation could include more information, such as the velocity of each vehicle, the occupancy of each lane or the traffic light status, which will help the agent choose the best actions; nevertheless, always paying attention to not over-increasing it and worsening the agent performance. The dictionary of the actions from which the agent can choose the phase to perform may benefit from more flexibility, for example, by being allowed to control the phase's duration time.

In the current environment, the arrival rate is modelled through a Poisson distribution treating each event as independent; indeed, the arrival rate is constant, while an improvement may be obtained using a not homogeneous Poisson distribution. Moreover, the departure rate is kept constant and left to the SUMO car modelling. A more accurate departure rate can be implemented by extrapolating the acceleration profile from the simulator. Like most works in literature, the present thesis deals with a single road intersection; however, further improvement can be made by considering a larger road network or introducing extra agents leading to a Multi-Agent environment.

Concerning the algorithm, this may be improved by introducing extra optimization elements such as Double Dueling Network or using a different optimization algorithm; currently, the most used is *PPO* which, differently from the *DQN* that uses Q-learning, it's based on a direct policy optimization. This is an on-policy algorithm, meaning it only uses the data collected from the most up-to-date policy and makes decisions based on that.

5.3 Future Work

Digital transformation in the industrial process and future cities' are feasible by developing secure, controlled and computationally feasible operational software suites. The cornerstone of these techniques is solid mathematical foundations to cope with this fast-moving field. The recent *DRL* breakthrough has gained much attention from the *OR* and Control community. As addressed in this thesis, the techniques can solve one part of the digital transformation ecosystem. In *OR*, we need to account for domain heuristics, and in terms of *MPC*, we are limited to the prediction horizon and need extensive computation. The breakthrough in the process industry is expected by combining the *DRL* technique with existing methods *OR* or *MPC* to gain industrial excellence and offer different solutions. In reality, we need hybrid approaches by combining the strengths of *OR*, Control and *DRL* techniques that are deployable in a cross-industry industrial environment to address domain-dependent constraints, heuristics, and mathematical and computational limitations.

In future, we are very interested in developing hybrid approaches in the following ways:

- To combines the *OR* and *DRL* technique to develop the hybrid approach to overcome the extensive use of domain-dependent constraints and heuristics.

- To combine the control, specifically *MPC* and *DRL* to develop the hybrid approaches to overcome constraint, computation, and time horizon limitation.

It is a promising area of system design, and mathematical guarantees are crucial for real-time deployment, considering human safety as the central point of interest. The main goal is to develop a general framework/enterprise for the cross-disciplinary deployment of interconnected and data-intensive industrial and intelligent societies. The developed software suite provides a decision support system to industrial operators and managers based on real-time data to predict asset failure and enable predictive and proactive maintenance initiatives to increase the Return on Assets. Government agencies provide efficient service to taxpayers, manage the law and order situation and meet the United Nations Sustainable targets.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] F. Abbracciavento, F. Zinnari, S. Formentin, A. G. Bianchessi, and S. M. Savaresi. Real-time optimal traffic management in signal-controlled intersections: a receding-horizon approach. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 1947–1952. IEEE, 2021.

[3] F. Abbracciavento, F. Zinnari, S. Formentin, A. G. Bianchessi, and S. M. Savaresi. Multi-intersection traffic signal control: A decentralized mpc-based approach. *IFAC Journal of Systems and Control*, page 100214, 2022.

[4] D. Adebanjo, P.-L. Teh, and P. K. Ahmed. The impact of supply chain relationships and integration on innovative capabilities and manufacturing performance: the perspective of rapidly developing countries. *International journal of production research*, 56(4):1708–1721, 2018.

[5] K. Ağpak and S. Zolfaghari. Mathematical models for parallel two-sided assembly line balancing problems and extensions. *International Journal of Production Research*, 53(4):1242–1254, 2015.

[6] T. Altenmüller, T. Stüker, B. Waschneck, A. Kuhnle, and G. Lanza. Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints. *Production Engineering*, 14(3):319–328, 2020.

[7] I. Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Management science*, 32(8):909–932, 1986.

[8] C. Becker and A. Scholl. A survey on problems and methods in generalized assembly line balancing. *European journal of operational research*, 168(3):694–715, 2006.

[9] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[10] N. Boysen, M. Fliedner, and A. Scholl. A classification of assembly line balancing problems. *European journal of operational research*, 183(2):674–693, 2007.

[11] J. R. Buch, Y. P. Kakad, and Y. H. Amengonu. Performance comparison of extended kalman filter and unscented kalman filter for the control moment gyroscope inverted pendulum. In *2017 25th International Conference on Systems Engineering (ICSEng)*, pages 57–62. IEEE, 2017.

[12] Total Manufacturing Revenue by Segment , World Markets 2018-2025. (2019, March 11). `https://omdia.tech.informa.com/`. Accessed: 2019.

[13] A. Cataldo. Model predictive control in manufacturing plants. 2017.

[14] Y.-Y. Chen, C.-Y. Cheng, and J.-Y. Li. Resource-constrained assembly line balancing problems with multi-manned workstations. *Journal of Manufacturing Systems*, 48:107–119, 2018.

[15] Y. Chervonyi, P. Dutta, P. Trochim, O. Voicu, C. Paduraru, C. Qian, E. Karagozler, J. Q. Davis, R. Chippendale, G. Bajaj, et al. Semi-analytical industrial cooling system model for reinforcement learning. *arXiv preprint arXiv:2207.13131*, 2022.

[16] M. Coşkun, A. Baggag, and S. Chawla. Deep reinforcement learning for traffic light optimization. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 564–571. IEEE, 2018.

[17] D. Darma, Z. Ilmi, S. Darma, and Y. Syaharuddin. Covid-19 and its impact on education: Challenges from industry 4.0. *Aquademia https://doi. org/10.29333/aquademia/8453*, 2020.

[18] R. Davies. Industry 4.0: Digitalisation for productivity and growth. 2015.

[19] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[20] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[21] G. Erboz. How to define industry 4.0: main pillars of industry 4.0. *Managerial trends in the development of enterprises in globalization era*, 761:767, 2017.

[22] R. Esmaeilbeigi, B. Naderi, and P. Charkhgard. The type e simple assembly line balancing problem: A mixed integer linear programming formulation. *Computers & Operations Research*, 64:168–177, 2015.

[23] J. Gao, Y. Shen, J. Liu, M. Ito, and N. Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *arXiv preprint arXiv:1705.02755*, 2017.

[24] W. Genders and S. Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.

[25] W. Grzechca. Assembly line balancing problem with reduced number of workstations. *IFAC Proceedings Volumes*, 47(3):6180–6185, 2014.

[26] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016.

[27] M. S. A. Hameed and A. Schwung. Reinforcement learning on job shop scheduling problems using graph networks. *arXiv preprint arXiv:2009.03836*, 2020.

[28] J. Heger, H. Bani, and B. Scholz-Reiter. Improving production scheduling with machine learning. *Artificial Intelligence and Logistics*, page 43, 2012.

[29] J. Huang, W. Wang, L. Wang, H. Chen, Q. Deng, H. Fan, and Y. Yu. Application of deep reinforcement learning in optimization of traffic signal control. In *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 1958–1964. IEEE, 2021.

[30] INDUSTRY 4.0 MARKET - GROWTH, TRENDS, COVID-19 IMPACT, AND FORECASTS (2023 - 2028). `https://www.mordorintelligence.com/industry-reports/industry-4-0-market/`. Accessed: 2020.

[31] S. Jafari, Z. Shahbazi, and Y.-C. Byun. Improving the performance of single-intersection urban traffic networks based on a model predictive controller. *Sustainability*, 13(10):5630, 2021.

[32] N. Kamarudin and M. A. Rashid. Modelling of simple assembly line balancing problem type 1 (salbp-1) with machine and worker constraints. In *Journal of Physics: Conference Series*, volume 1049, page 012037. IOP Publishing, 2018.

[33] H.-H. Kao, D.-H. Yeh, Y.-H. Wang, et al. Resource constrained assembly line balancing problem solved with ranked positional weight rule. *Review of Economics & Finance*, 1:71–80, 2011.

[34] S. Krauß. Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. *Hauptabteilung Mobilität und Systemtechnik des DLR Köln, ISSN 1434-8454*, 1998.

[35] A. Kuhnle, N. Röhrig, and G. Lanza. Autonomous order dispatching in the semiconductor industry using reinforcement learning. *Procedia CIRP*, 79:391–396, 2019.

[36] A. Kuhnle, L. Schäfer, N. Stricker, and G. Lanza. Design, implementation and evaluation of reinforcement learning for an adaptive order dispatching in job shop manufacturing systems. *Procedia CIRP*, 81:234–239, 2019.

[37] M. S. Kumar, R. D. Raut, V. S. Narwane, and B. E. Narkhede. Applications of industry 4.0 to overcome the covid-19 operational challenges. *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, 14(5):1283–1289, 2020.

[38] L. Li, Y. Lv, and F.-Y. Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016.

[39] M. Li-xin, X. Hai-hong, and Z. Jian-bo. Assembly process reengineering applied to production line balancing. In *2009 International Conference on Electronic Commerce and Business Intelligence*, pages 95–98. IEEE, 2009.

[40] X. Liang, X. Du, G. Wang, and Z. Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019.

[41] F. Liberati. Model predictive control of a road junction. *Smart Cities*, 3(3):806–817, 2020.

[42] F. Liberati, A. Tortorelli, C. Mazquiaran, M. Imran, and M. Panfili. Optimal control of industrial assembly lines. In *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 721–726. IEEE, 2020.

[43] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[44] Z. Ma, T. Cui, W. Deng, F. Jiang, and L. Zhang. Adaptive optimization of traffic signal timing via deep reinforcement learning. *Journal of Advanced Transportation*, 2021, 2021.

[45] Business Fortune Insight, "Market Research Report," 2020. `https://www.reportlinker.com/market-report/Marketing/452579/Marketing-Research/`. Accessed: 2023.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[47] M. C. O. Moreira, C. Miralles, and A. M. Costa. Model and heuristics for the assembly line worker integration and balancing problem. *Computers & Operations Research*, 54:64–73, 2015.

[48] M. C. O. Moreira, R. Pastor, A. M. Costa, and C. Miralles. The multi-objective assembly line worker integration and balancing problem of type-2. *Computers & Operations Research*, 82:114–125, 2017.

[49] M. C. O. Moreira, R. Pastor, A. M. Costa, and C. Miralles. The multi-objective assembly line worker integration and balancing problem of type-2. *Computers & Operations Research*, 82:114–125, 2017.

[50] D. Octavian, V. Gurgu, E. Minca, A. Filipescu, F. Dragomir, and O. DRAGOMIR. Optimal control of the complete assembly/disassembly cycle for a mechatronics line prototype. In *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*, pages 620–625. IEEE, 2019.

[51] Open aigym. `https://gym.openai.com/`. Accessed: 2010-09-30.

[52] adm optimizer. `https://keras.io/api/optimizers/adam/`. Accessed: 2010-09-30.

[53] U. Özcan. Balancing and scheduling tasks in parallel assembly lines with sequence-dependent setup times. *International Journal of Production Economics*, 213:81–96, 2019.

[54] U. Özcan, H. Gökçen, and B. Toklu. Balancing parallel two-sided assembly lines. *International Journal of Production Research*, 48(16):4767–4784, 2010.

[55] E. Papadonikolaki, R. Vrijhoef, and H. Wamelink. Supply chain integration with bim: a graph-based model. *Structural Survey*, 2015.

[56] J. Pereira. Modelling and solving a cost-oriented resource-constrained multi-model assembly line balancing problem. *International Journal of Production Research*, 56(11):3994–4016, 2018.

[57] J. Pereira. Modelling and solving a cost-oriented resource-constrained multi-model assembly line balancing problem. *International Journal of Production Research*, 56(11):3994–4016, 2018.

[58] S. J. Qin and T. A. Badgwell. An overview of industrial model predictive control technology. In *AIche symposium series*, volume 93, pages 232–256. New York, NY: American Institute of Chemical Engineers, 1971-c2002., 1997.

[59] M. Rabani, M. Yazdanbakhsh, and H. Farrokhi-Asl. Solving a multi-objective mixed-model assembly line balancing and sequencing problem. *Journal of Industrial and Systems Engineering*, 10(special issue on production and inventory):155–170, 2017.

[60] M. Rabani, M. Yazdanbakhsh, and H. Farrokhi-Asl. Solving a multi-objective mixed-model assembly line balancing and sequencing problem. *Journal of Industrial and Systems Engineering*, 10(special issue on production and inventory):155–170, 2017.

[61] A. Rasheed, O. San, and T. Kvamsdal. Digital twin: Values, challenges and enablers from a modeling perspective. *Ieee Access*, 8:21980–22012, 2020.

[62] J. B. Rawlings. Tutorial overview of model predictive control. *IEEE control systems magazine*, 20(3):38–52, 2000.

[63] M. M. Razali, M. F. F. A. Rashid, and M. R. A. Make. Mathematical modelling of mixed-model assembly line balancing problem with resources constraints. In *IOP Conference Series: Materials Science and Engineering*, volume 160, page 012002. IOP Publishing, 2016.

[64] Y. C. F. Reyna, Y. M. Jiménez, J. M. B. Cabrera, and B. M. M. Hernández. A reinforcement learning approach for scheduling problems. *Investigación Operacional*, 36(3):225–231, 2015.

[65] N. Rida, M. Ouadoud, A. Hasbi, and S. Chebli. Adaptive traffic light control system using wireless sensors networks. In *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)*, pages 552–556. IEEE, 2018.

[66] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *arXiv preprint arXiv:2109.11978*, 2021.

[67] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[68] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, P. Engel, and M. Harnisch. Industry 4.0: The future of productivity and growth in manufacturing industries. *Boston consulting group*, 9(1):54–89, 2015.

[69] H. Sarimveis, P. Patrinos, C. D. Tarantilis, and C. T. Kiranoudis. Dynamic modeling and control of supply chain systems: A review. *Computers & operations research*, 35(11):3530–3561, 2008.

[70] A. Scholl, M. Fliedner, and N. Boysen. Absalom: Balancing assembly lines with assignment restrictions. *European Journal of Operational Research*, 200(3):688–701, 2010.

[71] A. Seric and D. Winkler. Covid-19 could spur automation and reverse globalisation–to some extent. *CEPR Policy Portal [en línea] https://voxeu. org/article/covid-19-could-spur-automation-and-reverseglobalisation-some-extent*, 2020.

[72] H. D. Sherali and W. P. Adams. *Reformulation–Linearization Techniques for Discrete Optimization Problems*, pages 2849–2896. Springer New York, New York, NY, 2013.

[73] S. I. Tay, T. Lee, N. Hamid, and A. N. A. Ahmad. An overview of industry 4.0: Definition, components, and government initiatives. *Journal of Advanced Research in Dynamical and Control Systems*, 10(14):1379–1387, 2018.

[74] T. E. Thomas, J. Koo, S. Chaterji, and S. Bagchi. Minerva: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations. In *2018 10th international conference on communication systems & networks (COMSNETS)*, pages 129–136. IEEE, 2018.

[75] N. T. Thomopoulos et al. *Assembly line planning and control*. Springer, 2014.

[76] A. Tortorelli, M. Imran, F. Delli Priscoli, and F. Liberati. A parallel deep reinforcement learning framework for controlling industrial assembly lines. *Electronics*, 11(4):539, 2022.

[77] R. Wang, X. Ma, C. Jiang, Y. Ye, and Y. Zhang. Heterogeneous information network-based music recommendation system in mobile networks. *Computer Communications*, 150:429–437, 2020.

[78] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia Cirp*, 72:1264–1269, 2018.

[79] Y. Wu and R. Y. Zhong. Manpower planning for mtr carriage assembly with an integer programming. In *LISS 2020*, pages 703–715. Springer, 2021.

[80] B.-L. Ye, W. Wu, K. Ruan, L. Li, T. Chen, H. Gao, and Y. Chen. A survey of model predictive control methods for traffic signal control. *IEEE/CAA Journal of Automatica Sinica*, 6(3):623–640, 2019.

[81] L. C. Zammit, S. G. Fabri, and K. Scerri. Self-tuning model predictive control for signalized traffic junctions. In *2021 European Control Conference (ECC)*, pages 2585–2590. IEEE, 2021.

[82] T. Zhang, S. Xie, and O. Rose. Real-time job shop scheduling based on simulation and markov decision processes. In *2017 Winter Simulation Conference (WSC)*, pages 3899–3907. IEEE, 2017.

[83] T. Zhang, S. Xie, and O. Rose. Real-time batching in job shops based on simulation and reinforcement learning. In *2018 Winter simulation conference (WSC)*, pages 3331–3339. IEEE, 2018.

[84] W. Zhang and T. Dietterich. High-performance job-shop scheduling with a time-delay td (λ) network. *Advances in neural information processing systems*, 8, 1995.

[85] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[86] W. Zhang and M. Gen. An efficient multiobjective genetic algorithm for mixed-model assembly line balancing problem considering demand ratio-based cycle time. *Journal of Intelligent Manufacturing*, 22(3):367–378, 2011.

[87] Y. Zhang, L.-f. Tao, and F. Ju. Balancing of mixed-model assembly line based on ant colony optimization algorithm. In *2011 IEEE 18th International Conference on Industrial Engineering and Engineering Management*, pages 898–901. IEEE, 2011.

[88] M. Zweben, E. Davis, B. Daun, and M. J. Deale. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1588–1596, 1993.

Appendix A

Appendix

This chapter presents further implementation details of the proposed algorithms and code related to the transportation use cases. Unfortunately, the code associated with the SESAME project is the property of all partners and is not reported for confidentiality reasons.

A.1 Proposed Control Algorithm

This section presents the control algorithm and its implementation in Julia.

Algorithm 1 MPC control of a road junction

1: **for** every time $k = 0, 1, ...$ **do**
2: Measure or estimate the current state of the road junction (as defined next).
3: Build and solve an optimal control problem (call it $\mathcal{P}_k$) to determine the optimal *Traffic Light (TL)* control signals to apply at the current time k, i.e., $g_{k,j}$, $y_{k,j}$, $r_{k,j}$ (green, yellow, and red signals). Problem $\mathcal{P}_k$ is defined over the time window $[k, ..., k + N - 1]$, i.e., it is meant to optimize the near future performance of the intersection.
4: Actuate the found optimal TL control signals $g_{k,j}$, $y_{k,j}$, $r_{k,j}$ over the interval $[kT, (k + 1)T]$.
5: **end for**

A.1.1 MPC main Iteration Code

```
c
#=
Note:
mpc_iteration_code
=#

using JuMP, Gurobi

function mpc_iteration!(g_0::AbstractVector , y_0::AbstractVector ,
                        r_0::AbstractVector , t_y_0::AbstractVector ,
                        n_0::AbstractVector , v_in_P::AbstractMatrix ,
                        v_esc::AbstractVector , T::Float64 , N::Int64 ,
                        alfa::Float64 , beta::Float64 , gamma::Float64 ,
```

```
                        n_traffic_lights::Int64, l_media::Float64 ,
                        t_y_min::Float64 , t_y_max::Float64,
                        g_opt_k::AbstractVector, y_opt_k::AbstractVector ,
                        r_opt_k::AbstractVector , n_bar_opt_k::AbstractVector ,
g2y_opt_k::AbstractVector , s_opt_k::AbstractVector,
                        n_vehicles::Int64, n_q_0::AbstractMatrix,
                        s_q_opt_k::AbstractMatrix, n_plus_0::AbstractMatrix,
                        w_0::AbstractMatrix, q_k::AbstractMatrix,
                        p_bar_opt_k::AbstractMatrix )

# T  tempo di campionamento in secondi
# N lenghezza dela finestra MPC (in numero di campioni)
# t_y_min, t_y_max durata minima e massima del giallo in secondi
# alpha, beta, gamma coefficienti funzione obiettivo
# n_traffic_lights numero di semafori
# l_media lunghezza media occupata dai veicoli in metri

# n_vehicles numero veicoli in coda

M = 1.0e6



#-------------Define Model and Solver
mdl = Model(Gurobi.Optimizer) # Presolve = -1, OutputFlag = 0 , MIPGap = 1e-2)
set_optimizer_attribute(mdl, "Presolve", -1)
set_optimizer_attribute(mdl, "OutputFlag", 0)
      set_optimizer_attribute(mdl, "MIPGap" , 1e-2)
# Variabili green, yellow, red (1) - ok
@variable(mdl, g[collect(1:N),collect(1:n_traffic_lights)],Bin)
@variable(mdl, y[collect(1:N),collect(1:n_traffic_lights)],Bin)
@variable(mdl, r[collect(1:N),collect(1:n_traffic_lights)],Bin)

# processed vehicles
@variable(mdl, proc_vehicles[collect(1:N),collect(1:n_traffic_lights)] >= 0 )

# constraint (1) - ogni semaforo è o giallo o verde o rosso - ok
@constraint(mdl, [i=1:N, j=1:n_traffic_lights], g[i,j] + y[i,j] + r[i,j] == 1)

# vincolo per inibire le transizioni non corrette, tipo da giallo a verde (2-4) - ok
@constraint(mdl, [j=1:n_traffic_lights] , r[1,j] <= 1 - g_0[j] )
@constraint(mdl, [j=1:n_traffic_lights] , y[1,j] <= 1 - r_0[j] )
@constraint(mdl, [j=1:n_traffic_lights] , g[1,j] <= 1 - y_0[j] )
```

```
@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , r[i+1,j] <= 1 - g[i,j] )
@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , y[i+1,j] <= 1 - r[i,j] )
@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , g[i+1,j] <= 1 - y[i,j] )

# Variabili durata giallo - ok
@variable(mdl, t_y[collect(1:N),collect(1:n_traffic_lights)]>=0)

# variabile Booleana indicatrice di scatto green to yellow - ok
@variable(mdl, g2y[collect(1:N),collect(1:n_traffic_lights)],Bin)

# definizione variabile di scatto green to yellow g2y (6) - ok
epsilon = 0.1

@constraint(mdl, [j=1:n_traffic_lights] , y[1,j] - y_0[j] <= (1+epsilon)*g2y[1,j] )

@constraint(mdl, [j=1:n_traffic_lights] , (1+epsilon)*g2y[1,j] <= y[1,j] - y_0[j] + 1 )

@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , y[i+1,j] - y[i,j] <= (1+epsilon)*g2y[i+
@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , (1+epsilon)*g2y[i+1,j] <= y[i+1,j] - y[

# equazione dinamica durata giallo (5) - vedere bene i tempi qui - ok
@constraint(mdl, [j=1:n_traffic_lights] , t_y[1,j] == t_y_0[j] )
@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , t_y[i+1,j] == t_y[i,j] + T*y[i,j] - t_y

# vincolo su durata minima di giallo - ok
@constraint(mdl, [i = 1:N , j=1:n_traffic_lights] , 1 - t_y[i,j]/t_y_min <= y[i,j] )

# numero veicoli in coda (8) - ok - messo vincolo positività - attenzione, non è intero
@variable(mdl, n[collect(1:N),collect(1:n_traffic_lights)] >= 0 )

# indicatore di coda (=1 se c'è più di un veicolo in coda) - ok
@variable(mdl, n_bar[collect(1:N),collect(1:n_traffic_lights)],Bin)

# vincolo di indicatore di coda (10) - ok
@constraint(mdl, [i = 1:N , j=1:n_traffic_lights] , n_bar[i,j] <= M*n[i,j] )
@constraint(mdl, [i = 1:N , j=1:n_traffic_lights] , n[i,j] <= M*n_bar[i,j] )

# definizione variabile slack in (14) - ok - AGGIUNTO >= 0
@variable(mdl, s[collect(1:N),collect(1:n_traffic_lights)] >= 0 )

# vincolo di dinamica coda (14) - nota slack variable per tenere n positivo
@constraint(mdl, [j=1:n_traffic_lights] , n[1,j] ==  n_0[j] )
#@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , n[i+1,j] == n[i,j] + v_in[j] - v_in[j]
```

```
        - v_esc[j]*n_bar[i,j]*g[i,j] + s[i,j]  )
@constraint(mdl, [i = 1:N-1 , j=1:n_traffic_lights] , n[i+1,j] == n[i,j] + v_in_P[j] - v_in_P
        - v_esc[j]*n_bar[i,j]*g[i,j] + s[i,j] )

# processed vehicles definition - ok
@constraint(mdl, [i = 1:N , j=1:n_traffic_lights] , proc_vehicles[i,j]
        == v_in_P[j]*(1-n_bar[i,j])*g[i,j] + v_esc[j]*n_bar[i,j]*g[i,j] - s[i,j]     )

# variabile lunghezza coda in (15) - ok
#@variable(mdl, l[collect(1:N),collect(1:n_traffic_lights)] >= 0 )

# vincolo definizione lunghezza coda (15) - ok
 # @constraint(mdl, [i = 1:N , j=1:n_traffic_lights] , l[i,j] == n[i,j]*l_media )

# variabile average waiting time in (16) - ok
#@variable(mdl, d[collect(1:N) , collect(1:n_traffic_lights)] >= 0  )

# vincolo definizione tempo medio attesa (16) - v out
# @constraint(mdl, [i = 1:N , j=1:n_traffic_lights] , d[i,j] == n[i,j]/v_esc[j]*T    )

# conflict avoiding constraints - ok
# semafori A,B,D, cioè indici 1,2,4
@constraint(mdl, [i = 1:N ] , g[i,1] + y[i,1] + g[i,2] + y[i,2] + g[i,4] + y[i,4] <= 1 )
# semafori A,C, cioè indici 1,3
@constraint(mdl, [i = 1:N ] , g[i,1] + y[i,1] + g[i,3] + y[i,3] <= 1 )
# semafori B,E, cioè indici 2,5
@constraint(mdl, [i = 1:N ] , g[i,2] + y[i,2] + g[i,5] + y[i,5] <= 1 )


######## ESTENSIONE WAITING TIME #########

#variabile posizione del veicolo nella coda
@variable(mdl, n_q[collect(1:N) , collect(1:n_traffic_lights), collect(1:n_vehicles)] )#  >= (
@constraint(mdl, [j=1:n_traffic_lights, v=1:n_vehicles] , n_q[1,j,v] == n_q_0[j,v] )



# variabile tempi di attesa
#@variable(mdl, w[i = collect(1:N) , j = collect(1:n_traffic_lights), v
        = collect(1:n_vehicles); q_k[j,v] > 0] >= 0  )
#@constraint(mdl, [j=1:n_traffic_lights, v=1:n_vehicles; (q_k[j,v] > 0)] , w[1,j,v] ==  w_0[j

@variable(mdl, w[i = collect(1:N) , j = collect(1:n_traffic_lights), v = collect(1:n_vehicles
```

```
@constraint(mdl, [j=1:n_traffic_lights, v=1:n_vehicles] , w[1,j,v] ==  w_0[j,v] )

# n_plus
@variable(mdl, n_plus[collect(1:N) , collect(1:n_traffic_lights), collect(1:n_vehicles)], Bin

@constraint(mdl, [j = 1:n_traffic_lights, v=1:n_vehicles] , n_plus[1,j,v] == n_plus_0[j,v])

#p_bar
@variable(mdl, p_bar[collect(1:N) , collect(1:n_traffic_lights), collect(1:n_vehicles)], Bin


# s_q variable which avoids n_q to be negative
@variable(mdl, s_q[collect(1:N) , collect(1:n_traffic_lights), collect(1:n_vehicles)] >= 0)


## n_plus (13) - (17) ##


#eq (13) - ok -
@constraint(mdl, [i = 1:N-1 ,j = 1:n_traffic_lights, v=1:n_vehicles] , n_plus[i+1,j,v] <= n_p


#eq (14) - ok -
@constraint(mdl, [i = 1:N-1 ,j = 1:n_traffic_lights, v=1:n_vehicles] , n_plus[i,j,v] + y[i,j]


#eq (15) - ok -
@constraint(mdl, [i = 1:N,  j = 1:n_traffic_lights, v=1:n_vehicles], (n_q[i,j,v] - v_esc[j])/

#eq (16) - ok -
@constraint(mdl, [i = 1:N,j = 1:n_traffic_lights, v=1:n_vehicles], (v_esc[j] - n_q[i,j,v])/M

#eq (17)
@constraint(mdl, [i = 1:N-1, j = 1:n_traffic_lights, v=1:n_vehicles], -M*(2 - n_plus[i,j,v] -


#eq (17)
@constraint(mdl, [i = 1:N-1, j = 1:n_traffic_lights, v=1:n_vehicles], n_plus[i+1,j,v] <= p_ba



#eq (11)
```

```
#@constraint(mdl, [i = 1:N-1, j = 1:n_traffic_lights, v = 1:n_vehicles; (q_k[j,v] > 0)], w[i+
@constraint(mdl, [i = 1:N-1, j = 1:n_traffic_lights, v = 1:n_vehicles], w[i+1,j,v] == (w[i,j,

#eq (18) - ok -
@constraint(mdl, [i = 1:N-1 , j = 1:n_traffic_lights, v = 1:n_vehicles], n_q[i+1,j,v] == n_q[



# aggiungere veicoli processati

# Eq. (1) - ok - aggiunta penalità slack variable
secondary_opt_param = 0.0
@objective(mdl, Min, sum(sum(n[i,j]^2  -  0.0*proc_vehicles[i,j]^2  + M*s[i,j]  for j=1:n_tra
        #   + sum(sum(sum(w[i,j,v]^2   for v = 1:n_vehicles ) for i = 1:N ) for j = 1:n_traff
#@objective(mdl, Min, secondary_opt_param*sum(g[i,1] + g[i,4] + r[i,5] + r[i,2] + r[i,3] for
        +  gamma*l[i,j]^2 + 10e5*s[i,j]^2  for j=1:n_traffic_lights )  for i = 1:N )  )
#@objective(mdl, Min, sum(sum(sum(M*w[i,j,v]^2 + M*s_q[i,j,v] for v = 1:n_vehicles) for i = 1
optimize!(mdl)
for j=1:n_traffic_lights
g_opt_k[j] = round( Int64 , value(g[1,j]) )
y_opt_k[j] = round( Int64 , value(y[1,j]) )
r_opt_k[j] = round( Int64 , value(r[1,j]) )
n_bar_opt_k[j] = round( Int64 , value(n_bar[1,j]) )
g2y_opt_k[j] = round( Int64 , value(g2y[1,j]) )
s_opt_k[j] = round( Int64 , value(s[1,j]) )


for v=1:n_vehicles
p_bar_opt_k[j,v] = round(Int64, value(p_bar[1,j,v]))
s_q_opt_k[j,v] = round(Int64, value(s_q[1,j,v]))
end
end
end
```

Car Generation

```
% Cars queue

using Distributions

T = 5.0 # tempo campionamento in secondi
```

```
duration_simulation = round(Int64 , 3*60*60/T) # 1 giorno


#---- Dati incrocio
n_traffic_lights = 5
l_media = 4.0 # spazio medio occupato da veicolo in coda
t_y_max = 100.0 # durata massima giallo
t_y_min = 5.0 # durata minima giallo [secondi?]

n_vehicles = 10
v_esc = [2;2;2;2;2] # vettore dei flussi di uscita dei veicoli nelle code (macchine/secondo)

g_opt_k = ones(n_traffic_lights, 1)

#---- Create the scenario
# lambda è il numero medio di macchine ogni T secondi.
lambda_basso = 0.2
lambda_medio = 0.8
lambda_alto = 1.5

# Poisson arrival rates

#= -- simulazione giornata
lambda = lambda_basso*ones(n_traffic_lights,round(Int64 , 24*60*60/T))
lambda[:,round(Int64 , 6*60*60/T):round(Int64 , 24*60*60/T)] .= lambda_medio
lambda[:,round(Int64 , 12*60*60/T):round(Int64 , 14*60*60/T)] .= lambda_alto
lambda[:,round(Int64 , 18*60*60/T):round(Int64 , 21*60*60/T)] .= lambda_alto
=#

#-- simulo 3 ore
lambda = lambda_basso*ones(n_traffic_lights,duration_simulation)
lambda[:,round(Int64 , 1*60*60/T):round(Int64 , 2*60*60/T)] .= lambda_medio
lambda[:,round(Int64 , 2*60*60/T):round(Int64 , 3*60*60/T)] .= lambda_alto

# green_proportional_gain = [0.65 0.43 0.43 0.25 1] Quelli di seguito sono i parametri per il
#Gli altri sono moltiplicati per un fattore che smorza, dato che hanno un verde più corto.
lambda[1,:] = 0.65*lambda[1,:]
lambda[2,:] = 0.43*lambda[2,:]
lambda[3,:] = 0.43*lambda[3,:]
lambda[4,:] = 0.25*lambda[4,:]

n_q = zeros(Int64, n_traffic_lights, duration_simulation, n_vehicles)
```

```
# sample Poisson distribution
v_in_P = zeros(Int64, n_traffic_lights, duration_simulation)
for k = 1:duration_simulation
for j=1:n_traffic_lights
v_in_P[j,k] = rand(Poisson(lambda[j,k]))
end
end

punt = ones(Int64, n_traffic_lights, duration_simulation)
global pos = 1
for k = 1:duration_simulation-1
for j=1:n_traffic_lights
for v=1:n_vehicles
#n_q[j,k+1,v] =  n_q[j,k,v] - 1*v_esc[j]*g_opt_k[j]


if v_in_P[j,k] > 0
global x = v_in_P[j,k]
while x > 0
for punt = pos:pos + x
n_q[j,k,punt] = punt
x -= 1
end
end

end
end
end


end

# SECONDA POSSIBILITà (più veloce)-- not ok --

# while n_p_bar_opt_k[j,v] == 0 # se l'auto è uscita
# cont[j,k] += 1 # contatore delle macchine che lasciano l'incrocio all'istante k
# queue_j = deleteat!(n_q_hystory[j,k,:], v) # elimina le auto che hanno lasciato l'incrocio

# if n_q_hystory[j,k,v] > 0

# n_q_hystory[j,k,:] - cont # corregge gli indici
# push!(n_q_hystory[j,k,v], 0)
```

```
# n_q_hystory[j,k,:] = queue_j

# end
# end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Car Processing Function

```
% Plot processed cars

function plot_processed_cars(processed_cars::AbstractArray , T::Float64)

rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
font0 = Dict(
"font.size" => 16,
"axes.labelweight" => "normal",
"axes.labelsize" => 16,
"xtick.labelsize" => 16,
"ytick.labelsize" => 16,
"legend.fontsize" => 16,
)
merge!(rcParams, font0)

fig = figure("processed_cars",figsize=(10,10))

#PyPlot.plot(processed_cars')
PyPlot.plot(processed_cars[1,:] , marker="None" , color="black")
PyPlot.plot(processed_cars[2,:] , marker="None" , linestyle = "dashed" , color="black")
PyPlot.plot(processed_cars[3,:] , marker="1" , color="black")
PyPlot.plot(processed_cars[4,:] , marker="+" , color="black")
PyPlot.plot(processed_cars[5,:] , marker="x" , color="black")

legend(["1", "2", "3", "4", "5"])

legend(["1", "2", "3", "4", "5"])

xticks(0:round(Int64,60*60/T):size(processed_cars,2))
xlim(0,size(processed_cars,2))

ax = gca()
ylabel("Time Step")
ylabel("Processed cars")

fig.canvas.draw()
```

```
plt.show()
end
```

Car Queue

```
% plot queue %

function plot_queue(n_hystory::AbstractArray , T::Float64)

rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
font0 = Dict(
"font.size" => 16,
"axes.labelweight" => "normal",
"axes.labelsize" => 16,
"xtick.labelsize" => 16,
"ytick.labelsize" => 16,
"legend.fontsize" => 16,
)
merge!(rcParams, font0)



fig = figure("n_hystory",figsize=(10,10))

#PyPlot.plot(n_hystory') ,marker="None"
PyPlot.plot(n_hystory[1,:] , marker="None" , color="black")
PyPlot.plot(n_hystory[2,:] , marker="None" , linestyle = "dashed" , color="black")
PyPlot.plot(n_hystory[3,:] , marker="1" , color="black")
PyPlot.plot(n_hystory[4,:] , marker="+" , color="black")
PyPlot.plot(n_hystory[5,:] , marker="x" , color="black")

legend(["1", "2", "3", "4", "5"])

xticks(0:round(Int64,60*60/T):size(n_hystory,2))
xlim(0,size(n_hystory,2))

ax = gca()
ylabel("Time Step")
ylabel("Lenght of the queues")

fig.canvas.draw()

plt.show()
```

```
end
```

A.1.2 Ploting functions

```
% Plot.jl file %

using PyPlot
rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
font0 = Dict(
        "font.size" => 16,
        "axes.labelweight" => "normal",
        "axes.labelsize" => 16,
        "xtick.labelsize" => 16,
        "ytick.labelsize" => 16,
        "legend.fontsize" => 16,
)
merge!(rcParams, font0)
################################################################




fig = figure("pyplot_subplot_column",figsize=(10,10))

subplot(511)


PyPlot.plot(g_hystory[1,:],color="green")
PyPlot.plot(y_hystory[1,:],color="yellow")
PyPlot.plot(r_hystory[1,:],color="red")

legend(["\$g\$", "\$y\$", "\$r\$"])

xticks(0:1:size(g_hystory[1,:],2))
xlim(0,size(g_hystory[1,:],2))

ax = gca()
ylabel("State 1")



subplot(512)
```

```
PyPlot.plot(g_hystory[2,:],color="green")
PyPlot.plot(y_hystory[2,:],color="yellow")
PyPlot.plot(r_hystory[2,:],color="red")

legend(["\$g\$", "\$y\$", "\$r\$"])

xticks(0:1:size(g_hystory[1,:],2))
xlim(0,size(g_hystory[1,:],2))

ax = gca()
ylabel("State 1")


subplot(513)

PyPlot.plot(g_hystory[3,:],color="green")
PyPlot.plot(y_hystory[3,:],color="yellow")
PyPlot.plot(r_hystory[3,:],color="red")

legend(["\$g\$", "\$y\$", "\$r\$"])

xticks(0:1:size(g_hystory[1,:],2))
xlim(0,size(g_hystory[1,:],2))

ax = gca()
ylabel("State 1")

subplot(514)

PyPlot.plot(g_hystory[4,:],color="green")
PyPlot.plot(y_hystory[4,:],color="yellow")
PyPlot.plot(r_hystory[4,:],color="red")

legend(["\$g\$", "\$y\$", "\$r\$"])

xticks(0:1:size(g_hystory[1,:],2))
xlim(0,size(g_hystory[1,:],2))

ax = gca()
ylabel("State 1")

subplot(515)
```

```
PyPlot.plot(g_hystory[5,:],color="green")
PyPlot.plot(y_hystory[5,:],color="yellow")
PyPlot.plot(r_hystory[5,:],color="red")

legend(["\$g\$", "\$y\$", "\$r\$"])

xticks(0:1:size(g_hystory[1,:],2))
xlim(0,size(g_hystory[1,:],2))

ax = gca()
ylabel("State 1")
fig.canvas.draw()

plt.show()


% Plot showing

function plot_solving_times(input_cars::AbstractArray , T::Float64)

rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
font0 = Dict(
"font.size" => 16,
"axes.labelweight" => "normal",
"axes.labelsize" => 16,
"xtick.labelsize" => 16,
"ytick.labelsize" => 16,
"legend.fontsize" => 16,
)
merge!(rcParams, font0)

fig = figure("solving_times",figsize=(10,10))

PyPlot.plot(solving_times , marker="None" , color="black")

xticks(0:200:length(solving_times))
xlim(0,length(solving_times))

ylim(0,15)

ax = gca()
xlabel("Time Step")
ylabel("Solving times [s]")
```

```
grid("on")

fig.canvas.draw()

plt.show()
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% plot time left

function plot_time_left(tasks_duration_left::AbstractArray)

rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
font0 = Dict(
"font.size" => 16,
"axes.labelweight" => "normal",
"axes.labelsize" => 16,
"xtick.labelsize" => 16,
"ytick.labelsize" => 16,
"legend.fontsize" => 16,
)
merge!(rcParams, font0)

fig = figure("tasks_duration_left",figsize=(10,10))

for i = 1:size(tasks_duration_left,1)
PyPlot.plot(tasks_duration_left[i,:] , marker="None" )
end

# legend(["1", "2", "3", "4", "5"])

#xticks(0:round(Int64,60*60/T):size(tasks_duration_left,2))
xlim(0,size(tasks_duration_left,2))

ax = gca()
ylabel("Time Step")
ylabel("Processed cars")

fig.canvas.draw()

plt.show()
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% plot_traffic_light


function plot_traffic_light(g_hystory::AbstractArray , y_hystory::AbstractArray , r_hystory::

rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
font0 = Dict(
"font.size" => 16,
"axes.labelweight" => "normal",
"axes.labelsize" => 16,
"xtick.labelsize" => 16,
"ytick.labelsize" => 16,
"legend.fontsize" => 16,
)
merge!(rcParams, font0)



fig = figure("processed_traffic_light",figsize=(10,10))

PyPlot.plot(g_hystory[n,:],color="green" ,drawstyle = "steps")
PyPlot.plot(y_hystory[n,:],color="yellow" , drawstyle = "steps")
PyPlot.plot(r_hystory[n,:],color="red"  , drawstyle = "steps" , linestyle = "dashed" )

xticks(0:round(Int64,60*60/T):size(g_hystory,2))
xlim(0,size(g_hystory,2))

ax = gca()
ylabel("Time Step")
ylabel("Status traffic light $n")
fig.canvas.draw()

plt.show()
end
```

A.2 Learning algorithms

This section presents the *RL* algorithm and implementation code. A brief description of each class and function is provided below:

Algorithm 2 DQN Algorithm

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights Θ
Initialize target action-value function $\hat{Q}$ with weights $\Theta^{-} = \Theta$
for episode=1, M **do**
  Initialise sequence $s_1 = x_1$ and pre-processed sequence $\phi_1 = \phi(s_1)$
  for t=1, T **do**
    With probability ϵ select a random action a_t otherwise select $a_t = argmax_a Q(\phi(s_t), a; 0)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step} j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \Theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_i - Q(\phi_j, a_j, \Theta))^2$ with respect to the network parameters Θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

- *buildmodel*: Generates the neural network specifying the minimization method and the optimizer
- *predict_one*: Returns the action predicted from a single state
- *predict_batch*: Returns the action predicted from a batch of states
- *train_batch*: Passes the data into the Model and updates
- *save_model*: Saves the trained Model into the selected path
- *load_my_model*: Loads the .h5 trained model for testing purpose
- *add_sample*: Adds sample to memory; if full deletes the older one
- *get_sample*: Extracts n batch samples from memory
- *run*: Called at the beginning of each episode to start SUMO simulation
- *set_yellow$_p$hase*: Sets the yellow phase if the current action is different from the previous one
- *set_green_phase*: Sets the new phase according to the ϵ greedy policy
- *get_state*: Extracts the state information from the simulation environment
- *collect_waiting_time*: Returns the cumulative waiting time of the intersection

- *get_queue_length*: Returns the number of halting vehicles at the intersection
- *get_CO2Emissions*: Returns the total CO_2 emitted from the intersection
- *choose_action*: Returns the action to perform according to the ϵ greedy policy
- *simulate*: Provides the steps to do for each chose phase
- *replay*: Experience Replay implementation
- *set_sumo*: Sets SUMO.exe path
- *routes*: Implements the vehicle generation

%%

```
import os
import sys
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # kill warning about tensorflow
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import losses
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import load_model


class Model:

def __init__(self, num_layers, width, batch_size, learning_rate,
input_dim, output_dim):
    self.batch_size = batch_size
    self.learning_rate = learning_rate
    self.input_dim = input_dim
    self.output_dim = output_dim
    self.model = self.build_model(num_layers, width)
    self.model.summary()

# Generating the neural network: Refering to: https://keras.io/api/models/model/
def build_model(self, num_layers, width):
    inputs = keras.Input(shape=(self.input_dim,))
    x = layers.Dense(width, activation='relu')(inputs)
    for _ in range(num_layers):
        x = layers.Dense(width, activation='relu')(x)
```

```
    outputs = layers.Dense(self.output_dim, activation='linear')(x)

    model = keras.Model(inputs=inputs, outputs=outputs, name='my_model')
    model.compile(loss=losses.mean_squared_error,

    optimizer=Adam(learning_rate=self.learning_rate))
    return model

# Function which predicts the action from a single state
def predict_one(self, state):
    state = np.reshape(state, [1, self.input_dim])
    return self.model.predict(state)

# Function which predicts the action from a batch of states
def predict_batch(self, states):
    return self.model.predict(states)

    #  https: // www.educba.com / keras - fit /
def train_batch(self, states, q_sa):
    self.model.fit(states, q_sa, epochs=1, verbose=0)

# SAVING MODEL

def save_model(self, path):
    self.model.save(os.path.join(path, 'trained_model.h5'))
    plot_model(self.model, to_file=os.path.join(path, 'model_structure.png'),
               show_shapes=True, show_layer_names=True)



class Model_test:

def __init__(self, input_dim, model_path):
    self.input_dim = input_dim
    self.model = self.load_my_model(model_path)


def load_my_model(self, model_folder_path):

    model_file_path = os.path.join(model_folder_path, 'trained_model.h5')

    if os.path.isfile(model_file_path):
        loaded_model = load_model(model_file_path)
```

```
        return loaded_model
    else:
        sys.exit("Model number not found")


def predict_one(self, state):

    state = np.reshape(state, [1, self.input_dim])
    return self.model.predict(state)

import random


class Memory:
def __init__(self, size_max, size_min):
    self.samples = []
    self.size_max = size_max
    self.size_min = size_min

# Function which add a sample into memory
def add_sample(self, sample):
    self.samples.append(sample)

    if len(self.samples) > self.size_max:
        self.samples.pop(0)
        print("Sample: ", self.samples)

# Function which extracts n (batch size) samples from memory
def get_samples(self, n):

    if len(self.samples) < self.size_min:
        return []

    if n > len(self.samples):
        return random.sample(self.samples, len(self.samples))  # get all the samples
    else:
        return random.sample(self.samples, n)  # get "batch size" number of samples




import random
import timeit
```

```
import numpy as np
import traci




# Dictionary of all the actions

PHASE1G2R3R4R5G5 = 0  # TL1 Green, TL2 Red, TL3 Red, TL4 Red, TL5 Green for 5 seconds
PHASE1Y2R3R4R5Y5 = 1
PHASE1R2G3G4R5R5 = 2
PHASE1R2Y3Y4R5R5 = 3
PHASE1R2R3G4G5G5 = 4
PHASE1R2R3Y4Y5Y5 = 5


class Simulation:

def __init__(self, Model, Memory, fullTrafficGenerator, sumoCmd, max_steps, num_actions,

num_states, training_epochs, gamma, total_episodes):

    self.Model = Model
    self.Memory = Memory

    self.fullTrafficGenerator = fullTrafficGenerator

    self.sumoCmd = sumoCmd

    self.max_steps = max_steps
    self.num_actions = num_actions
    self.num_states = num_states
    self.training_epochs = training_epochs
    self.gamma = gamma

    self.total_episodes = total_episodes

    # PLOTS

    self.reward_store = []  # Cumulative Reward
    self.avg_queue_length_store = []  # Average queue length
    self.cumulative_wait_store = []  # Cumulative wait store
    self.avg_waiting_time_store = []  # Average waited time for each car
```

```
    self.cumulative_CO2Emission_store = []
    self.cumulative_fuelCons_store = []

# Function which is called for each episode starting sumo and then the simulation

def run(self, episode, epsilon):

    start_sim_time = timeit.default_timer()  # Starting timer

    self.fullTrafficGenerator.routes(seed=episode)

    # Starting SUMO
    traci.start(self.sumoCmd)  # Simulation starts
    print("Simulating...")

    # Inits
    self.step = 0
    self.waiting_times = {}


    self.sum_queue_length = 0
    self.sum_waiting_time = 0
    self.sum_neg_reward = 0
    self.sum_actions = 0

    self.sum_CO2Emissions = 0
    self.sum_fuelConsum = 0
    old_total_wait = 0
    old_state = -1
    old_action = -1


    while self.step < self.max_steps:
        # Get Current State of the environment
        current_state = self.get_state()

        #  Get current total waiting time
        current_total_wait = self.collect_waiting_times()

        # Compute reward
        reward = (old_total_wait - current_total_wait)
```

```
        # Saving this data into the Memory
        if self.step != 0:
            self.Memory.add_sample((old_state, old_action, reward, current_state))

        # Select the action to be performed based on epsilon policy
        action = self.choose_action(current_state, epsilon)

        # Selecting the yellow phase if the new action is different from the old one
        if self.step != 0 and old_action != action:
            self.set_yellow_phase(old_action)
            yellow_dur = traci.trafficlight.getPhaseDuration("J7")
            self.simulate(yellow_dur)

        # Executing the selected action
        self.set_green_phase(action)
        green_dur = traci.trafficlight.getPhaseDuration("J7")
        self.simulate(green_dur)

        # print("Training...")
        start_train_time = timeit.default_timer()
        self.replay()
        training_time = round(timeit.default_timer() - start_train_time, 1)

        # Updating the state, action and reward
        old_state = current_state
        old_action = action
        old_total_wait = current_total_wait

        if reward < 0:
            self.sum_neg_reward += reward

    # SAVE EPISODE STATS

    self.reward_store.append(self.sum_neg_reward)
    self.cumulative_wait_store.append(self.sum_waiting_time)
    self.avg_queue_length_store.append(self.sum_queue_length / self.max_steps)
    self.avg_waiting_time_store.append(self.sum_waiting_time /

    self.fullTrafficGenerator.total_num_vehic)

    self.cumulative_CO2Emission_store.append(self.sum_CO2Emissions)
    self.cumulative_fuelCons_store.append(self.sum_fuelConsum)
```

```
    print("Negative Reward: ", self.sum_neg_reward, "Epsilon: ", epsilon)
    traci.close()  # Simulation ends

    simulation_time = round(timeit.default_timer() - start_sim_time, 1)

    return simulation_time, training_time


def set_yellow_phase(self, old_action):
    yellow_phase_code = (old_action * 2) + 1
    traci.trafficlight.setPhase("J7", yellow_phase_code)
    traci.trafficlight.setPhase("J9", yellow_phase_code)
    traci.trafficlight.setPhase("J12", yellow_phase_code)

# Setting the new phase chosen based on the epsilon greedy policy
def set_green_phase(self, action_number):
    if action_number == 0:
        traci.trafficlight.setPhase("J7", PHASE1G2R3R4R5G5)
        traci.trafficlight.setPhase("J9", PHASE1G2R3R4R5G5)
        traci.trafficlight.setPhase("J12", PHASE1G2R3R4R5G5)

    elif action_number == 1:
        traci.trafficlight.setPhase("J7", PHASE1R2G3G4R5R5)
        traci.trafficlight.setPhase("J9", PHASE1R2G3G4R5R5)
        traci.trafficlight.setPhase("J12", PHASE1R2G3G4R5R5)

    elif action_number == 2:
        traci.trafficlight.setPhase("J7", PHASE1R2R3G4G5G5)
        traci.trafficlight.setPhase("J9", PHASE1R2R3G4G5G5)
        traci.trafficlight.setPhase("J12", PHASE1R2R3G4G5G5)

def get_state(self):
    state = np.zeros(self.Model.input_dim)

    #  Halting vehicles at each TL
    state[0] = traci.lane.getLastStepHaltingNumber("E2TL1_0")
    state[1] = traci.lane.getLastStepHaltingNumber("N2TL23_1")
    state[2] = traci.lane.getLastStepHaltingNumber("N2TL23_0")
    state[3] = traci.lane.getLastStepHaltingNumber("W2TL45_1")
    state[4] = traci.lane.getLastStepHaltingNumber("W2TL45_0")

    #  Waiting time at each TL
```

```
    state[5] = traci.lane.getWaitingTime("E2TL1_0")
    state[6] = traci.lane.getWaitingTime("N2TL23_1")
    state[7] = traci.lane.getWaitingTime("N2TL23_0")
    state[8] = traci.lane.getWaitingTime("W2TL45_1")
    state[9] = traci.lane.getWaitingTime("W2TL45_0")

    #  Medium Arrival rate
    state[10] = self.fullTrafficGenerator.lane_1/(self.fullTrafficGenerator.total_num_vehic)
    state[11] = self.fullTrafficGenerator.lane_2/(self.fullTrafficGenerator.total_num_vehic)
    state[12] = self.fullTrafficGenerator.lane_3/(self.fullTrafficGenerator.total_num_vehic)
    state[13] = self.fullTrafficGenerator.lane_4/(self.fullTrafficGenerator.total_num_vehic)
    state[14] = self.fullTrafficGenerator.lane_5/(self.fullTrafficGenerator.total_num_vehic)
    #print("state", state)
    return state

def collect_waiting_times(self):

    incoming_roads = ["E2TL1", "N2TL23", "W2TL45"]
    car_list = traci.vehicle.getIDList()
    for car_id in car_list:
        wait_time = traci.vehicle.getAccumulatedWaitingTime(car_id)
        road_id = traci.vehicle.getRoadID(car_id)
        if road_id in incoming_roads:
            self.waiting_times[car_id] = wait_time
        else:
            if car_id in self.waiting_times:
                del self.waiting_times[car_id]
    total_waiting_time = sum(self.waiting_times.values())
    return total_waiting_time

# Counts the number of cars halting, i.e, the cars with speed < 0.1m/s

def get_queue_length(self):

    halt_N = traci.edge.getLastStepHaltingNumber("N2TL23")
    halt_E = traci.edge.getLastStepHaltingNumber("E2TL1")
    halt_W = traci.edge.getLastStepHaltingNumber("W2TL45")
    queue_length = halt_N + halt_E + halt_W
    return queue_length

def get_CO2Emissions(self):
    Emission_N = traci.edge.getCO2Emission("N2TL23")
    Emission_E = traci.edge.getCO2Emission("E2TL1")
```

```
    Emission_W = traci.edge.getCO2Emission("W2TL45")
    CO2Emission = Emission_N + Emission_E + Emission_W
    return CO2Emission

def get_fuelCons(self):
    fuelCons_N = traci.edge.getFuelConsumption("N2TL23")
    fuelCons_E = traci.edge.getFuelConsumption("E2TL1")
    fuelCons_W = traci.edge.getFuelConsumption("W2TL45")
    fuelConsum = fuelCons_N + fuelCons_E + fuelCons_W
    return fuelConsum


def choose_action(self, state, epsilon):

    if random.random() < epsilon:
        return random.randint(0, self.num_actions - 1)  # random action
    else:
        return np.argmax(self.Model.predict_one(state))  # the best action given the current

def simulate(self, steps_todo):
    if (self.step + steps_todo) >= self.max_steps:
        steps_todo = self.max_steps - self.step

    while steps_todo > 0:
        traci.simulationStep()
        self.step += 1
        steps_todo -= 1

        self.sum_queue_length += self.get_queue_length()
        self.sum_waiting_time += self.collect_waiting_times()

        self.sum_CO2Emissions += self.get_CO2Emissions()
        self.sum_fuelConsum += self.get_fuelCons()


def replay(self):

    batch = self.Memory.get_samples(self.Model.batch_size)

    if len(batch) > 0:  # If the memory is sufficiently full
        states = np.array([val[0] for val in batch])
        next_states = np.array([val[3] for val in batch])
```

```
        # PREDICTION

        q_s_a = self.Model.predict_batch(states)
        q_s_a_d = self.Model.predict_batch(next_states)

        # TRAINING ARRAYS
        x = np.zeros((len(batch), self.num_states))
        y = np.zeros((len(batch), self.num_actions))

        for i, b in enumerate(batch):
            state, action, reward, _ = b[0], b[1], b[2], b[3]

            current_q = q_s_a[i]
            current_q[action] = reward + self.gamma * np.amax(q_s_a_d[i])

            x[i] = state
            y[i] = current_q

        self.Model.train_batch(x, y)  # TRAINS THE NEURAL NETWORK
```

A.2.1 SUMO Interface

```
import os
import sys
from sumolib import checkBinary

# Setting the sumo-gui path

max_steps = 10800


def set_sumo(gui):

if 'SUMO_HOME' in os.environ:
    tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
    sys.path.append(tools)
else:
    sys.exit("please declare environment variable 'SUMO_HOME'")

# setting the cmd mode or the visual mode
if gui == False:
    sumoBinary = checkBinary('sumo')
else:
```

```
    sumoBinary = checkBinary('sumo-gui')

# setting the cmd command to run sumo at simulation time
sumo_cmd = [sumoBinary, "-c", 'main.sumocfg', "--no-step-log", "true",

"--waiting-time-memory", str(max_steps)]

return sumo_cmd


import os
import sys


def set_train_path(models_path_name):

models_path = os.path.join(os.getcwd(), models_path_name, '')
os.makedirs(os.path.dirname(models_path), exist_ok=True)

dir_content = os.listdir(models_path)
if dir_content:
    previous_versions = [int(name.split("_")[1]) for name in dir_content]
    new_version = str(max(previous_versions) + 1)
else:
    new_version = '1'

data_path = os.path.join(models_path, 'model_'+new_version, '')
os.makedirs(os.path.dirname(data_path), exist_ok=True)
return data_path


def set_test_path(models_path_name, model_n):

model_folder_path = os.path.join(os.getcwd(), models_path_name, 'model_'+str(model_n), '')

if os.path.isdir(model_folder_path):
    plot_path = os.path.join(model_folder_path, 'test', '')
    os.makedirs(os.path.dirname(plot_path), exist_ok=True)
    return model_folder_path, plot_path
else:
    sys.exit('The model number specified does not exist in the models folder')
```

A.2.2 Visualization

```
import matplotlib.pyplot as plt
import os


class Visualization:

def __init__(self, path, dpi):
    self._path = path
    self._dpi = dpi

def save_data_and_plot(self, data, filename, xlabel, ylabel):

    min_val = min(data)
    max_val = max(data)

    plt.rcParams.update({'font.size': 24})  # set bigger font size

    plt.plot(data)
    plt.ylabel(ylabel)
    plt.xlabel(xlabel)
    plt.margins(0)
    plt.ylim(min_val - 0.05 * abs(min_val), max_val + 0.05 * abs(max_val))
    fig = plt.gcf()
    fig.set_size_inches(20, 11.25)
    fig.savefig(os.path.join(self._path, 'plot_' + filename + '.png'), dpi=self._dpi)
    plt.close("all")

    with open(os.path.join(self._path, 'plot_' + filename + '_data.txt'), "w") as file:
        for value in data:
            file.write("%s\n" % value)



import timeit
import numpy as np
import traci

# Dictionary of all the actions

PHASE1G2R3R4R5G5 = 0
PHASE1Y2R3R4R5Y5 = 1
PHASE1R2G3G4R5R5 = 2
```

```
PHASE1R2Y3Y4R5R5 = 3
PHASE1R2R3G4G5G5 = 4
PHASE1R2R3Y4Y5Y5 = 5


class Simulation_test:

def __init__(self, Model, fullTrafficGenerator, sumoCmd, max_steps,

num_actions, num_states):

    self.Model = Model

    self.fullTrafficGenerator = fullTrafficGenerator

    self.sumoCmd = sumoCmd

    self.max_steps = max_steps
    self.num_actions = num_actions
    self.num_states = num_states

    self.reward_store = []
    self.queue_length_episodeLow = []
    self.queue_length_episodeMed = []
    self.queue_length_episodeHig = []

    self.queue_length_episode = []

    self.wait_time = []


    self.cumulative_wait_storeLow = []
    self.cumulative_wait_storeMed  = []
    self.cumulative_wait_storeHig = []

    self.cumulative_wait_store = []

    self.avg_queue_length_store = []
    self.avg_waiting_time_store = []
    self.actions = []

    self.ep = 0
```

```
# Function which is called for each episode starting sumo and then the simulation

def run(self, episode):

    start_sim_time = timeit.default_timer()

    self.fullTrafficGenerator.routes(seed=episode)

    # Starting SUMO
    traci.start(self.sumoCmd)
    print("Simulating...")

    # Inits
    self.step = 0
    self.waiting_times = {}

    old_total_wait = 0
    old_action = -1

    while self.step < self.max_steps:
        # Get Current State of the environment
        current_state = self.get_state()


        current_total_wait = self.collect_waiting_times()

        # print("Current total waiting time: ", current_total_wait)

        # Compute reward
        reward = (old_total_wait - current_total_wait)

        # Select the action to be performed based on epsilon policy
        action = self.choose_action(current_state)

        # Selecting the yellow phase if the new action is different from the old one
        if self.step != 0 and old_action != action:
            self.set_yellow_phase(old_action)
            yellow_dur = traci.trafficlight.getPhaseDuration("J7")
            self.simulate(yellow_dur)
            # self.switch += 1
            # print("Number of switches: ", self.switch)

        # Executing the selected action
```

```
            self.set_green_phase(action)
            green_dur = traci.trafficlight.getPhaseDuration("J7")
            self.simulate(green_dur)


            old_action = action
            old_total_wait = current_total_wait

            self.reward_store.append(reward)

        traci.close()

        simulation_time = round(timeit.default_timer() - start_sim_time, 1)

        return simulation_time


def set_yellow_phase(self, old_action):
    yellow_phase_code = (old_action * 2) + 1
    traci.trafficlight.setPhase("J7", yellow_phase_code)
    traci.trafficlight.setPhase("J9", yellow_phase_code)
    traci.trafficlight.setPhase("J12", yellow_phase_code)

# Setting the new phase
def set_green_phase(self, action_number):
    if action_number == 0:
        traci.trafficlight.setPhase("J7", PHASE1G2R3R4R5G5)
        traci.trafficlight.setPhase("J9", PHASE1G2R3R4R5G5)
        traci.trafficlight.setPhase("J12", PHASE1G2R3R4R5G5)

    elif action_number == 1:
        traci.trafficlight.setPhase("J7", PHASE1R2G3G4R5R5)
        traci.trafficlight.setPhase("J9", PHASE1R2G3G4R5R5)
        traci.trafficlight.setPhase("J12", PHASE1R2G3G4R5R5)

    elif action_number == 2:
        traci.trafficlight.setPhase("J7", PHASE1R2R3G4G5G5)
        traci.trafficlight.setPhase("J9", PHASE1R2R3G4G5G5)
        traci.trafficlight.setPhase("J12", PHASE1R2R3G4G5G5)

def get_state(self):
    state = np.zeros(self.Model.input_dim)
```

```
    state[0] = traci.lane.getLastStepHaltingNumber("E2TL1_0")
    state[1] = traci.lane.getLastStepHaltingNumber("N2TL23_1")
    state[2] = traci.lane.getLastStepHaltingNumber("N2TL23_0")
    state[3] = traci.lane.getLastStepHaltingNumber("W2TL45_1")
    state[4] = traci.lane.getLastStepHaltingNumber("W2TL45_0")

    state[5] = traci.lane.getWaitingTime("E2TL1_0")
    state[6] = traci.lane.getWaitingTime("N2TL23_1")
    state[7] = traci.lane.getWaitingTime("N2TL23_0")
    state[8] = traci.lane.getWaitingTime("W2TL45_1")
    state[9] = traci.lane.getWaitingTime("W2TL45_0")

    #  Medium Arrival rate
    state[10] = self.fullTrafficGenerator.lane_1 /

    (self.fullTrafficGenerator.total_num_vehic)
    state[11] = self.fullTrafficGenerator.lane_2 /

    (self.fullTrafficGenerator.total_num_vehic)
    state[12] = self.fullTrafficGenerator.lane_3 /

    (self.fullTrafficGenerator.total_num_vehic)
    state[13] = self.fullTrafficGenerator.lane_4 /

    (self.fullTrafficGenerator.total_num_vehic)
    state[14] = self.fullTrafficGenerator.lane_5 /

    (self.fullTrafficGenerator.total_num_vehic)

    # print("state", state)
    return state


def collect_waiting_times(self):

    incoming_roads = ["E2TL1", "N2TL23", "W2TL45"]
    car_list = traci.vehicle.getIDList()
    for car_id in car_list:
        wait_time = traci.vehicle.getAccumulatedWaitingTime(car_id)
        road_id = traci.vehicle.getRoadID(car_id)
        if road_id in incoming_roads:
            self.waiting_times[car_id] = wait_time
        else:
```

```
            if car_id in self.waiting_times:
                del self.waiting_times[car_id]
    total_waiting_time = sum(self.waiting_times.values())
    return total_waiting_time

# Counts the number of cars halting, i.e, the cars with speed < 0.1m/s
def get_queue_length(self):

    halt_N = traci.edge.getLastStepHaltingNumber("N2TL23")
    halt_E = traci.edge.getLastStepHaltingNumber("E2TL1")
    halt_W = traci.edge.getLastStepHaltingNumber("W2TL45")
    queue_length = halt_N + halt_E + halt_W
    return queue_length


def choose_action(self, state):

        return np.argmax(self.Model.predict_one(state))  # the best action given the current

def simulate(self, steps_todo):
    if (self.step + steps_todo) >= self.max_steps:
        steps_todo = self.max_steps - self.step

    while steps_todo > 0:
        traci.simulationStep()
        self.step += 1
        steps_todo -= 1

        queue_length = self.get_queue_length()
        waiting_time = self.collect_waiting_times()

        if self.step <= 3600:
            self.queue_length_episodeLow.append(queue_length)
            self.cumulative_wait_storeLow.append(waiting_time)
        elif self.step > 3600 and self.step <= 7200:
            self.queue_length_episodeMed.append(queue_length)
            self.cumulative_wait_storeMed.append(waiting_time)
        elif self.step >= 7200 and self.step <= 10800:
            self.queue_length_episodeHig.append(queue_length)
            self.cumulative_wait_storeHig.append(waiting_time)

        #self.cumulative_wait_store.append(waiting_time)
```

```
import numpy as np


class fullTrafficGenerator:

def routes(self, seed):

    np.random.seed(seed)

    self.Low1 = np.random.poisson(0.13, 720)
    self.Low2 = np.random.poisson(0.086, 720)
    self.Low3 = np.random.poisson(0.086, 720)
    self.Low4 = np.random.poisson(0.05, 720)
    self.Low5 = np.random.poisson(0.2, 720)

    self.Med1 = np.random.poisson(0.52, 720)
    self.Med2 = np.random.poisson(0.35, 720)
    self.Med3 = np.random.poisson(0.35, 720)
    self.Med4 = np.random.poisson(0.2, 720)
    self.Med5 = np.random.poisson(0.8, 720)

    self.Hig1 = np.random.poisson(0.975, 720)
    self.Hig2 = np.random.poisson(0.645, 720)
    self.Hig3 = np.random.poisson(0.645, 720)
    self.Hig4 = np.random.poisson(0.375, 720)
    self.Hig5 = np.random.poisson(1.5, 720)

    self.lane_1 = sum(self.Low1) + sum(self.Med1) + sum(self.Hig1)
    self.lane_2 = sum(self.Low2) + sum(self.Med2) + sum(self.Hig2)
    self.lane_3 = sum(self.Low3) + sum(self.Med3) + sum(self.Hig3)
    self.lane_4 = sum(self.Low4) + sum(self.Med4) + sum(self.Hig4)
    self.lane_5 = sum(self.Low5) + sum(self.Med5) + sum(self.Hig5)
    self.total_num_vehic = self.lane_1 + self.lane_2 + self.lane_3 +

    + self.lane_4 + self.lane_5




    self.Low = np.add(self.Low1, self.Low2)
    self.Low = np.add(self.Low, self.Low3)
    self.Low = np.add(self.Low, self.Low4)
    self.Low = np.add(self.Low, self.Low5)
```

```
self.Med = np.add(self.Med1, self.Med2)
self.Med = np.add(self.Med, self.Med3)
self.Med = np.add(self.Med, self.Med4)
self.Med = np.add(self.Med, self.Med5)


self.Hig = np.add(self.Hig1, self.Hig2)
self.Hig = np.add(self.Hig, self.Hig3)
self.Hig = np.add(self.Hig, self.Hig4)
self.Hig = np.add(self.Hig, self.Hig5)

self.Traffic_Arrival = np.concatenate((self.Low, self.Med, self.Hig))

with open("incrocio.rou.xml", "w") as routes:
    print("""<routes>
    <vType id="typeE2TL1" accel="2.6" decel="4.5"  length="4.0"

    minGap="2" maxSpeed="50" sigma="0.5" />
    <vType id="typeN2TL2" accel="2.6" decel="4.5"  length="4.0"

    minGap="2" maxSpeed="50" sigma="0.5" />
    <vType id="typeN2TL3" accel="2.6" decel="4.5"  length="4.0"

    minGap="2" maxSpeed="50" sigma="0.5" />
    <vType id="typeW2TL4" accel="2.6" decel="4.5"  length="4.0"

    minGap="2" maxSpeed="50" sigma="0.5" />
    <vType id="typeW2TL5" accel="2.6" decel="4.5"  length="4.0"

    minGap="2" maxSpeed="50" sigma="0.5" />

    <route id = "E2TL1" edges = "E2TL1 E17 E18 E10"/>
    <route id = "N2TL2" edges = "N2TL23 E7 E12"/>
    <route id = "N2TL3" edges = "N2TL23 E8 E10"/>
    <route id = "W2TL4" edges = "W2TL45 E11 E19 E20 E22"/>
    <route id = "W2TL5" edges = "W2TL45 E11 E12"/>
    """, file=routes)

    veh_num = 0
    indexl = 0
```

```
for i in range(0, 3600, 5):
    vehLow1 = self.Low1[indexl]

    while vehLow1 > 0:
        print(' <vehicle id = "E2TL1_%i" type = "typeE2TL1"

        route = "E2TL1" depart = "%i" />' % (veh_num, i),
              file=routes)
        veh_num += 1
        vehLow1 -= 1

    vehLow2 = self.Low2[indexl]
    while vehLow2 > 0:
        print(' <vehicle id = "N2TL2_%i" type = "typeN2TL2"

        route = "N2TL2" depart = "%i" />' % (veh_num, i),
              file=routes)
        veh_num += 1
        vehLow2 -= 1

    vehLow3 = self.Low3[indexl]
    while vehLow3 > 0:
        print(' <vehicle id = "N2TL3_%i" type = "typeN2TL3"

        route = "N2TL3" depart = "%i" />' % (veh_num, i),
              file=routes)
        veh_num += 1
        vehLow3 -= 1

    vehLow4 = self.Low4[indexl]
    while vehLow4 > 0:
        print(' <vehicle id = "W2TL4_%i" type = "typeW2TL4"

        route = "W2TL4" depart = "%i" />' % (veh_num, i),
          file=routes)
        veh_num += 1
        vehLow4 -= 1

    vehLow5 = self.Low5[indexl]
    while vehLow5 > 0:
        print(' <vehicle id = "W2TL5_%i" type = "typeW2TL5"

        route = "W2TL5" depart = "%i" />' % (veh_num, i),
```

```
            file=routes)
        veh_num += 1
        vehLow5 -= 1
    indexl += 1

indexm = 0
for j in range(3600, 7200, 5):
    vehMed1 = self.Med1[indexm]

    while vehMed1 > 0:
        print(
            ' <vehicle id = "E2TL1_%i" type = "typeE2TL1"

            route = "E2TL1" depart = "%i" />' % (veh_num, j),
            file=routes)
        veh_num += 1
        vehMed1 -= 1

    vehMed2 = self.Med2[indexm]
    while vehMed2 > 0:
        print(
            ' <vehicle id = "N2TL2_%i" type = "typeN2TL2"

            route = "N2TL2" depart = "%i" />' % (veh_num, j),
            file=routes)
        veh_num += 1
        vehMed2 -= 1

    vehMed3 = self.Med3[indexm]
    while vehMed3 > 0:
        print(
            ' <vehicle id = "N2TL3_%i" type = "typeN2TL3"

            route = "N2TL3" depart = "%i" />' % (veh_num, j),
            file=routes)
        veh_num += 1
        vehMed3 -= 1

    vehMed4 = self.Med4[indexm]
    while vehMed4 > 0:
        print(
            ' <vehicle id = "W2TL4_%i" type = "typeW2TL4"
```

```
                route = "W2TL4" depart = "%i" />' % (veh_num, j),
                file=routes)
            veh_num += 1
            vehMed4 -= 1

        vehMed5 = self.Med5[indexm]
        while vehMed5 > 0:
            print(
                ' <vehicle id = "W2TL5_%i" type = "typeW2TL5"

                route = "W2TL5" depart = "%i" />' % (veh_num, j),
                file=routes)
            veh_num += 1
            vehMed5 -= 1
        indexm += 1

    indexh = 0
    for t in range(7200, 10800, 5):
        vehHig1 = self.Hig1[indexh]

        while vehHig1 > 0:
            print(
                ' <vehicle id = "E2TL1_%i" type = "typeE2TL1"

                route = "E2TL1" depart = "%i" />' % (veh_num, t),
                file=routes)
            veh_num += 1
            vehHig1 -= 1

        vehHig2 = self.Hig2[indexh]
        while vehHig2 > 0:
            print(
                ' <vehicle id = "N2TL2_%i" type = "typeN2TL2"

                route = "N2TL2" depart = "%i" />' % (veh_num, t),
                file=routes)
            veh_num += 1
            vehHig2 -= 1

        vehHig3 = self.Hig3[indexh]
        while vehHig3 > 0:
            print(
                ' <vehicle id = "N2TL3_%i" type = "typeN2TL3"
```

```
                route = "N2TL3" depart = "%i" />' % (veh_num, t),
                file=routes)
            veh_num += 1
            vehHig3 -= 1

        vehHig4 = self.Hig4[indexh]
        while vehHig4 > 0:
            print(
                ' <vehicle id = "W2TL4_%i" type = "typeW2TL4"

                route = "W2TL4" depart = "%i" />' % (veh_num, t),
                file=routes)
            veh_num += 1
            vehHig4 -= 1

        vehHig5 = self.Hig5[indexh]
        while vehHig5 > 0:
            print(
                ' <vehicle id = "W2TL5_%i" type = "typeW2TL5"

                route = "W2TL5" depart = "%i" />' % (veh_num, t),
                file=routes)
            veh_num += 1
            vehHig5 -= 1
        indexh += 1

    print("</routes>", file=routes)
```

A.2.3 Main class

```
from __future__ import absolute_import
from __future__ import print_function
from Model import Model_test
from Simulation_test import Simulation_test
from Sumo import set_sumo
from visual import Visualization
from utils import set_test_path
from fullgeneration import fullTrafficGenerator

if __name__ == "__main__":

sumoCmd = set_sumo(True)
```

```
model_path, path_plot = set_test_path(models_path_name='models', model_n=109)

Model = Model_test(input_dim=15, model_path=model_path)

fullTrafficGenerator = fullTrafficGenerator()

Visualization = Visualization(
    path_plot,
    dpi=96
)

Simulation_test = Simulation_test(Model, fullTrafficGenerator, sumoCmd,

max_steps=10800, num_states=15,
                                  num_actions=3)

print('\n----- Test episode')
simulation_time = Simulation_test.run(episode=10000)  # run the simulation
print('Simulation time:', simulation_time, 's')

print("----- Testing info saved at:", path_plot)

Visualization.save_data_and_plot(data=Simulation_test.queue_length_episodeLow,

filename='queueLow', xlabel='Step',
                                  ylabel='Queue length (vehicles)')

Visualization.save_data_and_plot(data=Simulation_test.queue_length_episodeMed,

filename='queueMed', xlabel='Step',
                                  ylabel='Queue length (vehicles)')

Visualization.save_data_and_plot(data=Simulation_test.queue_length_episodeHig,

filename='queueHig', xlabel='Step',
                                  ylabel='Queue length (vehicles)')

Visualization.save_data_and_plot(data=Simulation_test.cumulative_wait_storeLow,

filename='waitingLow', xlabel='Step',
                                  ylabel='Cumulative Waiting Time')
```

```
Visualization.save_data_and_plot(data=Simulation_test.cumulative_wait_storeMed,

filename='waitingMed', xlabel='Step',
                                ylabel='Cumulative Waiting Time')

Visualization.save_data_and_plot(data=Simulation_test.cumulative_wait_storeHig,

filename='waitingHig', xlabel='Step',
                                ylabel='Cumulative Waiting Time')

Visualization.save_data_and_plot(data=fullTrafficGenerator.Traffic_Arrival,

filename='Arrival Generation', xlabel='Step',
                                ylabel='Number of vehicles Arriving')


from __future__ import absolute_import
from __future__ import print_function
from Model import Model
from Memory import Memory
from Simulation import Simulation
from Sumo import set_sumo
from visual import Visualization
from utils import set_train_path
from fullgeneration import fullTrafficGenerator

if __name__ == "__main__":

sumoCmd = set_sumo(True)

path = set_train_path(models_path_name='models')

Model = Model(num_layers=5, width=800, batch_size=32, learning_rate=0.00025,

input_dim=15, output_dim=3)

Memory = Memory(size_min=400, size_max=500000)

fullTrafficGenerator = fullTrafficGenerator()

Visualization = Visualization(
    path,
    dpi=96
```

```
)

Simulation = Simulation(Model, Memory, fullTrafficGenerator, sumoCmd,

gamma=0.5, max_steps=10800, num_states=15,
                        num_actions=3, training_epochs=1, total_episodes=1200)

episode = 0

total_episodes = 1200
# Running the simulation following an epsilon greedy policy

while episode < total_episodes:
    print('\n----- Episode', str(episode+1), 'of', str(total_episodes))
    epsilon = 1.0 - (episode / total_episodes)  # epsilon-greedy policy
    simulation_time, training_time = Simulation.run(episode, epsilon)
    print('Simulation time:', simulation_time, 's - Training time:',

    training_time, 's - Total:', round(simulation_time+training_time, 1), 's')
    episode += 1

Model.save_model(path)
```