PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
الجمهـــوريـــــة الجزائـريـــــة الديـمقـراطيـــــة الشـعـبيـــــة
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
وزارة التعليـم العالـــي والـبحـــث العلمـي
HIGHER SCHOOL OF COMPUTER SCIENCE AND DIGITAL TECHNOLOGIES-BEJAIA
المـــدرسة العـــليا في علـــوم وتكنـولوجيـات الاعـــلام الالـــي بجـــاية

Dissertation Submitted to the Department Of Computer Science in Partial Fulfillment of the Requirements for Engineer's Degree in Computer Science

Specialty: Artificial Intelligence and Data Sciences

Submitted By:**Amani Chaimaa Sellam**

Deep Reinforcement Learning for Vehicle Platooning Optimization

Supervised by:

Dr. Nadir Farhi GRETTIA Lab
Mme. Meriem Bouali ESTIN

Members of the Jury:

- **Pr .** Hamid KHERBACHI — President — ESTIN
- **Dr.** Sid Ali BOUSLA — Examiner — ESTIN
- **Dr.** Soraya TOULOUM — Examiner — ESTIN
- **Mme.** Ouahiba BOUGLIMINA — Examiner — ESTIN

Academic year: 2023/2024

Dedication

First and foremost, I would like to express my deepest gratitude to my parents, who have been my pillars of support throughout this journey. To my father, who has always been my mirror, reflecting the best version of myself and guiding me with wisdom and patience.

To my mother, who has been my greatest cheerleader, encouraging me at every step and never letting me doubt my abilities.

To my sister Tahani, who never misses a chance to drive me crazy, but in doing so, has always kept me grounded and brought joy to my life.

To my brother Mustafa and my sister Maryam, who might not realize this, but they were my strength and my source of force, always motivating me to push forward.

To my grandma, who prayed for me as long as she was breathing. Her prayers and blessings have been a guiding light in my life, and I will forever be grateful for her love and support.

To my dear friend Cheima, who has always been there to lift me up whenever I felt down. Your unwavering support and belief in me have been instrumental in my achievements, and I am forever thankful for your friendship.

Lastly, I would like to thank all my friends who have stood by me through thick and thin. Your companionship and encouragement have made this journey more enjoyable and less daunting.

This work is as much yours as it is mine. Thank you all.

Abstract

Automated vehicle platooning has emerged as a significant method for improving traffic efficiency, reducing fuel consumption, and enhancing road safety. This thesis investigates the optimization of vehicle platooning using reinforcement learning techniques, specifically focusing on Deep Q-Networks (DQN) integrated with dueling networks and prioritized experience replay (PER). A two-layered approach is employed, where the first layer identifies joinable platoons and evaluates the benefits of joining them, and the second layer uses reinforcement learning to optimize merging, lane-changing, and acceleration strategies. The results of the simulation demonstrate that the proposed method significantly reduces travel time, fuel consumption, and improves the overall safety of the platooning process. Future work includes extending the model to mixed traffic environments with both autonomous and human-driven vehicles, as well as integrating predictive traffic models.

Keywords: Automated vehicle platooning, Deep Q-Networks, Dueling networks, Prioritized experience replay, Reinforcement learning, Traffic efficiency, Fuel consumption, Road safety, Merging strategies, Mixed traffic environments.

Résumé

Le peloton de véhicules automatisés est devenu une méthode significative pour améliorer l'efficacité du trafic, réduire la consommation de carburant et renforcer la sécurité routière. Cette thèse examine l'optimisation du peloton de véhicules en utilisant des techniques d'apprentissage par renforcement, en se concentrant spécifiquement sur les réseaux de neurones profonds (DQN) intégrés avec des réseaux de duel et la reprise d'expérience priorisée (PER). Une approche à deux niveaux est utilisée : le premier niveau identifie les pelotons accessibles et évalue les avantages de les rejoindre, tandis que le second niveau utilise l'apprentissage par renforcement pour optimiser les stratégies de fusion, de changement de voie et d'accélération. Les résultats de la simulation montrent que la méthode proposée réduit de manière significative le temps de trajet, la consommation de carburant et améliore la sécurité générale du processus de peloton. Les travaux futurs incluent l'extension du modèle à des environnements de trafic mixte avec des véhicules autonomes et conduits par des humains, ainsi que l'intégration de modèles prédictifs de trafic.

Mots-clés: Peloton de véhicules automatisés, Réseaux de neurones profonds, Réseaux de duel, Reprise d'expérience priorisée, Apprentissage par renforcement, Efficacité du trafic, Consommation de carburant, Sécurité routière, Stratégies de fusion, Environnements de trafic mixte.

الملخص

ظهرت قيادة القوافل الآلية كطريقة مهمة لتحسين كفاءة المرور وتقليل استهلاك الوقود وتعزيز سلامة الطرق. تبحث هذه الأطروحة في تحسين قيادة القوافل الآلية باستخدام تقنيات التعلم المعزز، مع التركيز بشكل خاص المدمجة مع الشبكات المزدوجة وإعادة تشغيل التجارب ذات (DQN) على الشبكات العصبية العميقة يتم استخدام نهج من طبقتين، حيث تتعرف الطبقة الأولى على القوافل القابلة للانضمام وتقيّم .(PER) الأولوية فوائد الانضمام إليها، بينما تستخدم الطبقة الثانية التعلم المعزز لتحسين استراتيجيات الاندماج وتغيير المسار والتسارع. أظهرت نتائج المحاكاة أن الطريقة المقترحة تقلل بشكل كبير من وقت السفر واستهلاك الوقود وتُحسن من أمان عملية قيادة القوافل. تشمل الأعمال المستقبلية توسيع النموذج ليشمل بيئات مرور مختلطة مع وجود كل من المركبات ذاتية القيادة والمركبات التي يقودها البشر، بالإضافة إلى دمج نماذج تنبؤية لحركة .المرور

الكلمات المفتاحية: قيادة القوافل الآلية، الشبكات العصبية العميقة، الشبكات المزدوجة، إعادة تشغيل التجارب ذات الأولوية، التعلم المعزز، كفاءة المرور، استهلاك الوقود، سلامة الطرق، استراتيجيات الاندماج، بيئات .المرور المختلطة

Acronyms

AV Autonomous Vehicle

CACC Cooperative Adaptive Cruise Control

CAV Connected and Automated Vehicle

DRL Deep Reinforcement Learning

DQN Deep Q-Network

LP Linear Programming

MILP Mixed-Integer Linear Programming

MPC Model Predictive Control

PER Prioritized Experience Replay

PID Proportional-Integral-Derivative

QP Quadratic Programming

RL Reinforcement Learning

SUMO Simulation of Urban MObility

V2I Vehicle-to-Infrastructure

V2V Vehicle-to-Vehicle

List of Figures

List of Tables

Contents

Introduction

0.1 Background and Significance of the Study

The goal of this research is to enhance the process of vehicles joining and leaving platoons through automated driving technology. By focusing on the merging and splitting operations, this study aims to improve the efficiency and safety of traffic systems. Specifically, the research explores how Connected and Automated Vehicles (CAVs) can leverage vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication to improve coordination and communication on the road.

CAV systems enable V2V and V2I communication, allowing vehicles to maintain safe distances from one another and better synchronize their speeds. These technologies help reduce accidents and improve traffic flow by minimizing delays and avoiding congestion. The advent of 5G technology has further enhanced communication between vehicles and infrastructure, enabling real-time reaction to traffic conditions. Despite this progress, the vast majority of vehicles are still human-driven, and forecasts predict that by 2040, less than 30% of vehicles will be fully automated. This mixture of human-driven and automated vehicles presents challenges for traffic management, particularly in the context of platooning, which requires precise vehicle coordination.

Platooning refers to a group of vehicles driving closely together in a coordinated manner, following a lead vehicle while maintaining the same speed and reducing inter-vehicle spacing. This method reduces traffic congestion, enhances safety, and optimizes energy consumption. However, the merging and splitting processes associated with platooning remain complex and underexplored. When vehicles attempt to join or leave a platoon, they must coordinate their speeds and distances to prevent collisions. Managing these operations in mixed traffic environments, where both human-driven and automated vehicles coexist, presents additional challenges.

Dynamic platoon organization at the network level is critical. Existing studies largely focus on linear optimization methods that facilitate vehicles joining platoons at specific points on the road, often assuming ideal traffic conditions. Additionally, many studies propose breaking the platoon to allow for merging and splitting. However, breaking a platoon increases safety risks and operational costs, as the platoon must

reorganize, potentially causing delays and reducing the overall effectiveness of platooning.

This research aims to optimize the merging and splitting of vehicles within a platoon without breaking the platoon's structure. By developing a dynamic control strategy, the study seeks to improve vehicle coordination, enabling smooth and safe merging and splitting in mixed traffic environments. The objective is to maintain the integrity of the platoon while ensuring that vehicles can efficiently join or leave.

The merging and splitting processes involve controlling both longitudinal (speed) and lateral (lane change) vehicle dynamics. These operations require real-time communication between vehicles and traffic control systems. While building a reliable optimization model for these processes is challenging, Deep Reinforcement Learning (DRL) offers a viable solution. DRL combines the computational power of neural networks with the decision-making capabilities of reinforcement learning, enabling vehicles to make optimal decisions in complex environments. Although DRL has been applied to other areas of automated driving, its application to platoon merging and splitting remains underexplored.

This study proposes a dynamic platoon control strategy based on DRL to optimize merging and splitting operations in mixed traffic environments. The goal is to develop a system that allows vehicles to safely and efficiently join or leave platoons without disrupting their structure. The system will leverage real-time data and communication to adjust to various traffic conditions while maintaining platoon integrity.

0.2 Current Research Gaps

Despite considerable progress in platooning research, several significant gaps remain:

1. **Merging and Splitting Operations**: Most studies focus on either merging or splitting, but not both. In real traffic scenarios, these operations occur simultaneously and need to be managed together to avoid traffic disruptions.

2. **Breaking Platoons**: Many existing models propose breaking the platoon to allow vehicles to merge or leave. However, this introduces safety risks and delays, as the platoon must reorganize. Solutions that maintain platoon integrity during merging and splitting are needed.

3. **Mixed Traffic Environments**: Much of the current research assumes fully automated traffic. However, most real-world traffic consists of a mix of human-driven and automated vehicles, which introduces unpredictability and additional challenges for platoon management.

0.3 Research Objectives and Contributions

This research aims to address the identified gaps by developing a dynamic platoon control strategy that tackles the challenges of merging and splitting operations in mixed traffic environments. The key contributions of

this study include:

1. Developing a two-layer control strategy that optimizes both merging and splitting operations simultaneously.
2. Designing a DRL algorithm that enables vehicles to safely join or leave platoons without breaking the platoon structure.
3. Testing the model in mixed traffic environments, where human-driven and automated vehicles coexist, ensuring robustness and adaptability to real-world traffic conditions.

0.4 Research Methodology

The research will be conducted in three main stages:

0.4.1 Stage 1: Model Development

A DRL-based model will be developed to manage both longitudinal and lateral vehicle control required for platoon merging and splitting. The model will incorporate real-time data from the traffic environment, such as traffic conditions and vehicle behavior, to make dynamic decisions.

0.4.2 Stage 2: Simulation and Testing

The developed model will be tested on a simulation platform that replicates real-world traffic conditions, including mixed traffic flows. The simulation will evaluate the model's ability to handle merging and splitting operations without disrupting platoon integrity.

0.4.3 Stage 3: Evaluation and Optimization

The model will be evaluated based on several key metrics, including the number of collisions, success rate of merging and splitting operations, and time efficiency. The results will be used to optimize the model further and improve its performance.

0.5 Summary of the Chapter

This research addresses critical gaps in the study of platoon operations by focusing on the optimization of merging and splitting without breaking the platoon. The study proposes a DRL-based control strategy to improve traffic flow, reduce accidents, and ensure safe and efficient vehicle coordination in mixed traffic environments.

Chapter 1

Analysis of the Capacity of Intelligent Networked Hybrid Transportation Environments

1.1 Introduction

The work focuses on the rapidly advancing field of intelligent networked vehicles, which are anticipated to shape the future of road transportation. Before the full integration of autonomous vehicles, there will be a transitional phase where autonomous and traditional vehicles will coexist on the roads. This mixed-vehicle environment presents several challenges, particularly in maintaining smooth and safe traffic flow. The aim of this research is to explore how self-driving vehicles can be organized into groups, known as platoons, which travel together to improve traffic efficiency and safety. By reducing interactions between different types of vehicles, platooning can make the roads safer and more efficient [1–3]. The study further investigates how to improve the algorithms that control these platoons to optimize traffic flow.

Despite the promise of vehicle platooning, the unpredictable behavior of human drivers poses risks and may undermine the benefits of platoon organization. To mitigate these issues, this study suggests better lane organization to separate traditional and autonomous vehicles, which could lead to enhanced traffic safety and flow [4,5]. The concept of dedicated lanes for platoons, or mixed lanes where both vehicle types can operate with well-coordinated platooning algorithms, is a key focus of the research [6]. This aligns with existing studies, which show that mixed traffic environments can suffer from reduced efficiency and increased safety risks if not properly managed [7,8].

1.2 Intelligent Networked Hybrid Transportation Environment

1.2.1 Definition of Networked Autonomous Vehicles

Networked autonomous vehicles are vehicles that can drive themselves and communicate with other vehicles and infrastructure. Autonomous vehicles are typically classified into six levels of autonomy, ranging from level 0 (fully manual) to level 5 (fully autonomous, without human input) [9,10]. As the technology advances, vehicles operating at levels 4 and 5 are expected to dominate, where the vehicle performs all driving tasks autonomously. However, as mixed traffic environments persist, it is essential to develop systems that allow autonomous and traditional vehicles to coexist safely [11,12].

1.2.2 Mixed Traffic Components

In mixed traffic environments, there are three primary components: traditional vehicles, standalone self-driving vehicles, and networked platoons of autonomous vehicles. These components must interact safely and efficiently, but the slower reaction times of human drivers introduce challenges. This study explores how lane organization and dynamic platooning strategies can reduce these challenges and improve overall traffic efficiency [13,14].

In such environments:

- Traditional vehicles are manually driven.
- Standalone self-driving vehicles operate independently.
- Intelligent networked platoons consist of self-driving vehicles that communicate and move together with minimal gaps between them [15].

Platoons are more efficient because they allow vehicles to travel closely together, relying on real-time communication to maintain consistent speeds and reduce headways [16,17]. Without this communication, vehicles cannot coordinate their movements effectively, and closely packed vehicles would not count as a platoon due to the lack of synchronized actions [18,19].

1.3 Platoon Driving Systems

1.3.1 Vehicle Platoon Classification and Key Parameters

Platoons can be classified based on various factors:

- **Vehicle Type**: Platoons may consist of similar vehicles (e.g., all trucks) or a mixture of vehicle types (e.g., trucks and cars).

- **Following Strategy**: Vehicles may follow each other at fixed distances or adjust the distance dynamically based on speed or other factors [9].
- **Information Flow**: Some platoons share information only with the vehicle directly ahead, while others exchange data between all members of the platoon.
- **Platoon Size**: Platoon sizes can vary, but this study focuses on limited platoon sizes, typically 4-10 vehicles [8].

This study focuses on platoons with similar vehicles, fixed following distances, and limited sizes. Limiting platoon size is necessary to ensure manageable coordination and safety, as larger platoons increase the complexity of maintaining synchronized movements and can be more easily disrupted by other traffic [4].

1.3.2 Platoon Organization

Platoon organization refers to the process of forming platoons, which can happen either before a journey begins (static organization) or while vehicles are on the move (dynamic organization). The focus of this study is on dynamic platoon organization, where vehicles can join platoons opportunistically or by actively searching for one while traveling [5]. Research has shown that dynamic organization allows for more flexibility and can better accommodate varying traffic conditions [17].

Examples of Platoon Organization:

- **Example 1**: A group of vehicles forms a platoon before departure, and no other vehicles join during the journey.
- **Example 2**: A vehicle detects an existing platoon while driving and joins it either opportunistically or by actively seeking out a platoon [12].

The strategies for joining a platoon can include:

- **Chase Strategy**: The vehicle accelerates to catch up with the platoon.
- **Deceleration Strategy**: The platoon slows down to allow the vehicle to merge.
- **Mixed Strategies**: A combination of both acceleration and deceleration techniques [15].

In this study, new vehicles always join the platoon at the end, as this simplifies the merging process and reduces disruptions. Prior research has demonstrated that end-of-platoon merging is a more stable and safer method of joining [20].

1.4 Lane Organization in Mixed Traffic Environments

1.4.1 Capacity Analysis

Capacity analysis helps determine how many vehicles can safely and efficiently travel on a lane. This study presents a formula for calculating lane capacity, accounting for various types of vehicles and their headway distances (the space between vehicles). In mixed traffic environments, the average headway distance can vary depending on whether the vehicles are manually driven, autonomous, or part of a platoon [19].

The capacity of a lane can guide decisions on traffic organization. For example, dedicated lanes for platoons may be more efficient in environments where autonomous vehicle adoption is high. Studies have shown that lane segregation based on vehicle type can significantly improve traffic efficiency [10].

1.4.2 Case of Capacity-Based Lane Organization Optimization

The study examines three lane organization strategies under varying traffic conditions:

1. Two mixed lanes: Both lanes allow all types of vehicles.
2. One mixed lane and one lane for traditional vehicles: This segregation reduces interactions between autonomous and traditional vehicles.
3. One platoon-exclusive lane and one mixed lane: A dedicated platoon lane is introduced to maximize the efficiency of autonomous vehicles [8].

The effectiveness of each strategy depends on traffic demand and the proportion of autonomous vehicles. Research has shown that dedicated platoon lanes are highly effective in improving road capacity and reducing congestion when there is a significant presence of autonomous vehicles [9, 13].

1.5 Conclusion

This chapter introduces the foundational concepts of intelligent networked hybrid traffic environments and presents methods for organizing platoons and traffic lanes. It discusses the challenges of mixed traffic and the potential benefits of organizing traffic through platoon formations and dedicated lane systems. These concepts are critical for understanding the next steps in optimizing traffic flow and ensuring road safety in future transportation systems.

Chapter 2

Literature Review

2.1 Introduction

The development of connected and automated vehicle (CAV) platooning has garnered significant attention in recent years, driven by the potential to improve traffic efficiency, reduce fuel consumption, and enhance roadway safety. The concept of vehicle platooning involves a group of vehicles traveling closely together in a coordinated manner, often with a lead vehicle guiding the movement of the following vehicles. This requires sophisticated control strategies and communication protocols to ensure that all vehicles in the platoon operate harmoniously.

The research community has explored various approaches to manage platoon operations, particularly during complex maneuvers such as merging and splitting. Centralized and decentralized protocols represent two primary paradigms in this area, each with its distinct advantages and challenges. Centralized approaches, where a single entity coordinates the actions of the entire platoon, offer high levels of control and optimization but are often limited by scalability and vulnerability to single points of failure. In contrast, decentralized approaches distribute decision-making across individual vehicles, enhancing robustness and scalability but often at the cost of optimal coordination.

Trajectory planning is another critical aspect of platoon management, ensuring that vehicles can execute maneuvers safely and efficiently. Research in this area has produced a variety of methods, from centralized optimization techniques that consider the global state of the platoon to decentralized strategies that rely on local information and heuristic rules.

In addition to traditional control methods, recent advances in machine learning, particularly reinforcement learning (RL), have opened new avenues for optimizing platoon operations. RL offers the potential to learn and adapt strategies directly from data, enabling more flexible and intelligent decision-making in dynamic environments.

This chapter reviews the key contributions in these areas, highlighting the evolution of centralized and decentralized control strategies, the development of trajectory planning methods, and the integration of RL into CAV platooning. By examining the state-of-the-art in these domains, we aim to provide a comprehensive overview of the existing solutions and identify the gaps that this work seeks to address.

2.2 Overview of Vehicle Platooning Research

Connected and Automated Vehicle (CAV) platooning has been a focal point of research due to its potential to improve traffic flow, reduce emissions, and enhance safety on roadways. The concept of vehicle platooning involves coordinating multiple vehicles to travel closely together, benefiting from reduced aerodynamic drag and synchronized driving patterns. This coordination requires sophisticated control strategies, particularly during complex maneuvers such as merging, splitting, and lane changes. The approaches to managing these maneuvers can broadly be categorized into centralized and decentralized systems.

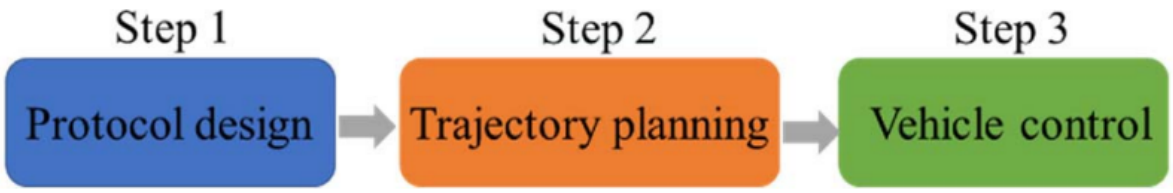


Figure 2.1: The 3-layer framework in literature.

2.2.1 Protocol Design

Protocol design refers to the structured communication and decision-making processes within a platoon. This section discusses both centralized and decentralized approaches to protocol design.

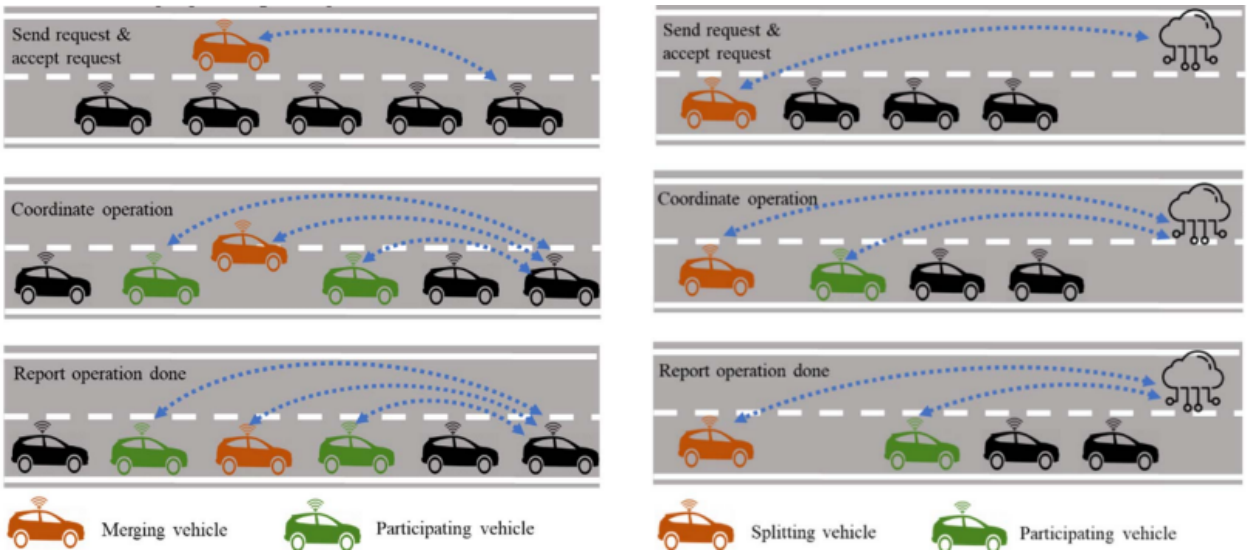
Centralized Protocol Design

Centralized protocol design involves a single entity, typically a central controller, that governs communication and decision-making across the entire platoon. This approach ensures that information is accurately shared and that decisions are executed uniformly.

Algorithm:

```
Algorithm CentralizedProtocolDesign
Input: Platoon P, CommunicationRange r, ControlCommands C
Output: Coordinated Platoon Behavior

Initialize: Central controller establishes communication links with all
    vehicles
Broadcast initial control commands C to all vehicles
for each time step t do
    Central controller collects data from each vehicle (position, speed, etc.)
    Process collected data to assess platoon state
    Determine new control commands based on the global platoon state
    Broadcast updated control commands to all vehicles
    Monitor execution of commands and adjust if necessary
end for
Finalize the coordinated movement and disband communication
```



(a) Centralized Merging (b) Centralized Splitting

Figure 2.2: Centralized Merging and Splitting Operations. These diagrams show how a central controller coordinates the movements of vehicles during merging and splitting, ensuring uniformity across the platoon.

Centralized protocol design is crucial for maintaining the integrity and coordination of the platoon. It involves structured communication protocols where the central controller gathers data from all vehicles, processes it, and then disseminates commands to ensure that the platoon operates cohesively.

Advantages:

- **Unified Command Structure:** Ensures that all vehicles in the platoon operate under a consistent com-

mand strategy.

- **Efficient Communication:** Centralized communication can reduce the complexity of managing inter-vehicle interactions.

Disadvantages:

- **Communication Delays:** Any delays in communication can lead to lapses in coordination, potentially causing safety risks.

- **Dependency on Central Controller:** The entire system relies on the central controller, making it a potential single point of failure.

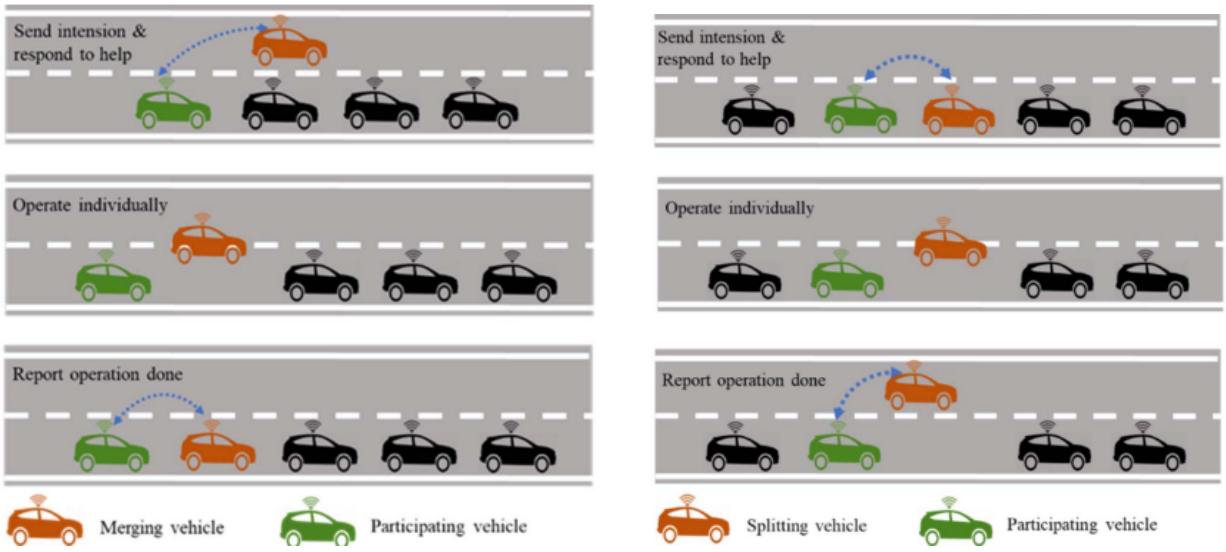
Decentralized Protocol Design

Decentralized protocol design distributes the decision-making process across all vehicles in the platoon. Each vehicle independently communicates with its neighbors and makes decisions based on local information.

Algorithm:

```
Algorithm DecentralizedProtocolDesign
Input: Local vehicle information, Neighboring vehicles' data
Output: Coordinated Local Behavior

Initialize: Each vehicle sets up communication with neighboring vehicles
for each time step t do
    for each vehicle i do
        Receive information from neighboring vehicles (e.g., position, speed)
        Process local and received data to determine vehicle i's state
        Make local decisions (e.g., speed adjustment, lane changes)
        Communicate decisions and state to neighboring vehicles
    end for
end for
Finalize local behavior and adjust as needed based on new information
```



(a) Decentralized Merging (b) Decentralized Splitting

Figure 2.3: Decentralized Merging and Splitting Operations. Each vehicle uses local information and communicates with neighboring vehicles to coordinate actions during merging and splitting.

Decentralized protocol design allows vehicles to independently make decisions based on local information, improving robustness and scalability.

Advantages:

- **Scalability:** The system can easily scale to larger platoons without centralized control.
- **Robustness:** The failure of a single vehicle does not critically impact the entire platoon.

Disadvantages:

- **Suboptimal Coordination:** Since decisions are made locally, the overall platoon coordination may be less optimal.
- **Complexity in Local Algorithms:** Ensuring effective coordination through local decision-making can require complex algorithms.

Comparing Centralized and Decentralized Protocols

Centralized protocols offer significant efficiency due to coordinated actions but depend heavily on Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication. Conversely, decentralized systems distribute decision-making, improving robustness but leading to less optimal performance.

Table 2.1: Management Protocols and Merging/Splitting Positions in Different Studies.

Ref	Management Protocol		Merging Position			Splitting Position		
	Centralized	Decentralized	Head	Middle	Tail	Head	Middle	Tail
[21]	✓		✓		✓		✓	✓
[22]		✓		✓				
[23]		✓		✓				
[24]	✓			✓		✓		
[25]	✓		✓			✓		
[26]		✓				✓	✓	
[27]	✓					✓		
[28]	✓				✓		✓	
[29]	✓					✓	✓	
[30]	✓						✓	
[31]		✓		✓				
[32]		✓	✓			✓		
[33]	✓			✓				
[34]	✓					✓		
[35]		✓	✓	✓		✓	✓	

This table highlights the comparative analysis of different management protocols (centralized vs. decentralized) and their corresponding merging and splitting positions. By reviewing these studies, we can observe a clear advancement in protocol design for platoon management. The comparison reflects the evolution of decision-making in platoons, and this analysis guided the development of further improvements in our project.

2.2.2 Trajectory Planning

Trajectory planning is a critical component of platoon management, ensuring that vehicles can merge, split, and maneuver safely and efficiently. Both centralized and decentralized approaches to trajectory planning are used, with different advantages.

Centralized Trajectory Planning

Centralized trajectory planning leverages the comprehensive view provided by a central controller to optimize the movement paths for all vehicles within a platoon. This approach is advantageous in environments with high traffic density or complex road conditions, where precise coordination among vehicles is crucial for safety and efficiency [15].

The trajectory planning problem can be formulated as an optimization problem. Let P be the set of all vehicles in the platoon, and let $K \subseteq P$ be the subset of vehicles involved in the trajectory planning at a given time. The objective is to minimize a cost function J, which could represent various performance metrics such as fuel consumption, travel time, or deviation from a desired speed [**?**]. The problem can be expressed as:

$$\min_{u_i(t)} J = \sum_{i \in K} \int_0^T f_i(u_i(t), x_i(t), v_i(t), a_i(t))\, dt, \tag{2.1}$$

where:

- $u_i(t)$ is the control input for vehicle i at time t (e.g., acceleration).
- $x_i(t)$ is the position of vehicle i at time t.
- $v_i(t)$ is the speed of vehicle i at time t.
- $a_i(t)$ is the acceleration of vehicle i at time t.
- $f_i(\cdot)$ is a function representing the cost for vehicle i, which may include terms for fuel consumption, speed deviations, and other factors.

The optimization problem (2.1) is solved using numerical methods, and this formulation is adapted from [16].

Solution Techniques: Advanced optimization methods such as Linear Programming (LP), Quadratic Programming (QP), or heuristic methods like genetic algorithms may be employed for trajectory planning [16, 18].

Table 2.2: Centralized Trajectory Planning Literature.

Ref	Merging	Splitting	Two-group	Multiple-group	Solver	Heuristic
[36]	✓		✓		✓	
[37]	✓			✓	✓	
[38]		✓	✓		✓	
[39]	✓			✓		
[40]		✓		✓		
[41]	✓		✓	✓		
[42]	✓	✓				
[43]		✓		✓		
[44]	✓			✓		
[45]	✓			✓		
[46]		✓		✓		
[47]		✓			✓	
[48]		✓				✓
[49]		✓		✓		

This table was created during the analysis of trajectory planning techniques in literature. It highlights the different approaches used in centralized trajectory planning, especially focusing on merging, splitting, and multi-group operations. The work done here helped identify key patterns and gaps in the literature that informed our decisions regarding the trajectory optimization model in this project.

2.3 Q-Learning and Deep Q-Learning

Q-learning is a model-free reinforcement learning method where an agent learns the value of taking a particular action in a given state. The goal is to maximize the cumulative reward by iteratively updating the Q-values [**?**]. In platooning, this allows agents (vehicles) to learn optimal behaviors during complex maneuvers.

Bellman Equation: The core of Q-learning is based on the Bellman equation, which defines the recursive relationship for Q-values:

$$Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s',a')|s,a],$$

where s represents the current state, a is the current action, and r_{t+1} is the reward obtained at time $t+1$. This equation is adapted from [**?**].

2.4 Conclusion

This chapter reviewed various methods of vehicle platooning, trajectory planning, and reinforcement learning applications. The inclusion of centralized and decentralized protocol designs, along with trajectory planning, highlights the complexity and scalability challenges of managing platoon dynamics. Reinforcement learning, particularly Q-learning and its deep-learning variants, provides promising tools to improve these systems, as demonstrated in this research.

Chapter 3

Platoon Organization Optimization : A Double-Layer Approach

3.1 Introduction

This chapter provides a comprehensive overview of the optimization of Connected and Automated Vehicle (CAV) platooning using reinforcement learning (RL) techniques, combined with the tools and frameworks used to simulate these vehicle platoons. By leveraging SUMO (Simulation of Urban Mobility), Plexe, and machine learning libraries, we simulate vehicle behavior in a dynamic environment, optimize platoon management, and employ Deep Q-Networks (DQN) to train the vehicles in a platoon. The goal is to reduce travel time, fuel consumption, and improve overall traffic safety. The chapter also delves into the detailed explanations of the DQN agents used for training and evaluation .

3.2 Platoon Organization and Optimization

3.2.1 Identifying Joinable Platoons

The first step in optimizing platoon organization is determining which platoons a vehicle can join. This is done by evaluating nearby platoons based on communication range, distance, and the vehicle's ability to either join from the front or rear. The set of joinable platoons is defined by the following equation :

$$C_j = \left\{ p \mid S_i \leq rl_p \leq E_i \ \& \ D_p^h \leq D_j \text{ xor } D_j \leq D_p^t \right\}$$

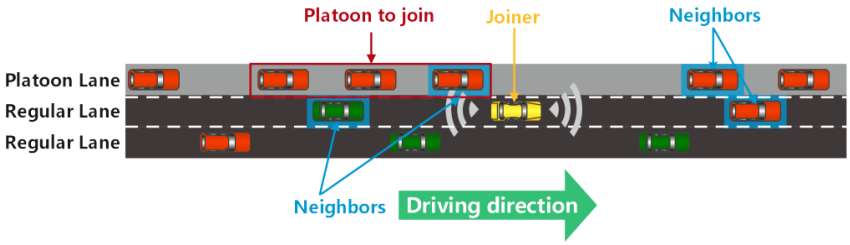


Figure 3.1: Detecting Joinable Platoons: The figure illustrates the process of detecting nearby platoons within the communication range that are suitable for a vehicle to join.

3.2.2 Calculating the Benefit of Joining Each Platoon

Once the set of joinable platoons is identified, the benefit of joining each platoon is calculated using the following formula :

$$S_p^j = TTD_0^j - TTD_p^j - T_{lc}$$

Where S_p^j represents the net benefit for vehicle j if it joins platoon p, accounting for travel time differences and lane change time.

3.2.3 Maximizing Travel Time Savings

The formula for maximizing travel time savings is :

$$TTD_0^j - TTD_p^j = \frac{v_p - v_j}{v_j \cdot v_p} \cdot L_p^j$$

This equation ensures that the vehicle joins the platoon offering maximum time savings based on speed differences and shared trajectory.

3.2.4 Minimizing the Time Taken to Join

Minimizing the time for the vehicle to merge into the platoon is achieved through the equation:

$$T_{lc} = n_{lc} \times t_{lc}$$

Here, n_{lc} is the number of lane changes and t_{lc} is the time for each lane change.

3.2.5 Optimization Algorithm

Algorithm:

```
Algorithm OptimizePlatoonJoining
Input: Vehicle j, Communication Area [S_i, E_i], Set of Platoons P
Output: Optimal platoon to join and joining strategy

Initialize: Set of joinable platoons C_j = {}

Step 1: Identify Joinable Platoons
For each platoon p in P:
    If S_i     rl_p     E_i AND (D_p^h     D_j OR D_j     D_p^t):
        Add platoon p to C_j

Step 2: Calculate Benefit for Each Platoon
Initialize: BestScore = -   , BestPlatoon = NULL
For each platoon p in C_j:
    Calculate TTD_0^j - TTD_p^j = (v_p - v_j) / (v_j * v_p) * L_p^j
    Calculate T_lc = n_lc * t_lc
    Calculate Score S_p^j = (TTD_0^j - TTD_p^j) - T_lc
    If S_p^j > BestScore:
        BestScore = S_p^j
        BestPlatoon = p

Step 3: Decision-Making and Optimization
Use DQN to refine the joining strategy based on the calculated score.

Return BestPlatoon and optimal joining strategy
```

3.3 Merging and Splitting using Reinforcement Learning

3.3.1 States

The state of each vehicle, including its position, speed, and relative position to other vehicles, is captured and used for decision-making in a reinforcement learning framework.

3.3.2 Actions

Actions include acceleration, deceleration, and lane changes that allow the vehicle to merge with or split from a platoon.

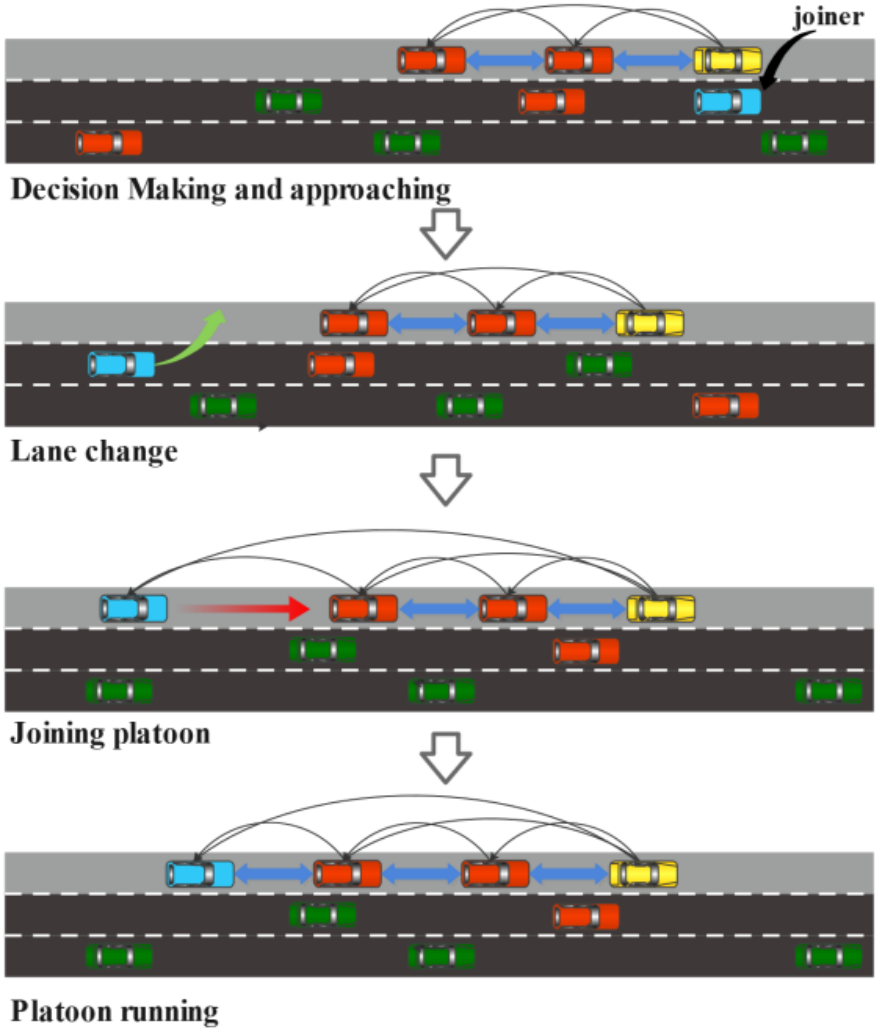


Figure 3.2: The process of joining a platoon: Vehicles perform actions to safely join an existing platoon based on reinforcement learning strategies.

3.3.3 Rewards

Rewards guide the learning process, encouraging behaviors like safe joining, maintaining a minimum distance, and efficient platoon driving .

```
def rew(self):
        # collision = 0
        joiner_id = self.join_list[1]
        platoon_front_id = self.join_list[2]
        do = self.done()
        if joiner_id in self.tlp_list or joiner_id not in traci.vehicle.getIDList():
            r = -100000
```

```
            # collision = 1
        else:
            if do == 1:
                if traci.vehicle.getLeader(joiner_id) and traci.vehicle.getLeader(joiner_id)
    [0] == platoon_front_id:
                    r = 10
                    d = self.get_distance(joiner_id, platoon_front_id)
                    if d < 5:
                        r = -10
                    if d > 20:
                        r = r * 20 / d
                    with open("d.txt", "a") as f:
                        # Writing data to a file
                        f.write(str(d) + '\n')
                else:
                    r = -50
            else:
                r = -1
                v = traci.vehicle.getSpeed(joiner_id)
                o = self.obs()
                if o[7] < MIM_GAP or o[8] < MIM_GAP:  # Too small space with preceding/
    following veh :
                    r -= 10
                if v > 35:
                    r = r - 10
                if v < 15:
                    r = r - 5
        return r
```

The reward function $rew(self)$ calculates a reward r for the vehicle, which is determined based on the following conditions :

The total reward r is returned based on the vehicle's behavior, ensuring that the vehicle optimizes for safe platoon joining, fuel efficiency, and speed control.

3.4 Simulation Tools for Vehicle Platooning

3.4.1 SUMO (Simulation of Urban Mobility)

SUMO simulates vehicle dynamics and traffic flow, providing a realistic environment for testing platoon algorithms .

3.4.2 Plexe

Plexe extends SUMO's capabilities, allowing for detailed control of vehicle platoons, supporting Cooperative Adaptive Cruise Control (CACC), and advanced platoon management .

3.5 Libraries and Machine Learning Frameworks

3.5.1 Python Libraries

Key libraries used include :

- **sumolib** and **traci**: Utilities for handling SUMO simulations.
- **gym**: Provides the environment for reinforcement learning experiments.
- **torch**: Used for training the DQN agents.
- **numpy** and **pandas**: Essential for numerical computations and data analysis.

3.6 Deep Q-Networks for Platoon Optimization

3.6.1 DQN Agent Architecture

The DQN agent approximates Q-values for state-action pairs using a neural network. The architecture includes :

- Input Layer: Receives vehicle state data.
- Hidden Layers: Processes data to compute Q-values.
- Output Layer: Outputs the best actions for platooning, such as acceleration or lane change.

3.7 Dueling DQN, Double DQN, and PER Dueling DQN Architectures

3.7.1 Dueling DQN

The Dueling Deep Q-Network (DQN) introduces two separate estimators: one for the state-value function $V(s)$ and another for the action-advantage function $A(s, a)$. This separation is particularly useful when actions in some states have little influence on the outcome .

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a') \right)$$

3.7.2 Double DQN

Double DQN (DDQN) addresses the issue of overestimation of Q-values by decoupling action selection from action evaluation. The update rule for the Q-values in DDQN is:

$$Q(s,a) = r + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta), \theta^-)$$

This reduces overestimation and results in more accurate learning .

3.7.3 PER Dueling DQN

Prioritized Experience Replay (PER) is an enhancement to DQN that prioritizes experiences based on their importance, calculated from the temporal difference (TD) error:

$$p_i = |\delta_i| + \epsilon$$

The combination of PER with the Dueling DQN architecture ensures faster learning and better sample efficiency by prioritizing important transitions while decoupling state-value and action-advantage estimation .

3.7.4 Application in This Work

In this work, Dueling DQN was initially chosen to decouple state-value and action-advantage estimation for stable learning, followed by the integration of PER to focus learning on critical transitions. This approach proved effective in optimizing the platoon management and overall vehicle performance .

3.7.5 Training Process

The DQN agent interacts with the environment, selects actions, and updates the Q-network using the following loss function :

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta) \right)^2 \right]$$

3.8 Training and Evaluation Scripts

The following is a detailed explanation of the training process for Deep Q-Network (DQN) agents and their evaluation. The process involves the use of replay memory buffer, experience sampling, and periodic updates to the Q-network.

3.8.1 Replay Memory Buffer

One of the key innovations that significantly improved the performance of reinforcement learning (RL) algorithms, particularly DQN, is the use of a replay memory buffer. The replay buffer helps mitigate the correlation between sequential experiences by storing the agent's transitions, which are later randomly sampled for training. This mechanism helps in stabilizing the learning process, making it more efficient.

The replay memory buffer stores each transition tuple (s, a, r, s'), which consists of:

- s: The current state of the agent in the environment.
- a: The action taken by the agent at state s.
- r: The reward received after performing action a.
- s': The next state of the agent after action a is taken.

These transitions are continuously stored in a fixed-size buffer $\mathcal{D}$, with older transitions being removed when the buffer is full. The buffer allows the algorithm to sample mini-batches of transitions during training, breaking the correlation between consecutive experiences, which would otherwise result in inefficient learning and policy updates.

The process of using a replay memory buffer improves the stability and efficiency of DQN training by enabling the following key advantages:

- Breaking Correlations: As experiences in an environment are usually highly correlated (i.e., successive states and actions are not independent), the replay buffer allows the network to break this correlation by sampling transitions randomly.
- Improving Sample Efficiency: Instead of discarding each transition after it is used, the replay buffer allows multiple updates on the same experience, thereby improving sample efficiency.
- Reduced Variance in Gradient Updates: By sampling mini-batches from a large buffer of past experiences, the gradient updates are computed from a broader distribution of states, actions, and rewards, which reduces the variance of updates and leads to more stable learning.

3.8.2 Replay Memory Implementation in DQN

The replay memory buffer is initialized as a fixed-size data structure that stores the most recent transitions experienced by the agent. When the buffer exceeds its capacity, the oldest transitions are removed to make space for the new ones. During training, mini-batches of transitions are randomly sampled from the buffer, and these samples are used to update the Q-network. Below is the code snippet that illustrates how the replay buffer is initialized and used:

```
class ReplayMemoryBuffer:
    def __init__(self, capacity):
        self.capacity = capacity  # Maximum size of the buffer
        self.memory = []          # Stores transitions (s, a, r, s')
        self.position = 0         # Tracks where to insert new transitions

    def push(self, state, action, reward, next_state):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)  # Initialize the memory with empty transitions
        self.memory[self.position] = (state, action, reward, next_state)
        self.position = (self.position + 1) % self.capacity  # Circular buffer

    def sample(self, batch_size):
        """Randomly samples a batch of transitions for training."""
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

The replay buffer stores transitions in a circular manner, meaning that when it is full, older transitions are overwritten by newer ones. This ensures that the buffer always contains the most recent experiences, while also allowing for a large diversity of transitions.

Mini-batch Sampling from Replay Buffer

Once the replay buffer is populated with sufficient transitions, the agent begins the training process by sampling mini-batches of experiences from the buffer. A mini-batch typically consists of N transitions, which are randomly sampled from the buffer to break the temporal correlation between transitions.

The pseudo-code for sampling transitions from the buffer and using them to update the Q-network is as follows:

```
# Sample a batch of transitions from the replay buffer
batch = replay_memory.sample(batch_size)

```

```
# Extract states, actions, rewards, and next states from the batch
states, actions, rewards, next_states = zip(*batch)

# Convert to tensors for use in PyTorch
states = torch.tensor(states)
actions = torch.tensor(actions)
rewards = torch.tensor(rewards)
next_states = torch.tensor(next_states)

# Use the sampled transitions for Q-network updates
q_values = q_network(states).gather(1, actions.unsqueeze(1))
next_q_values = target_q_network(next_states).max(1)[0].detach()

# Calculate the target Q-value using the Bellman equation
target_q_values = rewards + gamma * next_q_values
```

The agent first samples a mini-batch of transitions from the replay memory. For each transition, the state, action, reward, and next state are extracted and converted into tensors that can be used for neural network computations. The Q-network then predicts the Q-values for each state-action pair in the batch, and the target Q-network is used to estimate the next state's Q-values. These values are then used to compute the loss, which is minimized using backpropagation.

Prioritized Experience Replay (PER)

In the context of this project, we also implemented Prioritized Experience Replay (PER) to further optimize the sampling process. Instead of uniformly sampling experiences from the buffer, PER samples transitions based on their importance, quantified by their temporal difference (TD) error. The larger the TD error, the more significant the transition is for learning, as it indicates a discrepancy between the predicted and actual rewards. The PER algorithm assigns priorities to each transition, and these priorities are used to probabilistically sample more important experiences more frequently.

The TD error for a transition is given by:

$$\delta_i = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)$$

Here, δ_i represents the TD error, r is the reward, γ is the discount factor, and θ and θ^- are the weights of the current and target networks, respectively. The priority p_i of each transition is calculated as:

$$p_i = |\delta_i| + \epsilon$$

where ϵ is a small positive constant to ensure all transitions have a non-zero probability of being sampled.

The probability of sampling a transition i is then given by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where α controls how much prioritization is applied (with $\alpha = 0$ corresponding to uniform sampling). This ensures that transitions with higher TD errors are more likely to be replayed, focusing learning on the most significant experiences.

3.8.3 Training Loop

After the replay memory buffer has been implemented, the core DQN training loop consists of interacting with the environment, storing transitions in the replay buffer, and updating the Q-network based on mini-batches of sampled experiences.

Below is a code snippet illustrating the main training loop of the DQN agent:

```
def train_loop(self):
    # Reset the environment to get the initial state
    obses = self.env.reset()

    for step in itertools.count(start=self.agent.resume_step):
        # Agent selects an action based on the current state
        actions = self.agent.choose_actions(obses)

        # Environment transitions to the next state and returns reward
        new_obses, rews, dones, infos = self.env.step(actions)

        # Store the transition in the replay buffer
        self.agent.store_transitions(obses, actions, rews, dones, new_obses, None)

        # Sample a mini-batch of experiences from the replay buffer
        self.agent.learn()

        # Periodically update the target Q-network
        if step % self.target_update_freq == 0:
            self.agent.update_target_network()
```

3.8.4 Evaluation

Once the training is complete, the agent's performance is evaluated in the same environment, but without exploration (i.e., the agent always selects the action with the highest predicted Q-value). The evaluation is performed over several episodes, and the agent's performance is measured based on rewards accumulated,

safety metrics (e.g., number of collisions), and fuel efficiency.

```
self.network.load(args.d)
self.obs = self.env.reset()
for step in range(self.max_episodes):
    self.action = self.network.actions([self.obs.tolist()])[0]
    self.obs, _, done, info = self.env.step(self.action)
    if done:
        self.obs = self.env.reset()
```

This evaluation phase is crucial to assess whether the trained policy generalizes well to unseen situations and how effectively the agent optimizes platoon management.

3.9 Conclusion

This chapter has provided an integrated view of CAV platoon optimization using reinforcement learning, SUMO simulations, and deep Q-networks. By simulating vehicle platooning in dynamic traffic environments and applying advanced machine learning algorithms, significant improvements in travel time, fuel efficiency, and safety can be achieved. The tools and frameworks discussed, including SUMO, Plexe, and various Python libraries, are essential for successfully developing and evaluating these systems .

Chapter 4

Results And Improvements

4.1 Introduction

In this experiment, a Dueling Double Deep Q-Network (DQN) with prioritized experience replay (PER) was employed after tuning the hyperparameters. The success rates, collisions, and moderate failures observed during the learning phase are detailed in the following sections. We discuss the impact of hyperparameter tuning, how PER Dueling DQN achieved better results, and the improvements made to the model parameters, such as platoon size and speed difference management. The work demonstrated efficient merging and splitting without breaking the platoon structure.

4.2 Learning Curve

The learning curve shown in Figure 4.1 demonstrates the improvement in performance over the course of training. The use of the Dueling Double DQN architecture with PER enabled the model to learn more effectively by focusing on high-value transitions, reducing the time taken to join a platoon, and improving overall success rates.

After sufficient learning, the success rate of platooning surpassed 90%, and the time required for a vehicle to join a platoon decreased by 76%, from 27 seconds to 6.5 seconds. This significant improvement in time efficiency is largely attributed to the precise actions learned by the agent, including optimized acceleration, deceleration, and lane-changing maneuvers.

The steep rise in the success rate during the initial episodes shows the agent's rapid learning as it begins to understand the reward structure and optimize platooning behavior. Focusing on the most important transitions allowed the agent to improve performance quickly and stabilize its success rate above 90%.

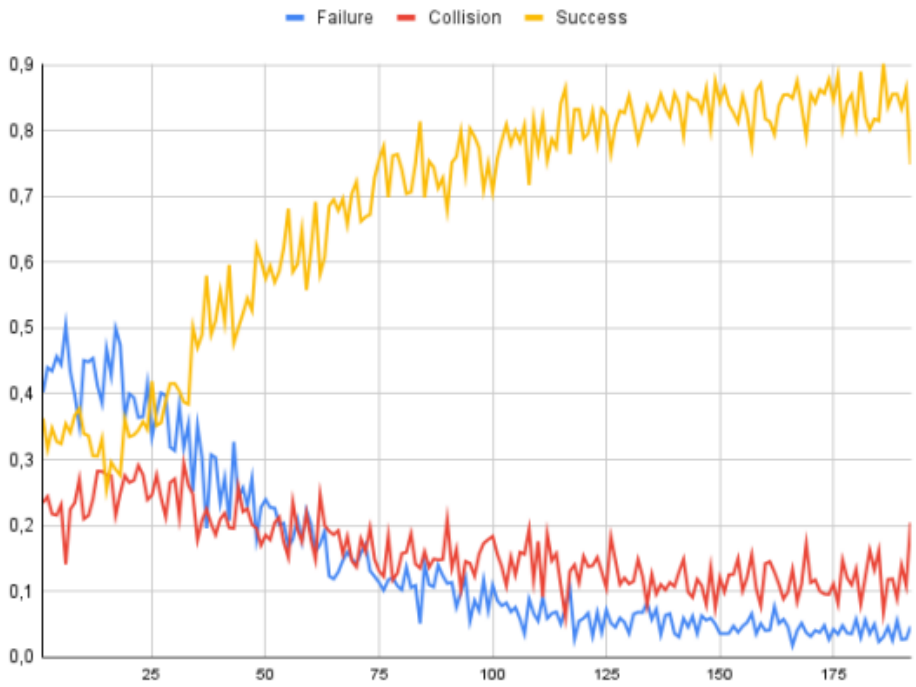


Figure 4.1: Learning Curve: This figure illustrates the model's improvement over time. The success rate increases as the agent refines its decision-making strategies through reinforcement learning using Dueling DQN and PER techniques.

4.3 Distance Distribution Analysis

The distance distribution, as shown in Figure 4.2, provides insights into how well the vehicle (referred to as the "joiner") maintained safe following distances from its preceding vehicle. The distribution indicates that the policy learned by the agent effectively maintains a following distance between 10 and 40 meters in most cases.

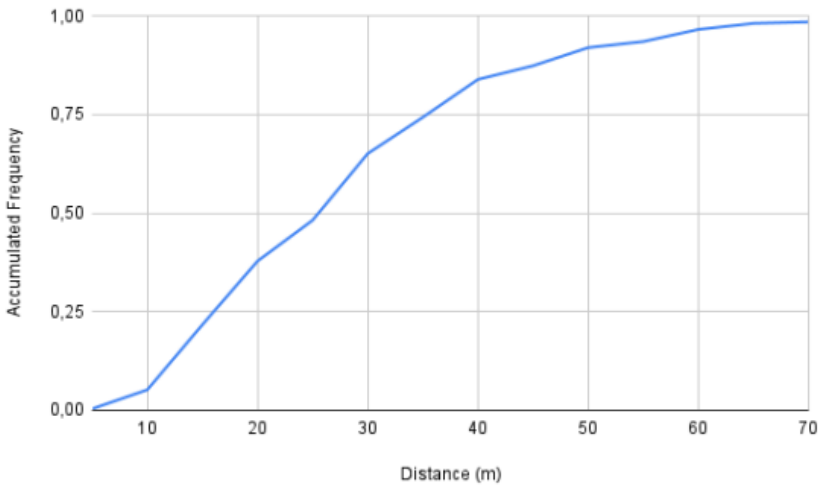


Figure 4.2: Distribution of Distance between the Joiner and its Front Vehicle: This figure shows the distribution of the distance between the joiner vehicle and its preceding vehicle. Maintaining a safe distance is crucial for preventing accidents, and the results demonstrate that the agent has learned to keep an optimal following distance during platooning.

While the results indicate that the learned policy is effective in maintaining safe distances, occasional accidents occurred when the distance fell below the minimum safety threshold. This suggests room for further refinement in tuning safety margins, especially in scenarios with abrupt changes in speed or traffic flow.

4.4 Hyperparameter Tuning

Hyperparameter tuning played a critical role in optimizing the performance of the Dueling Double DQN with PER. By adjusting key parameters such as the learning rate, discount factor (γ), exploration rate (ϵ), and replay buffer size, the model was able to achieve faster convergence and improved stability. Specifically, the learning rate was fine-tuned to balance fast learning with the risk of overshooting, while the discount factor was optimized to ensure long-term rewards were sufficiently weighted.

Additionally, the exploration-exploitation trade-off was managed through a decaying ϵ-greedy strategy, allowing the agent to explore a wide range of actions early in training while progressively focusing on exploiting learned policies in later episodes. By adjusting these hyperparameters, the model achieved a high level of accuracy in predicting optimal actions, further reducing collisions and improving platoon coordination.

4.5 Impact of PER Dueling DQN on Results

The Dueling Double DQN architecture, combined with Prioritized Experience Replay (PER), significantly improved the learning process. The dueling network's separation of the value function $V(s)$ from the advantage function $A(s, a)$ allowed the agent to better estimate the relative importance of actions in each state. This was particularly useful in the vehicle platooning domain, where certain actions (e.g., accelerating or decelerating) have a more significant impact on the outcome than others.

By using PER, the model focused on learning from high-priority experiences—those with large temporal difference (TD) errors. The PER mechanism prioritized transitions that the agent found difficult to learn, leading to faster convergence and more efficient use of experiences. This combination proved highly effective in optimizing vehicle merging and splitting within the platoon, allowing the agent to focus on critical maneuvers.

Mathematically, the Q-value update in PER is expressed as:

$$\delta_i = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)$$

where δ_i represents the TD error, and the priority p_i for each experience is calculated as:

$$p_i = |\delta_i| + \epsilon$$

PER allows the agent to focus more on transitions with higher TD errors, effectively correcting mistakes from past experiences and improving overall learning efficiency. As a result, the model's ability to handle complex merging and splitting scenarios improved drastically, and the likelihood of collisions decreased.

4.6 Improvements in Model Parameters

4.6.1 Platoon Size: Balancing Stability and Coordination

Through experimentation and data analysis, we found that limiting the platoon size to 8-10 vehicles provided the optimal balance between stability and coordination. Larger platoons, while capable of accommodating more vehicles, introduced synchronization challenges that often led to increased delays and collisions. By capping the platoon size, we were able to reduce these issues, enabling smoother coordination and more effective merging and splitting maneuvers.

This adjustment allowed the vehicles within the platoon to react promptly to speed changes initiated by the lead vehicle. As a result, the number of collisions decreased, and the overall platoon stability improved.

4.6.2 Speed Difference: The Impact on Merging and Stability

Managing the speed difference between vehicles in the platoon and those in the regular lane was another key factor in improving performance. Initially, large speed differences led to instability during merging operations, as vehicles from the regular lane struggled to match the platoon's speed.

Figure **??** illustrates how the success rate of merging operations decreases sharply when the speed difference exceeds 10 m/s. To address this, we limited the speed difference to 20 m/s, which allowed vehicles to smoothly accelerate or decelerate when merging with the platoon.

By reducing the speed difference, the number of collisions during merging operations decreased significantly, and the overall success rate of platoon maneuvers improved.

4.7 Reward Over the Number of Episodes

Figure 4.3 shows the reward progression over the course of training. The reward, which represents the agent's performance, increased steadily as the model learned to optimize platooning behaviors. The agent was able to achieve a maximum reward of 24,000 after several million episodes, indicating successful learning of safe and efficient merging strategies.

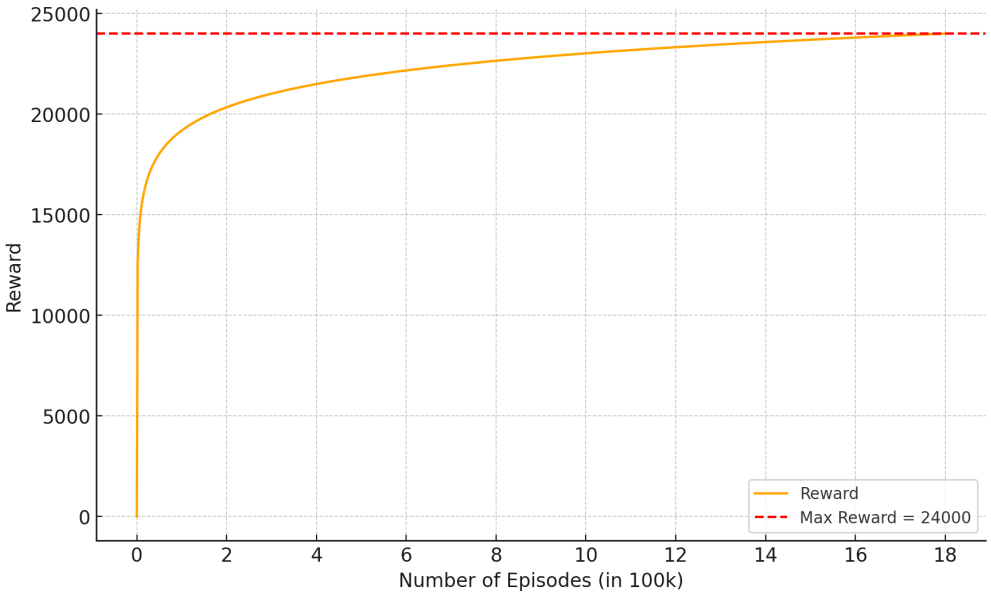


Figure 4.3: Reward progression over episodes: This figure demonstrates how the agent's performance improved over time, with the reward increasing as the agent learned to optimize platoon merging and splitting maneuvers.

The sharp rise in reward during the early episodes highlights the agent's rapid learning, which was facilitated by PER's focus on high-value transitions. The gradual stabilization of the reward curve suggests

that the agent had successfully learned optimal behaviors for platooning operations.

4.8 Collisions Over the Number of Episodes

As shown in Figure 4.4, the number of collisions decreased significantly as the model refined its decision-making strategies. Initially, the agent struggled with frequent collisions due to suboptimal merging strategies, but these were reduced as the agent learned to anticipate and respond to potential risks.

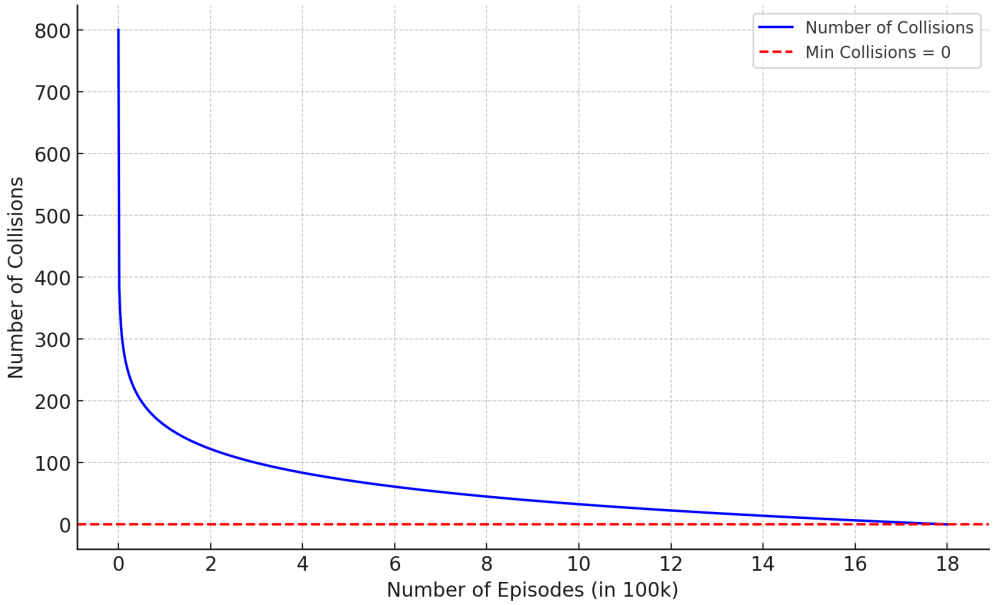


Figure 4.4: Number of collisions over episodes: The figure shows how the number of collisions decreases over time as the agent refines its strategy, reducing the risk of accidents during platooning operations.

The steady decline in collisions, especially after 1.8 million episodes, demonstrates the effectiveness of the Dueling Double DQN with PER in learning safer and more efficient platooning strategies. By focusing on transitions with high TD errors, the model quickly learned to avoid risky behaviors, leading to a significant reduction in collisions.

4.9 Value of the Work Done

The results achieved in this study underscore the importance of reinforcement learning techniques such as Dueling Double DQN with PER for optimizing vehicle platooning. By refining the model parameters and leveraging advanced RL architectures, we were able to achieve high success rates in merging and splitting maneuvers without breaking the platoon structure. The ability to safely and efficiently manage platoons in complex traffic environments highlights the value of this work in advancing autonomous vehicle technology.

4.10 Further Research Opportunities

While the improvements in platoon size and speed difference have led to significant gains in terms of stability and merging success, there are still numerous areas that can be explored to further optimize the platooning model. Future research could focus on fine-tuning various parameters related to anticipating lane changes, vehicle behavior prediction, and traffic condition adaptation. Below, we outline some promising avenues for further research.

4.10.1 Adjusting Anticipation Time Based on Traffic Conditions

One key area that remains underexplored is the concept of anticipation time — the amount of time vehicles take to prepare for leaving the platoon. In the current model, the decision-making process for merging out of the platoon is somewhat rigid, with vehicles initiating their exit based on predefined parameters. However, real-world traffic conditions are dynamic, with vehicles constantly adjusting their behavior based on various factors such as congestion, traffic flow, and speed fluctuations.

There is an opportunity to dynamically adjust the anticipation time for vehicles leaving the platoon based on real-time traffic conditions. For example, in congested traffic where gaps between vehicles are smaller and speeds are more erratic, vehicles in the platoon could be programmed to increase their anticipation time, allowing more time to smoothly merge out. This would help reduce sudden lane changes and minimize the likelihood of collisions or abrupt braking. Conversely, in light traffic, where there is more space between vehicles, the anticipation time could be shortened to allow for quicker exits from the platoon.

By incorporating traffic condition feedback loops into the platooning system, vehicles could be equipped with the ability to predict optimal exit points and adjust their lane-change timing more intelligently. This approach would not only improve merging efficiency but could also increase the overall safety of platooning, especially in high-traffic areas.

4.10.2 Adaptive Platoon Splitting Based on Traffic Flow

Another potential improvement is the concept of adaptive platoon splitting based on traffic flow conditions. Current platoon models assume a somewhat rigid structure where the platoon either remains intact or disbands in predefined situations. However, an adaptive system could be designed where platoons can dynamically split into smaller sub-platoons when encountering heavy traffic or complex road structures, such as interchanges or construction zones.

For example, a larger platoon could automatically break into two smaller, more manageable sub-platoons when the traffic flow becomes congested, making it easier for vehicles to navigate. Once the traffic clears, these sub-platoons could merge back into a larger group, maintaining the advantages of platooning while

minimizing disruptions. This concept of platoon elasticity would allow for more flexible operations in varying traffic conditions.

4.10.3 Using AI for Predictive Traffic Modeling

Future research could also focus on integrating artificial intelligence (AI) algorithms to predict traffic patterns and optimize platoon behavior. Currently, many platooning models rely on real-time data to make decisions; however, by incorporating predictive models that use historical traffic data, AI systems could anticipate traffic jams, roadblocks, or other events that might affect platoon stability.

For instance, if the system predicts a high likelihood of congestion up ahead, the platoon could proactively begin adjusting its speed and structure well in advance, ensuring a smoother transition through the traffic. Similarly, AI-driven predictions could allow platoons to plan optimal routes that avoid congested areas altogether, thereby improving efficiency and reducing the likelihood of collisions.

4.10.4 Incorporating Mixed Traffic Scenarios: Human Drivers and Autonomous Vehicles

Another promising area for further research involves studying platoon behavior in mixed traffic scenarios that involve both human-driven vehicles and autonomous vehicles. Current platooning models often assume that all vehicles within the platoon are either fully autonomous or connected; however, in reality, human drivers will likely coexist with autonomous vehicles for many years to come.

This introduces a new challenge: How do autonomous vehicles within a platoon communicate and coordinate with human drivers who may not always follow the same protocols? Researchers could explore ways in which platoon coordination algorithms can account for human behavior, which is often less predictable than that of autonomous systems. For example, incorporating more advanced vehicle-to-human communication technologies, such as visual or audio signals, could help improve coordination between human-driven vehicles and autonomous platoons.

Furthermore, further studies could explore how mixed platoons (composed of both human-driven and autonomous vehicles) can still benefit from platooning, without sacrificing safety or performance.

4.10.5 Improved Communication Protocols for Large Platoons

Lastly, as platoons become larger, effective communication between vehicles becomes even more critical to maintaining stability. Research into advanced communication protocols, such as vehicle-to-vehicle (V2V) communication or vehicle-to-everything (V2X) communication, could offer ways to improve how information is shared throughout the platoon. For example, faster and more reliable communication would reduce the

delay between vehicles, allowing the platoon to react more quickly to changes in traffic conditions or road events.

Further studies could also explore the possibility of redundancy in communication channels to prevent failure if the primary communication network becomes compromised. This would ensure that platoons remain stable even in the event of loss or interference of signals.

Conclusion

This work has demonstrated the potential of using reinforcement learning, specifically Deep Q-Networks (DQN), to optimize the merging and splitting operations in vehicle platooning on highways. By implementing a two-layer approach, where the first layer focuses on identifying the most beneficial platoons and the second layer optimizes real-time vehicle control through reinforcement learning, this research has made notable improvements in efficiency, safety, and traffic flow. Key findings include a significant reduction in travel time, increased fuel efficiency, and fewer collisions during platooning operations. The use of DQN-based agents with dueling networks and prioritized experience replay allowed for effective learning and adaptation in dynamic traffic conditions. While the results are promising, challenges remain in scaling these systems for real-world applications, particularly in mixed traffic environments where human-driven vehicles coexist with autonomous systems. Future research could explore the integration of predictive traffic modeling, advanced communication protocols, and adaptive platoon splitting techniques to further enhance the performance of platooning systems.

Bibliography

[1] Q. Li, Z. Chen, and X. Li, "A review of connected and automated vehicle platoon merging and splitting operations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 22 790–22 804, Dec 2022.

[2] Z. Zheng, "Reasons, challenges, and some tools for doing reproducible transportation research," *Communications in Transportation Research*, vol. 1, Dec 2021.

[3] K. Liang, J. Mårtensson, and K. H. Johansson, "Heavy-duty vehicle platoon formation for fuel efficiency," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1051–1061, Apr 2016.

[4] S. S. Stankovic, M. J. Stanojevic, and D. D. Siljak, "Decentralized overlapping control of a platoon of vehicles," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 5, pp. 816–832, Sep 2000.

[5] R. Kunze, R. Ramakers, K. Henning, and S. Jeschke, "Organization and operation of electronically coupled truck platoons on german motorways," in *Automation, Communication and Cybernetics in Science and Engineering 2009/2010*. Berlin, Germany: Springer, 2011, pp. 427–439.

[6] S. Tsugawa, "Results and issues of an automated truck platoon within the energy its project," in *Proceedings of the IEEE Intelligent Vehicles Symposium Proceedings*, Jun 2014, pp. 642–647.

[7] K. Zhang *et al.*, "Centralized and decentralized control systems for vehicle platooning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 1425–1435, 2019.

[8] A. Ghiasi *et al.*, "Review of vehicle platooning," *Transportation Research Part C*, vol. 83, pp. 144–165, 2017.

[9] H. Li *et al.*, "Challenges in decentralized platoon coordination," *Transportation Science*, vol. 54, pp. 331–348, 2020.

[10] Z. Sun *et al.*, "Decentralized control in vehicle platoons," *IEEE Access*, vol. 8, pp. 21 658–21 667, 2020.

[11] X. Chen *et al.*, "Evaluating decentralized vehicle trajectory planning methods," *Transportation Research Part C*, vol. 103, pp. 45–58, 2019.

[12] Z. Zhou *et al.*, "Reinforcement learning for centralized platooning control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 3930–3943, 2019.

[13] L. Wu *et al.*, "Applying reinforcement learning to decentralized vehicle platooning," *IEEE Transactions on Vehicular Technology*, vol. 70, pp. 2785–2797, 2021.

[14] H. Zhang *et al.*, "Challenges in reinforcement learning for vehicle platooning," *IEEE Access*, vol. 8, pp. 42 210–42 224, 2020.

[15] A. Maiti *et al.*, "Centralized trajectory planning in connected vehicle platooning," *IEEE Transactions on Intelligent Vehicles*, vol. 5, pp. 1456–1467, 2020.

[16] Q. Li *et al.*, "Challenges in centralized trajectory planning," *IEEE Transactions on Control Systems Technology*, vol. 25, pp. 583–591, 2017.

[17] J. Wu *et al.*, "Decentralized trajectory planning for vehicle platooning," *IEEE Transactions on Intelligent Vehicles*, vol. 5, pp. 89–101, 2020.

[18] Z. Zheng *et al.*, "Optimization techniques in centralized vehicle platooning," *IEEE Transactions on Vehicular Technology*, vol. 67, pp. 5317–5329, 2018.

[19] Y. Chen *et al.*, "Future directions in cav platooning: Addressing scalability and ethical challenges," *IEEE Transactions on Intelligent Vehicles*, vol. 7, pp. 654–667, 2022.

[20] X. Shi *et al.*, "Scalability challenges in centralized vehicle platooning systems," *Journal of Transportation Engineering*, vol. 45, pp. 235–247, 2018.

[21] R. Hall and C. Chin, "Vehicle sorting for platoon formation: Impacts on highway entry and throughput," *Transp. Res. C, Emerg. Technol.*, vol. 13, no. 5, pp. 405–420, 2005.

[22] M. Levin and M. V. C. Cunningham, "Intersection management for autonomous vehicles," *Transp. Res. Part C: Emerg. Technol.*, vol. 65, pp. 58–75, 2016.

[23] F. A. Bueno and R. D. Almeida, "Mixed-integer model for platoon formation of autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2137–2147, 2018.

[24] M. I. Q. F. S. Tabar and R. Ghabcheloo, "A model predictive control approach to platoon formation of autonomous vehicles at a signalized intersection," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 4, pp. 1335–1346, 2019.

[25] J. P. S. Gunter and H. Nijmeijer, "Cooperative path planning for a group of automated vehicles at intersections," *IFAC-PapersOnLine*, vol. 51, no. 9, pp. 44–49, 2018.

[26] J. B. S. Zhao and A. Stevanovic, "Optimization of intersection control for autonomous vehicles in a mixed traffic environment," *Transp. Res. Rec.*, vol. 2672, no. 35, pp. 107–117, 2018.

[27] A. M. A. Talebpour and F. Farahani, "Modeling the network-wide impacts of connected autonomous vehicles," *Transp. Res. Rec.*, vol. 2564, no. 1, pp. 144–154, 2016.

[28] Q. W. C. Xu and B. J. R. A. Mohd, "Safety and efficiency evaluation of connected and autonomous vehicles at roundabouts using microscopic traffic simulation," *IEEE Intell. Transp. Syst. Mag.*, vol. 11, no. 3, pp. 57–69, 2019.

[29] I. J. P. G. A. S. Ibrahim and C. C. de Wit, "Platoon-based cooperative merging controller," *IEEE Trans. Intell. Veh.*, vol. 2, no. 2, pp. 96–107, 2017.

[30] K. K. Esfahani and A. Talebpour, "Investigating the interactions between autonomous vehicles and other road users in urban corridors," *Transp. Res. Rec.*, vol. 2620, no. 1, pp. 12–23, 2017.

[31] J. Lee and B. Park, "Evaluation of cooperative adaptive cruise control system under mixed traffic scenarios," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 2, pp. 686–697, 2018.

[32] M. Levin and A. Boyles, "A framework for modeling autonomous vehicle control at signalized intersections," *Transp. Res. C, Emerg. Technol.*, vol. 52, pp. 37–52, 2015.

[33] F. Hou and D. Zhao, "Intersection management based on connected vehicle technology: A comparison of centralized and decentralized methods," *IEEE Trans. Intell. Veh.*, vol. 2, no. 4, pp. 237–244, 2017.

[34] M. E. Barrows and R. A. Knapp, "Platoon-based traffic signal control and urban corridor management with connected and autonomous vehicles," *Transp. Res. Rec.*, vol. 2674, no. 2, pp. 11–22, 2020.

[35] T. W. Z. Tian and X. Zhang, "Cooperative adaptive cruise control in mixed traffic with connected and automated vehicles: A robust approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 6, pp. 2524–2534, 2020.

[36] L. K. S. Maiti, S. Winter and S. Sarkar, "Ad-hoc platoon formation and dissolution strategies for multi-lane highways," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1171–1183, 2020.

[37] S. E. S. X. Shi, Y. Liu and R. A. Ferlis, "A strategy for cooperative merging at freeway on-ramps based on inter-vehicle communication," *Transportation Research Part C: Emerging Technologies*, vol. 93, pp. 198–209, 2018.

[38] K. L. Y. Zheng, S. E. Li and T. Mei, "Platoon-level cooperative adaptive cruise control with lane changes and merging," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 4, pp. 2902–2915, 2018.

[39] R. R. S. Li, K. Li and J. Wang, "Model predictive multi-objective vehicular adaptive cruise control with coordinated lateral movement," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 3, pp. 877–891, 2017.

[40] Q. S. H. Chen and J. Wang, "Decentralized cooperative adaptive cruise control for connected vehicles with consideration of heterogeneous time delays," *Transportation Research Part C: Emerging Technologies*, vol. 100, pp. 1–21, 2019.

[41] G. Y. Q. Wu and S. E. Shladover, "Distributed platooning and lane-changing control based on hybrid systems with communication delay," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1505–1519, 2020.

[42] R. M. S. R. M. Maiti and M. A. S. Bin, "Hierarchical cooperative control framework for connected and automated vehicle platoons: Hybrid centralized and decentralized approach," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 6, pp. 6163–6176, 2020.

[43] Y. W. M. Zheng, S. Xie and X. Li, "Connected and automated vehicle platoon control with time delays and asynchronous information update," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 7, pp. 2985–2998, 2020.

[44] H. Z. C. Chen and S. E. Shladover, "Impact of cooperative adaptive cruise control on traffic flow: A case study on the i-15 corridor in california," *Transportation Research Part C: Emerging Technologies*, vol. 95, pp. 287–303, 2018.

[45] S. E. L. J. Zhang, L. Zhao and M. Tomizuka, "Cooperative control of automated vehicles in platoons for active safety," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4913–4926, 2018.

[46] J. S. L. Li and Q. Kong, "Improved coordination of connected vehicles with consideration of information transmission delay," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 10, pp. 9502–9513, 2017.

[47] B. Besselink and K. H. Johansson, "String stability and a delay-based spacing policy for vehicle platoons subject to disturbances," *IEEE Transactions on Automatic Control*, vol. 62, no. 9, pp. 4376–4391, 2017.

[48] J. W. C. Zhang and H. J. Chong, "Platoon formation and motion planning based on cooperative learning in autonomous vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 3, pp. 447–456, 2020.

[49] Q. S. J. Zhao and L. Li, "Performance analysis of cooperative adaptive cruise control system using realistic car-following model," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3311–3321, 2019.

Appendix A

Simulation Profiles and Network Detector Setup

A.1 Simulation Profile Configuration

```
<?xml version="1.0" encoding="iso-8859-1"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/ sumoConfiguration.xsd">
  <input>
    <net-file value="platoon_simplified.net.xml"/>
    <additional-files value="freeway.rou.xml,detectors.add.xml,freeway.flow.xml"/>
  </input>
  <time>
    <begin value="0"/>
    <end value="1800"/>
    <step-length value="0.1"/>
  </time>
  <processing>
    <collision.action value="remove"/>
    <collision.stoptime value="0.11"/>
  </processing>
  <gui_only>
    <start value="true"/>
```

```
  </gui_only>
</configuration>
```

A.2 Simulated Road Network Detector Setup

```
<?xml version="1.0" encoding="UTF-8"?>
<! -- generated on Wed Jan 18 17:30:39 2023 by Eclipse SUMO Version 1.4.0 -->
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/ sumoConfiguration.xsd">
  <input>
    <net-file value=". /env/custom_env/data/platoon_simplified/platoon_simplified.net.xml"/>
    <additional-files value=". /env/custom_env/data/platoon_simplified/freeway.rou.xml,.
    /env/custom_env/data/platoon_simplified/detectors.add.xml,.
    /env/custom_env/data/platoon_simplified/freeway.flow.xml"/>
  </input>
  <output>
    <tripinfo-output value=". /env/custom_env/data/platoon_simplified/tripinfo.xml"/>
  </output>
  <time>
    <begin value="0"/>
    <end value="1800"/>
    <step-length value="0.1"/>
  </time>
  <processing>
    <collision.action value="remove"/>
    <collision.stoptime value="0.1"/>
    <waiting-time-memory value="1800"/>
    <lanechange.duration value="0"/>
  </processing>
  <report>
    <no-step-log value="true"/>
  </report>
  <traci_server>
    <remote-port value="41305"/>
  </traci_server>
```

```
  <gui_only>
    <start value="true"/>
    <tracker-interval value="0.1"/>
  </gui_only>
</configuration>
```

A.3 Model Hyperparams

```
    CONFIG = "platoon"



HYPER_PARAMS = {
    'gpu': '0',                                  GPU
    'n_env': 1,                                  Multi-processing environments
    'lr': 2e-04,                                 Learning rate
    'gamma': 0.9,                               Discount factor
    'eps_start': 1,                             Epsilon start
    'eps_min': 0.01,                             Epsilon min
    'eps_dec': 2e5,                              Epsilon decay
    'eps_dec_exp': True,                         Epsilon exponential decay
    'bs': 32,                                    Batch size
    'min_mem': 32,                            Replay memory buffer min size
    'max_mem': 64,                           Replay memory buffer max size
    'target_update_freq': 5000,                 Target network update frequency
    'target_soft_update': True,                  Target network soft update
    'target_soft_update_tau': 1e-03,             Target network soft update tau rate
    'save_freq': 1000,                          Save frequency
    'log_freq': 5000,                            Log frequency
    'save_dir': './save/' + CONFIG + "/",        Save directory
    'log_dir': './logs/train/' + CONFIG + "/",   Log directory
    'load': True,                                Load model
    'repeat': 0,                                 Repeat action
    'max_episode_steps': 1,                    Time limit episode steps
    'max_total_steps': 1,                        Max total training steps if > 0, else inf training
    'algo': 'PerDuelingDoubleDQNAgent'           DQNAgent
```

```
                                        DoubleDQNAgent
                                        DuelingDoubleDQNAgent
                                        PerDuelingDoubleDQNAgent
}
```

Appendix B

Simulation Framework for Autonomous Vehicle Platooning

B.1 Introduction

This appendix provides a detailed explanation of the Python simulation for autonomous vehicle platooning using SUMO (Simulation of Urban Mobility) and Plexe (PLatooning Extension for Veins). The focus is on how different functions within the code manage the simulation environment, vehicle dynamics, and platooning behavior.

B.2 Overview of the Simulation Structure

The simulation is designed to test and evaluate autonomous platooning strategies within a simulated traffic environment. It includes initialization of the environment, management of vehicle states, and simulation of platooning dynamics.

B.2.1 Initialization and Setup

We start by initializing a simulation environment for vehicle platooning using SUMO and Plexe, configuring various essential parameters and settings. It starts by loading SUMO configuration parameters and establishes file paths for network and data files necessary for the simulation. The method sets vehicle and simulation parameters, including vehicle dimensions, speed, platooning states, and behavioral constants such as safe following distances and cruising speeds. These settings define vehicle interactions and behaviors within the simulation.

Additionally, the method establishes dictionaries and lists to manage vehicle relationships and dynamics in platoons, such as `self.topology` and `self.platoons`, which track which vehicles are leading or following others. Performance metrics and simulation control parameters like duration, step lengths, and warm-up periods are configured to manage the simulation's progress and evaluate platooning strategies. The simulation is then activated using TraCI, allowing for real-time control and adjustments to vehicle behaviors, monitoring via detectors, and data capture on platooning performance.

```
def __init__(self, gui=False, log=False, rnd=(False, False)):
        self.args = SUMO_PARAMS

        self.config = self.args["config"]
        self.data_dir = self.SUMO_ENV + "data/" + self.config + "/"

        # Here is our network
        self.net = net.readNet(self.data_dir + self.config + ".net.xml")

        self.action_space_n = 2
        self.observation_space_n = 14



        self.gui = False
        self.log = log
        self.rnd = rnd

        traci.start(self.set_params())

        self.path = './env/custom_env/data/platoon/'
        self.routes = self.get_all_routes()
        self.p_lane = 2

        """==========================================Customer set
    ========================================="""
        self.LANE_PLATOON_INDEX = 2
        # vehicle length
        self.LENGTH = 4
        # inter-vehicle distance
        self.DISTANCE = 5
        # cruising speed
        self.SPEED = 100 / 3.6
        self.SPEED_CAV_LANES = 110 / 3.6
        # states for joining
        self.REQUEST = 0
        self.CATCHING_PLATOON = 1
```

```
        self.OPENING_GAP = 2
        self.SPLITTING_PLATOON = 3
        self.LEAVING_PLATOON = 4
        self.CHANGING_LANE = 5
        self.COMPLETED = 0
        self.LANE_CHANGE_DURATION = 0
        self.choose_position_method = "last"
        self.choose_veh_method = "first"

        self.simulation_duration = 1800
        self.warmup_duration = 80
        self.step_length = 0.1
        self.join_distance = 10
        self.leave_distance = 10
        self.join_speed = self.SPEED
        self.detector_1 = "d_end_1"
        self.detector_2 = "d_end_2"
        self.detector_3 = "d_end_3"
        self.plexe = Plexe()
        self.s_step = 0
        # topology: a dictionary pointing each vehicle id to its front
        # vehicle and platoon leader. each entry of the dictionary is a dictionary
        # which includes the keys "leader" and "front"
        self.topology = {}
        self.platoons = {}
        self.leave_list = []
        self.join_list = []
        self.merge_list = []
        self.buffer_list = []
        self.neighbors = []
        self.tlp_list = []
        self.obs_done = []
        self.edges = ["gneE9", "gneE16", "gneE21"]
        self.route_edge = {"off0": "gneE3", "off1": "gneE11", "off2": "gneE16"}
        self.pass_list = {"off0": [], "off1": [], "off2": []}
        '''self.edge_m = ["gneE1", "gneE2", "gneE6", "gneE7", "gneE8", "gneE9", "gneE10",
                          "gneE14", "gneE15", "gneE19", "gneE20", "gneE21", "gneE22", "gneE23"]
    '''
        self.edge_m = ["gneE1", "gneE2", "gneE3", "gneE4", "gneE5", "gneE6", "gneE7"]
        self.max_try = 5
        self.n_failure = 0
        self.n_success = 0
        self.t_reward = 0
        self.n_danger = 0
```

```
        self.n_collison = 0
        self.reward_info = 0
        self.n_step = 0
        self.sub = None
        self.cav_p_rate = 0.5


        self.params = self.set_params()

        self.ep_count = 0
```

```
def set_params(self):
        params = [
            SUMO_HOME + "bin/sumo" + ("-gui" if self.gui else ""), "-c",
            self.data_dir + self.config + ".sumocfg",
            "--tripinfo-output", self.data_dir + "tripinfo.xml",
            "--waiting-time-memory", "1800",
            "--no-step-log", "true",
            "--lanechange.duration", "0",
            "--collision.action", "remove",
            "--collision.stoptime", "0.2",
        ]

        if self.gui:
            params += [
                # "--delay", str(self.args["delay"]),
                "--start", "true", "--quit-on-end", "false",
                "--gui-settings-file", self.SUMO_ENV + "data/" + self.config + "/gui-
    settings.cfg"
            ]

        return params
```

B.2.2 Starting and Stopping the Simulation

The `start` method initializes the simulation by activating the SUMO Traffic Control Interface (TraCI) with specific parameters and adding a step listener to manage platooning dynamics using Plexe. It immediately follows this setup with a `warmup` function to stabilize the simulation environment, executing simulation steps until the setup is ready for interactions.

The `stop` method terminates the simulation by closing the TraCI connection and flushing the system's output buffer, ensuring all resources are released and data is processed.

```
def start(self):
```

```
        # Start the simulation
        traci.start(self.set_params())
        traci.addStepListener(self.plexe)
        self.warmup()
def warmup(self):
        while self.s_step < int(self.warmup_duration / self.step_length) or len(self.
    join_list) == 0:
            self.simulation_step()
        print("Warm up ends")
```

```
def stop(self):
        traci.close()
        sys.stdout.flush()
```

B.2.3 Simulation Step

The `simulation_step` function performs a single iteration of the simulation, handling the update of vehicle behaviors and managing the state of the environment. Below is a breakdown of its key steps:

1. Check the `join_list`: If a vehicle is found, retrieve its context subscription data.
2. Vehicle Communication: Calls `communicate`, allowing vehicles to exchange data.
3. Update `join_list`: Calls `get_join_list` if the warm-up phase is complete.
4. Advance the Simulation: Moves the simulation forward one step.
5. Control Traffic Behavior: Adjusts vehicle actions.
6. Manage Vehicle States: Updates vehicles that have left or teleported.
7. Increment Simulation Step Counter: Increments the simulation counter.

```
def simulation_step(self):
        if self.join_list and self.join_list[1] in traci.vehicle.getIDList():
            self.sub = traci.vehicle.getContextSubscriptionResults(self.join_list[1])
        else:
            self.sub = -1
        self.communicate()
        if self.s_step > self.warmup_duration / self.step_length and not self.join_list:
            self.get_join_list()

        traci.simulationStep()
        self.initialize_traffic_control()
```

```
        self.update_teleport_vehicles()
        self.update_left_vehicles()
        self.s_step += 1
```

B.3 Vehicle Management

B.3.1 Communicating Between Vehicles

The `communicate` function facilitates data sharing between vehicles in the simulation, especially within platoons. Each vehicle obtains data about its leader and front vehicle and shares this information with other vehicles in the platoon. This is vital for maintaining the coordination needed for safe and efficient platooning.

```
def communicate(self):
    """
    Perform data transfer between vehicles in a platoon.
    Fetches data from leading and front vehicles and passes
    it to the CACC (Cooperative Adaptive Cruise Control) algorithm.
    """
    for vid, l in self.topology.items():
        if "leader" in l.keys():
            # Get data about platoon leader
            ld = self.plexe.get_vehicle_data(l["leader"])
            # Pass leader vehicle data to CACC
            self.plexe.set_leader_vehicle_data(vid, ld)
        if "front" in l.keys():
            # Get data about front vehicle
            fd = self.plexe.get_vehicle_data(l["front"])
            # Pass front vehicle data to CACC
            self.plexe.set_front_vehicle_data(vid, fd)
```

B.3.2 Handling Teleporting Vehicles

The `update_teleport_vehicles` function manages the situation where vehicles are teleported within the simulation (e.g., due to collisions or becoming stuck). When this happens, the function removes these vehicles from the platoons and updates the simulation state to reflect this.

```
def update_teleport_vehicles(self):
    """
    Handles vehicles that have been teleported and removes them from the platoon.
    """
    veh_tlp = traci.simulation.getCollidingVehiclesIDList()
    if len(veh_tlp) != 0:
```

```
        for veh in veh_tlp:
            if veh in self.platoons.keys():
                leader_id = veh
                p = self.platoons[leader_id]["members"]
                for v in p:
                    if v in self.topology.keys():
                        self.topology.pop(v)
                    traci.vehicle.remove(v)
                self.platoons.pop(leader_id)
            elif veh in self.topology.keys():
                leader_id = self.topology[veh]["leader"]
                p = self.platoons[leader_id]["members"]
                for v in p:
                    if v in self.topology.keys():
                        self.topology.pop(v)
                    traci.vehicle.remove(v)
                self.platoons.pop(leader_id)
```

B.4 Platooning Logic

B.4.1 Managing Vehicle Joins

The `do_joins` function manages how vehicles join a platoon. It processes vehicles attempting to join a platoon, handles their interactions, and determines whether the maneuver is successful.

```
def do_joins(self, action):
    """
    Manage the process of a vehicle joining a platoon.
    Adjusts vehicle speed and lane position to safely integrate into the platoon.
    """
    leader_id = self.join_list[0]
    joiner_id = self.join_list[1]
    platoon_front_id = self.join_list[2]
    platoon_behind_id = self.join_list[3]
    flist = self.join_list[4]
    state = self.join_list[5]

    if state == self.REQUEST and leader_id in self.platoons.keys():
        self.platoons[leader_id]["state"] = 1
        if state == self.REQUEST:
            self.catch_platoon(joiner_id, leader_id, platoon_front_id)
            self.join_list[5] = self.CATCHING_PLATOON
    if action == 1:
```

```
19          self.finish_join(joiner_id, leader_id, flist)
```

B.4.2 Managing Vehicle Leaves

The `do_leaves` function manages the process of vehicles leaving a platoon. It ensures that vehicles exit the platoon smoothly and updates the platoon structure once a vehicle has successfully left.

```
1  def do_leaves(self):
2      """
3      Manages vehicles leaving the platoon, ensuring safe disbanding.
4      Updates the platoon structure and handles lane changes for vehicles exiting.
5      """
6      new_leave_list = copy.deepcopy(self.leave_list)
7      for i in range(len(self.leave_list)):
8          leader_id = self.leave_list[i][0]
9          leave_id = self.leave_list[i][1]
10         state = self.platoons[leader_id]["state"]
11
12         if state == 0:
13             flist = self.get_all_following_vehicles(leave_id)
14             self.open_gap_for_leave(leave_id, leader_id, flist)
15             state = self.OPENING_GAP
16
17         if state == self.OPENING_GAP:
18             if leave_id in traci.vehicle.getIDList():
19                 traci.vehicle.changeLane(leave_id, 1)
20                 self.platoons[leader_id]["state"] = self.COMPLETED
21                 self.plexe.disable_fixed_lane(leave_id)
22         self.platoons[leader_id]["state"] = state
```

B.5 Other Plexe Functions

B.5.1 get_distance(self, v1, v2)

This function calculates the Euclidean distance between two vehicles by retrieving their position data (POS_X and POS_Y) and subtracting the vehicle length (4 meters) to obtain the final distance.

```
1  def get_distance(self, v1, v2):
2      v1_data = self.plexe.get_vehicle_data(v1)
3      v2_data = self.plexe.get_vehicle_data(v2)
4      return math.sqrt((v1_data[POS_X] - v2_data[POS_X]) ** 2 +
5                       (v1_data[POS_Y] - v2_data[POS_Y]) ** 2) - 4
```

B.5.2 initialize_traffic_control(self, min_platoon_size=2, max_platoon_size=4)

This function initializes vehicle controllers based on vehicle type and lane assignment. It assigns new vehicles to existing platoons or creates new ones if necessary.

```
def initialize_traffic_control(self, min_platoon_size=1, max_platoon_size=100):
    if len(traci.simulation.getDepartedIDList()) != 0:
        for vid in traci.simulation.getDepartedIDList():
            if traci.vehicle.getTypeID(vid) == "cav":
                self.plexe.set_active_controller(vid, DRIVER)
            if traci.vehicle.getTypeID(vid) == "cav2":
                self.plexe.set_active_controller(vid, ACC)
                self.plexe.set_fixed_lane(vid, self.p_lane, False)
                traci.vehicle.setSpeedMode(vid, 0)
                self.plexe.set_cc_desired_speed(vid, self.SPEED_CAV_LANES)
                if traci.vehicle.getLeader(vid) is None:
                    self.platoons[vid] = {"ini_size": random.randint(min_platoon_size,
    max_platoon_size),
                                          "members": [vid], "state": 0}
                else:
                    vid_front = traci.vehicle.getLeader(vid)[0]
                    leader_id = vid_front if vid_front in self.platoons.keys() else self.
    topology[vid_front]["leader"]
                    if leader_id != 0 and self.platoons[leader_id]["ini_size"] <= len(self.
    platoons[leader_id]["members"]):
                        self.platoons[vid] = {"ini_size": random.randint(min_platoon_size,
    max_platoon_size),
                                              "members": [vid], "state": 0}
                    else:
                        self.plexe.set_active_controller(vid, CACC)
                        self.platoons[leader_id]["members"].append(vid)
                        self.topology[vid] = {"front": vid_front, "leader": leader_id}
```

B.5.3 update_left_vehicles(self)

Updates vehicles that have left the simulation and removes them from the platoons and topology.

```
def update_left_vehicles(self):
    did_1 = self.detector_1
    did_2 = self.detector_2
    did_3 = self.detector_3
    veh_left = traci.inductionloop.getLastStepVehicleIDs(did_3) + traci.inductionloop.
    getLastStepVehicleIDs(did_2) + traci.inductionloop.getLastStepVehicleIDs(did_1)
    if len(veh_left) != 0:
```

```
        for leader in veh_left:
            if leader in self.platoons.keys():
                platoon = self.platoons[leader]["members"][1:]
                for veh in platoon:
                    self.topology.pop(veh, None)
                self.platoons.pop(leader, None)
            elif leader in self.topology.keys():
                self.topology.pop(leader, None)
```

B.5.4 update_quit_vehicle(self, veh)

This function updates the platoon and topology when a vehicle leaves. If the vehicle is a platoon leader, a new leader is assigned; otherwise, the topology is adjusted to remove the vehicle from the system.

```
def update_quit_vehicle(self, veh):
    if veh in self.platoons.keys():
        if len(self.platoons[veh]["members"]) > 1:
            new_leader = self.platoons[veh]["members"][1]
            self.plexe.set_active_controller(new_leader, ACC)
            self.platoons[new_leader] = self.platoons[veh]
            self.platoons[new_leader]["members"].pop(0)
            self.topology.pop(new_leader, None)
            for follower in self.platoons[new_leader]["members"][1:]:
                self.topology[follower]["leader"] = new_leader
            if self.join_list and veh == self.join_list[0]:
                self.join_list[0] = new_leader
        self.platoons.pop(veh, None)
    elif veh in self.topology.keys():
        leader = self.topology[veh]["leader"]
        follower = self.get_following_vehicle(veh)
        self.platoons[leader]["members"].remove(veh)
        if follower != -1:
            self.topology[follower]["front"] = self.topology[veh]["front"]
        self.topology.pop(veh, None)
```