POLITECNICO DI TORINO

Master's Degree in Mechanical Engineering

Master's Degree Thesis

Development and Improvement of Cooperative Adaptive Cruise Control Strategies based on Reinforcement Learning

Supervisor:

Prof. Daniela Anna Misul

Co-supervisors:

Dr. Federico Miretti

Dr. Angelo Borneo

Candidate:

Navid Mehr Alizadeh

July 2024

Abstract

The recent rise of Advanced Driver Assistance Systems (ADAS) and connectivity in the automotive field applied to Connected and Autonomous Vehicles (CAVs) has led the research efforts to investigate new control strategies for improving Mobility solutions.

An alternative approach that gained recent interest and that can be applied to complex control problems is Reinforcement Learning (RL). RL is a branch of Machine Learning that consists of training an Agent to behave in a desired manner, and that has been recently proven to reach comparable or enhanced performance concerning more common optimal control strategies such as Model Predictive Control, especially in terms of computational costs and in presence of high dimensional and uncertain environments.

This thesis is proposed as a prosecution of previous works about Cooperative Adaptive Cruise Control (CACC) based on Reinforcement Learning. In particular, it aims at possible improvements and developments concerning performance, safety, comfort, and energy-saving features for heavyweight and/or lightweight CAVs. Moreover, the RL based controller is validated by considering various speed profiles of the leader vehicle, including a real drive cycle. Results show that the proposed control strategy is capable of quickly responding to unexpected maneuvers and of avoiding collisions between the platooning vehicles, still ensuring a minimum safety distance in the considered driving scenarios.

Acknowledgments

Table of Contents

List of Tables

List of Figures

Acronyms

ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance Systems
CACC	Cooperative Adaptive Cruise Control
CWS	Collision Warning Systems
DAS	Driver-Assistance System
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Networks
DSRC	Dedicated Short-Range Communication
MDP	Markov Decision Processes
MPC	Model Predictive Control
PID	Proportional–integral–derivative
RL	Reinforcement Learning
RMSE	Root Mean Square Error
R2V	Roadside-to-Vehicle
SARSA	State–Action–Reward–State–Action
V2V	Vehicle-to-Vehicle
WLTP	Worldwide Harmonised Light Vehicles Test Procedure

Chapter 1:

1.1 Introduction

The increased number of vehicles on the road causes an increase in traffic accidents, environmental pollution, and a variety of other issues. To address these issues, advanced driver assistance systems (ADAS) are a valuable tool. Indeed, by incorporating this advanced technology, vehicles can improve driving safety, convenience, and environmental friendliness.

ADAS, in this perspective, plays a major role in improving drive safety and comfort. Therefore, they are applied in a multitude of situations to assist the driver in avoiding forward collisions, keeping the lane, braking automatically, and guaranteeing pedestrian safety. The first generation of safety applications was designed using local sensors such as cameras and radars. Then, other sources of information are employed to provide more accurate information. Among them particularly relevant is vehicle-to-vehicle (V2V) communication, which allows one to send information about the state of the vehicle to other vehicles without considering a specific topological position. Indeed, its omnidirectional connectivity capabilities permit to adoption of different topological communication structures to improve performance. This technology plays a main role in the development of new features such as Cooperative Adaptive Cruise Control (CACC).

Cooperative Adaptive Cruise Control is the evolution of Adaptive Cruise Control that takes advantage of V2V communication to acquire information from the surrounding vehicles and drive the vehicle simultaneously avoiding collisions and maximizing traffic throughput. With CACC, the ego vehicle receives data about the speed, position, and acceleration of nearby vehicles, enabling it to define control actions that safely follow the preceding vehicle and form a platoon. To achieve the optimization for multiple objectives of the CACC system, a proper control algorithm is needed. There are many control algorithms for solving multi-objective optimization, for example, dynamic programming, genetic algorithm, and Model Predictive Control (MPC), however, Reinforcement learning is another effective method, because it is used to enhance vehicle coordination, optimize traffic flow, improve safety, and increase fuel efficiency by enabling vehicles to learn optimal driving strategies through experience and interaction with other vehicles.

Therefore, the purpose of this work is to develop a controller to equip vehicles with CACC, enhancing their safety, drivability, and comfort. To achieve this goal, several phases were undertaken: first, a bibliographical analysis was conducted to gather information about different types of Reinforcement Learning (RL) methods and agents and to select the most appropriate one for the case study. Then, the model of the vehicles was developed in MATLAB/Simulink. Afterward, the design of the controller was defined with the aid of the Reinforcement Learning (RL) Toolbox of MATLAB/Simulink. The resulting controller was validated through various drive cycles, demonstrating its effectiveness in enhancing safety and comfort. In the end, to test the reactiveness of the controller, an application involving a four-vehicle platoon was simulated.

1.2 Advanced Techniques for Adaptive Cruise Control

Advanced techniques for Adaptive Cruise Control (ACC) involve several methodologies and technologies that improve the usefulness, safety, and efficiency of the systems.

1.2.1 Model Predictive Control

Model Predictive Control (MPC) is widely adopted in industry as an effective means to deal with multivariable constrained control problems. MPC in a receding horizon fashion performs an optimization in every time-step, yielding state or situation dependent control [1]. Model Predictive Control (MPC) is used in Adaptive Cruise Control (ACC) systems to maximize vehicle speed and trajectory while maintaining safety and comfort. In MPC, the manipulated inputs are computed in real time by solving a mathematical programming problem, most frequently a Quadratic Program. The quadratic program is based on a model of the system's dynamics, which is typically learned from experimental data. To use MPC in embedded control systems with fast sampling and limited CPU and memory resources, you must be able to solve quadratic programs with high throughput, use simple code, perform arithmetic operations with limited machine precision, and provide tight execution time estimates. The MPC uses a model of the plant for making predictions on the future plant output behavior. It also employs an optimizer to ensure that the anticipated future plant outputs track the desired reference (Figure 1).

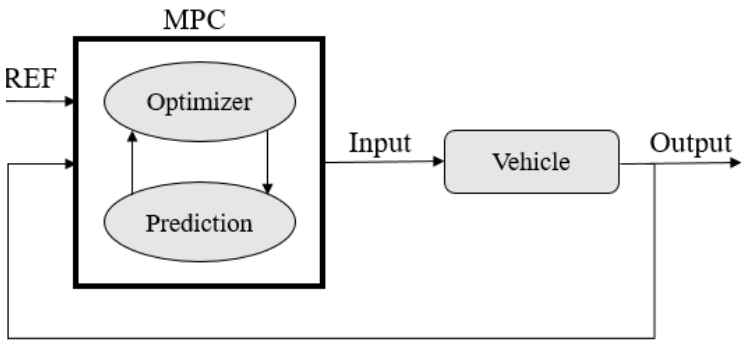

Figure 1. Model Predictive Control Design

Model Predictive Control (MPC) is the most commonly used control method for developing ACC algorithms. MPC control involves solving a finite-horizon optimization problem at each time step to get a sequence of control inputs. Only the first element in the sequence is then applied [2]. The process is repeated at the next time step, with new measurements. MPC excels at managing limits on control inputs (e.g., vehicle acceleration) and system states (e.g., following distance) [3].

By employing a linear and continuous model of car-following dynamics, Luo et al. [4] developed a Model Predictive Control (MPC) controller specifically designed for real-world car-following scenarios. Their approach leverages the principles of MPC to anticipate the future states of both the controlled vehicle and the preceding vehicle, enabling precise adjustments in speed and acceleration to maintain optimal following distances. This controller continuously updates its predictions and control actions, ensuring smooth and safe car-following behavior under varying traffic conditions. The effectiveness of their MPC-based solution highlights its potential for enhancing the performance of Adaptive Cruise Control (ACC) systems in practical driving environments. The MPC aims to regulate the acceleration of the following vehicle to ensure that the relative distance between the two vehicles remains within a safe range. Simulation results indicated that the MPC controller exhibited significantly safer behavior compared to real drivers, effectively maintaining optimal spacing and reducing the risk of collisions. This enhanced safety performance underscores the potential of MPC in improving the reliability and security of car-following systems in dynamic traffic conditions.

Takahama and Akasaka [5] developed a practical Model Predictive Control (MPC)-based Adaptive Cruise Control (ACC) algorithm optimized for low computational cost, making it suitable for implementation on embedded microprocessors. They employed a low-order prediction model to minimize computational demands, ensuring efficient real-time performance. The results demonstrated that their algorithm not only maintained high responsiveness but also significantly reduced driver discomfort, showcasing its effectiveness in enhancing the driving experience while ensuring safety and comfort.

Li et al. [6] introduced an MPC-based Adaptive Cruise Control (ACC) system designed to enhance tracking capability, fuel economy, and alignment with driver preferences. Their approach utilized a quadratic cost function to quantify tracking errors, fuel consumption, and conformity with driver characteristics. Simulation results demonstrated that this ACC system offered significant

improvements in fuel efficiency, precise tracking, and meeting the desired car-following behavior of drivers, highlighting its potential benefits for real-world driving applications.

Lefevre et al. [7] proposed a learning-based approach for autonomous car-following velocity control. Initially, they developed a driver model to replicate human car-following behavior accurately. The outputs of this driver model, specifically vehicle acceleration values, were used as reference points for the MPC controller. By addressing a constrained optimization problem, the MPC controller ensured that the vehicle adhered to the modeled behavior while also meeting essential safety criteria. This approach effectively blended learned driver behavior with formal control techniques to achieve safe and realistic autonomous car-following.

With the preceding studies demonstrating the effectiveness of MPC, this study chose RL for autonomous velocity control for two reasons: (1) RL is considerably faster than MPC during testing [8]. This is because MPC must solve a limited finite-time optimum control problem at each time step, whereas RL only requires states as input and output actions; and (2) RL may outperform MPC, as proved by Lin et al [9].

1.2.2 Proportional–integral–derivative

A proportional–integral–derivative controller (PID controller or three-term controller) is a feedback control loop mechanism extensively utilized in industrial control systems and various other applications requiring continuous control modulation. A PID controller (Figure 2) continuously computes an error value as the difference between a desired set point and a measured process variable. It then applies corrections based on proportional, integral, and derivative terms (denoted P, I, and D, respectively), which is how it gets its name.

PID systems automatically provide precise and responsive adjustments to control functions. A common example is the cruise control system in a vehicle. When the vehicle ascends a hill, maintaining constant engine power would reduce the vehicle's speed. The PID controller's algorithm increases the engine's power output in a controlled manner to restore the vehicle's speed to the desired level with minimal delay and overshoot [10].

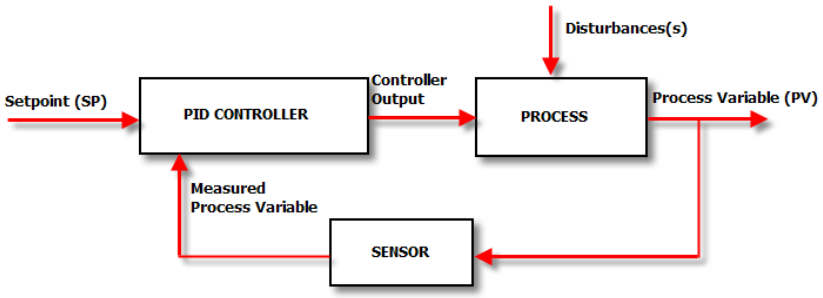

Figure 2. PID Controller Design

By adjusting the gains (K_P, K_I, K_D) assigned to each term, a PID controller can be tuned to adapt to different application requirements. Each term and its corresponding gain play a crucial role in the control process. The proportional term (K_P) produces an output that is directly proportional to the current error $e(t)$, which is the difference between the desired set point and the measured process variable. This term helps to reduce the overall error by adjusting the control signal in proportion to the error. A higher proportional gain (K_P) increases the response speed but can lead to overshoot and instability if set too high.

The integral term (K_I) accounts for the sum of past errors by integrating the error over time, effectively accumulating the total error. This term helps to eliminate steady-state error, which is the residual error that remains after the proportional response. By considering the accumulation of past errors, it ensures that the control signal is adjusted to bring the system back to the desired set point over time.

The derivative term (K_D) predicts future error based on the rate of change of the error, calculating the slope of the error curve and responding to how quickly the error is changing. This term provides a damping effect, improving the stability and response time of the system. It helps to reduce overshoot and settle the system more quickly by counteracting the proportional and integral actions if the error changes rapidly.

The control signal $u(t)$ is generated by summing the contributions of the proportional, integral, and derivative terms, each multiplied by their respective gains (K_P, K_I, K_D). This combined signal is then used to adjust the control variable to achieve the desired outcome.

In some cases, only specific terms of the PID controller are utilized, resulting in variations such as P controllers, PI controllers, PD controllers, or I controllers. Among these, PI controllers are most commonly used because the D term is highly sensitive to noise, while a controller lacking the I term cannot eliminate steady-state error. Typically, a PI controller is employed to remove steady-state error, and a PD controller is used to enhance the system's transient response. A P controller is simple and provides a basic level of control but cannot eliminate steady-state error, and an I controller, which uses only the integral term, can eliminate steady-state error but generally results in a slower response.

Application-specific tuning of a PID controller involves balancing the effects of each term. The derivative term can amplify noise in the control signal, making the system less stable, so it is often used with caution. The integral term is crucial for systems where eliminating steady-state error is important. The proportional and derivative terms are key to improving the system's transient response and accuracy. By adjusting K_P, K_I and K_D, a PID controller can be finely tuned to meet the specific needs of a control application, offering a flexible and robust control strategy capable of addressing a wide range of system dynamics and performance criteria [11].

1.3 Problem Statement and Research Objectives

The increasing deployment of Adaptive Cruise Control (ACC) systems in modern vehicles underscores the critical need for robust and reliable methods to manage vehicle spacing and velocity control. Traditional control strategies such as Model Predictive Control (MPC) and Proportional–Integral–Derivative (PID) controllers have been extensively used in ACC systems, each demonstrating varying degrees of success. However, these methods often face significant challenges in ensuring string stability, a crucial aspect of ACC systems. String stability refers to the system's ability to prevent the amplification of errors as they propagate through a platoon of vehicles. When string stability is compromised, minor errors in the leading vehicle's speed or position can escalate as they travel down the line of following vehicles, potentially leading to passenger discomfort and even causing collisions.

Reinforcement Learning (RL) presents a promising alternative to traditional ACC methods. RL offers the potential to adaptively learn and optimize control strategies based on dynamic environmental feedback, potentially outperforming traditional methods in complex scenarios. The central question this research aims to address is whether RL can be effectively employed to control vehicle velocity and maintain safe distances between vehicles in a longitudinal manner while ensuring string stability. This study seeks to explore the capabilities of RL in enhancing the safety and comfort of passengers within a platoon of self-driving vehicles, addressing the critical issue of error propagation in ACC systems.

The first objective is to conduct an exhaustive review of existing ACC methodologies, including MPC, PID, and RL. This review aims to identify the strengths and limitations of each approach. Based on the insights gained, RL has been selected as the focus for this study due to its adaptive learning capabilities and potential to handle complex, dynamic environments more effectively than traditional methods.

Following the literature review, the next objective is to develop an RL-based ACC model using the Deep Deterministic Policy Gradient (DDPG) algorithm. The DDPG algorithm is particularly suitable for continuous action spaces and offers robust learning capabilities. The RL model will be designed to control vehicle velocity and maintain safe longitudinal distances between the lead

vehicle and the following vehicles. This involves training RL agents to learn optimal control strategies through interactions with the environment, thereby enabling the system to adapt and respond to various driving conditions dynamically.

A critical objective is to investigate whether the RL-based ACC model can ensure string stability within a vehicle platoon. This involves assessing the model's ability to prevent error propagation that could lead to increased spacing errors and potential collisions. Ensuring string stability is vital for maintaining smooth and safe traffic flow in a platoon of autonomous vehicles. The analysis will include various scenarios to test the robustness of the RL model in maintaining string stability under different conditions.

Another key objective is to evaluate the safety and comfort provided by the RL-based ACC system. This will involve analyzing the system's performance under various scenarios, including different communication delay values. The evaluation will focus on the system's ability to maintain safe distances and control velocities effectively, ensuring a comfortable ride for passengers. By simulating real-world driving conditions, the study aims to validate the practical applicability of the RL-based ACC system.

The RL-based ACC model will be implemented in a simulated environment, followed by extensive validation of its performance. The validation process will involve testing the model across multiple driving scenarios to ensure it meets the desired criteria for safe distance maintenance, velocity control, string stability, and reduction of spacing errors among vehicles in the platoon. The implementation will leverage advanced simulation tools to create realistic driving conditions and evaluate the RL model's effectiveness comprehensively.

By addressing these objectives, this research aims to provide a thorough investigation into the feasibility and effectiveness of using RL for ACC systems. The study will demonstrate whether RL can enhance the performance of ACC systems in ensuring safe, comfortable, and string-stable car-following behavior in autonomous vehicle platoons. Additionally, the research will contribute to the broader understanding of RL's potential in automotive applications, paving the way for future advancements in autonomous driving technology.

Chapter 2:

2.1 Introduction to Reinforcement Learning

Reinforcement learning (RL) is a subset of machine learning that enables an AI-powered system (also known as an agent) to learn through trial and error based on feedback from previous actions. This input is either negative or positive, communicated as punishment or reward, with the goal of maximizing the reward function. RL learns from its mistakes and provides artificial intelligence that closely like natural intelligence as is currently achievable [12].

2.1.1 Overview of Reinforcement Learning

Reinforcement Learning (RL) shares a similarity with supervised learning in that both involve mapping between inputs and outputs. However, this is where their similarities end. In supervised learning, the feedback includes the correct set of actions for the agent to follow. In contrast, RL does not provide such an answer key; the agent must determine its actions independently to accomplish the task correctly. Compared to unsupervised learning, RL has fundamentally different objectives. The goal of unsupervised learning is to identify patterns or similarities within the data. In contrast, the objective of RL is to discover the optimal action model that maximizes the agent's total cumulative reward. Without a pre-defined training dataset, RL relies on the agent's interactions with the environment to solve problems [13].

RL techniques such as Monte Carlo methods, State–Action–Reward–State–Action (SARSA), and Q-learning offer a more adaptive and dynamic approach than traditional machine learning methods, paving the way for advancements in the field.

There are three main types of RL implementations:

1- Policy-based RL: Utilizes a policy or deterministic strategy that aims to maximize cumulative rewards.

2- Value-based RL: Focuses on maximizing a value function to determine the best course of action.

3- Model-based RL: Develops a virtual model of the environment, allowing the agent to learn and perform tasks within these constraints.

2.1.2 Basics of Reinforcement Learning

In RL literature, the learner or decision-maker is referred to as the agent, whereas the environment is where the agent lives and interacts. The agent can interact with its surroundings by taking certain actions, but these actions have no effect on the environment's laws or dynamics. In RL, agents act based on the current state of the environment and reward signals.

Rewards and penalties are used to train real-life agents. We reward our agent for good decisions and penalize negative ones. RL algorithms adjust the agent policy based on Prior prizes were gained by doing all feasible acts in various states. RL aims to provide a decision-making approach that maximizes total rewards or returns. In most RL issues, the return is often a user-defined reinforcement signal accumulated through immediate rewards. The agent uses a reward signal created by interaction with the environment to evaluate the optimal policy [14]. Figure 3 illustrates the framework presented in this section.

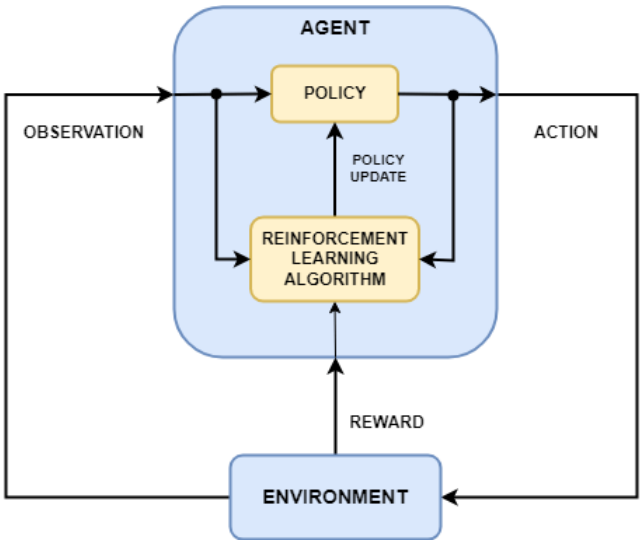

Figure 3. Block diagram of reinforcement learning [15]

If the agent performs any action u in a specific state x, the anticipated immediate reward can be represented as:

$$r_{k+1}(x,u) = E[r_{k+1}|x_k = x, u_k = u] \quad (1)$$

If the sequence of immediate rewards received after time step k is $\{r_{k+1}, r_{k+2}, r_{k+3} \dots, r_k\}$ for an episodic task, then the return G_k for this episodic task can be expressed as:

$$G_k = r_{k+1} + r_{k+2} + r_{k+3} + \cdots + r_k \quad (2)$$

In this context, k denotes the final time step. At each time step, the agent receives a description of the current state of the environment and chooses an action based on this state as well as previously observed rewards for the same or similar states. The final state signal provided to the RL agent is typically a preprocessed version of the original sensor data. Although the state can include information about past observations, it should not encompass all information about the environment [14].

An optimal state signal that encapsulates only the present state information of the environment in a compact form is known as Markov. The conditional probability distribution of future states in a stochastic process exhibits the Markov property if it relies solely on the current state. Markov Decision Processes (MDPs) are effective for modeling sequential decision-making problems [16]. Reinforcement Learning (RL) utilizes the formal framework of MDPs to represent the interaction between an agent and the environment in terms of states, actions, and rewards. A finite MDP is entirely defined by the sets of all possible states and actions, along with the one-step model dynamics of the environment. For a finite MDP, the one-step model dynamics are characterized by the state transition probability and the expected reward for a given state-action pair (x, u) [17],

$$p(x', r|x, u) = P\{x_k = x', r_k = r|x_{k-1} = x, u_{k-1} = u\} \quad (3)$$

Nearly all reinforcement learning (RL) algorithms employ an estimated state value function $V_\pi(x)$ and a state-action value function $Q_\pi(x, u)$ to evaluate how beneficial it is for the agent to be in a specific state or to take a particular action within that state. These value functions are crucial in RL for approximating optimal policies. They are predominantly based on expected future rewards or returns. The expected return for a given policy π is $V_\pi(x)$ if the agent starts from state x and then follows the policy π. For a given policy, the state value function $V_\pi(x)$ can be expressed as [17]:

$$V_\pi(x) = E_\pi[G_k|x_k = x] = E_\pi\left\{\sum_{n=0} \gamma^n R_{k+n+1}|x_k = x\right\}, \forall\, x \,\epsilon\, X \quad (4)$$

Here, γ represents a constant, with values between 0 and 1 inclusive, referred to as the discount factor. This factor determines the weight given by an RL agent to future rewards in comparison to the current reward. When γ is set to 0, the agent behaves in a purely myopic manner, focusing solely on actions that yield the highest immediate reward. If an agent begins at state x, executes action u, and subsequently adheres to policy π, the anticipated outcome is termed the state-action value function $Q_\pi(x, u)$ and is represented as [17]:

$$Q_\pi(x, u) = E_\pi[G_k|x_k = x\,, u_k = u] = E_\pi\left\{\sum_{n=0} \gamma^n R_{k+n+1}|x_k = x\,, u_k = u\right\}, \forall\, x\epsilon X \text{ and } u\epsilon U \quad (5)$$

The state value $V_\pi(x)$ of a state theoretically represents the average reward an agent would receive from visiting that state numerous times. In a similar manner, the average reward an agent would obtain by consistently performing a specific action u in a given state x is the state's action value $Q_\pi(x, u)$. These value functions can be mathematically calculated using the following equations [17]:

$$V_\pi(x) = \sum_u \pi(u|x) \sum_{x',r} p(x', r|x, u)[r + \gamma V_\pi(x')] \quad (6)$$

$$Q_\pi(x, u) = \sum_{x',r} p(x', r|x, u)\,[r + \gamma \sum_u \pi(u'|x')\, Q_\pi(x', u')] \quad (7)$$

$V_\pi(x)$ and $Q_\pi(x, u)$ can be efficiently estimated using the agent's interaction data. The value functions expressed in equations (6) and (7) are known as Bellman equations, which represent a recursive relationship between the value functions of current and successor states.

2.1.3 Value-based reinforcement learning

A value function is a fundamental concept in reinforcement learning (RL) that evaluates the utility or quality of being in a specific state or performing a particular action in a given state. The action

value function, denoted as $Q(s,a)$, is defined as $Q(s,a) = E[R_t|s_t = s, a_t = a]$, where E represents the expected value. This function captures the expected cumulative reward that an agent will receive if it takes action a in state s and then continues to follow a policy π. Essentially, it represents the long-term value of taking a specific action from a given state.

Value-based reinforcement learning methods focus on learning this action value function from the agent's past interactions with the environment. By leveraging historical experience, these methods aim to determine which actions are most beneficial in various states to maximize the agent's total rewards. One of the most prominent value-based RL algorithms is Q-learning. Q-learning is designed to find the optimal action value function, which guides the agent in making decisions that maximize expected returns over time.

The Q-learning algorithm initializes the Q-function arbitrarily, meaning that it starts with random estimates of the Q-values for each state-action pair. As the agent explores the environment and gathers more data, it updates these Q-values using the Bellman equation, which provides a recursive relationship for the expected rewards. The update rule in Q-learning is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \quad (8)$$

Breaking down this equation:

- α is the learning rate, which determines how much new information overrides the old information. A higher learning rate means that the agent learns more quickly, but it might also result in more fluctuation in learning.

- r is the immediate reward received after taking action a in state s. This reward provides immediate feedback on the action's short-term effectiveness.

- γ is the discount factor, which ranges between 0 and 1. It dictates the importance of future rewards compared to immediate rewards. A higher discount factor means the agent values future rewards more, leading to long-term planning.

- s' is the next state resulting from taking action a.

- $max_{a'} Q(s',a')$ represents the maximum expected future reward for the next state s' considering all possible actions a'.

This iterative process allows the Q-learning algorithm to estimate the optimal action value function. Once the Q-values converge, the agent can derive the optimal policy. The optimal policy is to select the action with the highest Q-value in each state, thereby maximizing the expected cumulative future rewards.

Value-based methods like Q-learning are particularly advantageous for problems with discrete and manageable state and action spaces. They provide a straightforward approach to learning optimal policies by directly estimating the value of actions. However, when the state or action space is large or continuous, value-based methods can become computationally expensive and may require function approximation techniques to generalize across states and actions [17].

In summary, value functions in reinforcement learning provide a measure of the expected return for being in a particular state or taking a specific action. Q-learning, a key value-based RL method, uses the Bellman equation to iteratively update Q-values, guiding the agent to learn optimal policies that maximize long-term rewards. This approach is foundational in RL, offering a clear and effective way to tackle sequential decision-making problems.

2.1.4 Policy-based reinforcement learning

Unlike value-based methods, which focus on estimating the value of states or state-action pairs to derive the best action, policy-based methods aim to directly optimize the policy $\pi(s:\theta)$. These methods adjust the policy parameters θ by using gradient ascent to maximize the expected return $E[R_t]$.This approach involves updating the policy parameters to improve the likelihood of selecting actions that yield higher rewards.

One prominent policy-based method is REINFORCE. It operates by adjusting the policy parameters θ in the direction that increases the expected return. The key idea is to compute the gradient of the expected return concerning the policy parameters and then update the parameters accordingly. The update rule for REINFORCE is given by [18]:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi\,(a_t|s_t;\theta) R_t \tag{9}$$

Here, α is the learning rate, $\pi(a_t|s_t;\theta)$ is the probability of taking action a_t given state s_t under the policy parameterized by θ, and R_t is the cumulative reward obtained after taking action a_t in state s_t.

The gradient $\nabla_\theta \log \pi\,(a_t|s_t;\theta)$ represents how sensitive the log-probability of the action taken is to changes in the policy parameters. By multiplying this gradient by the reward R_t, the method ensures that actions leading to higher rewards are more likely to be taken in the future. This process is repeated for each step in the episode, allowing the policy to learn and improve over time.

REINFORCE and other policy-based methods are particularly useful in environments with high-dimensional action spaces or when dealing with stochastic policies. They are advantageous in scenarios where it is challenging to compute the value functions required by value-based methods. By directly optimizing the policy, these methods can handle complex tasks with continuous or discrete action spaces and are capable of learning intricate strategies that value-based methods might struggle [19].

Overall, policy-based methods, such as REINFORCE, provide a powerful framework for directly optimizing the policy in reinforcement learning, enabling agents to effectively learn and adapt in dynamic and uncertain environments.

2.1.5 Deep Q-Network

Instead of calculating $Q(s,a)$ for each state-action pair individually, deep Q-learning employs neural networks as function approximators to estimate the action-value function [20]. This approach involves training a neural network to predict the value $Q(s,a)$ for given state-action pairs, and the agent selects actions based on the maximum $Q(s,a)$ value output by the network.

While Deep Q Networks (DQN) are effective in environments with discrete action spaces, they struggle with continuous action spaces, which are common in many real-world applications. To overcome this limitation, Lillicrap et al. [21] proposed an advanced algorithm known as Deep Deterministic Policy Gradient (DDPG).

DDPG extends the DQN framework by incorporating an actor-critic mechanism, making it suitable for continuous control problems. In this approach, there are two main components: the actor and the critic. The actor is responsible for selecting actions given the current state, while the critic evaluates the action chosen by the actor by estimating the corresponding $Q(s, a)$ value.

The actor network directly outputs the continuous actions, rather than selecting from a discrete set. The critic network, on the other hand, learns to approximate the action-value function, guiding the actor's updates by providing feedback on the quality of actions taken. This combination allows DDPG to handle the complexities of continuous action spaces effectively.

DDPG operates by using two neural networks for both the actor and the critic, with an additional set of target networks to stabilize training. The algorithm updates the actor network by following the gradient of the expected return, as estimated by the critic, and simultaneously updates the critic network by minimizing the temporal difference error. This actor-critic architecture enables DDPG to learn robust policies in continuous action environments, such as robotic control or autonomous driving.

By leveraging the strengths of deep learning and reinforcement learning, DDPG provides a powerful method for addressing continuous control problems, demonstrating the versatility and capability of neural network-based approaches in complex, high-dimensional tasks.

2.1.6 Deep deterministic policy gradient

DDPG employs two distinct neural networks to approximate the actor and critic functions [21]. The critic network, parameterized by θ^Q, estimates the action-value function $Q(s, a|\theta^Q)$. Meanwhile, the actor network, parameterized by θ^μ, explicitly represents the agent's policy $\mu(s, \theta^\mu)$. To ensure stable and robust learning, DDPG incorporates experience replay and target networks, as originally proposed in DQN.

Experience replay is used to break the correlation between sequentially generated experience samples, which can negatively impact learning. It involves a replay buffer, a finite-sized cache D that stores transitions (s_t, a_t, r_t, s_{t+1}) sampled from the environment. This buffer is continuously

updated, replacing old samples with new ones. During training, the actor and critic networks are updated using random mini-batches of transitions drawn from this buffer. This technique ensures that the learning algorithm trains on a more diverse set of experiences, promoting better generalization and stability.

Target networks are introduced to stabilize the learning process and prevent the divergence of the algorithm. Two target networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s, \theta^{\mu'})$, mirror the main critic and actor networks, respectively. While these target networks share the same architecture as their main counterparts, they have different parameters. The parameters of the target networks are updated to slowly track the parameters of the main networks using a soft update rule:

$$\theta' \leftarrow \theta + \alpha \nabla_\theta \log \pi\,(a_t|s_t;\theta) R_t \tag{10}$$

where τ is a small value, typically close to 0. This gradual update ensures that the target values evolve slowly, which greatly enhances the stability of the learning process by reducing the variance of target estimates and preventing abrupt changes in network parameters.

By employing these mechanisms, DDPG effectively balances exploration and exploitation, leveraging past experiences and stable target estimates to learn robust policies for continuous control problems. This approach has proven particularly useful in domains such as robotic manipulation and autonomous driving, where the ability to handle high-dimensional state and action spaces is crucial [20].

2.2 Reinforcement Learning in Adaptive Cruise Control

2.2.1 Advancements in Driver-Assistance Systems

Recent advancements in sensing, communication, and computing technologies have resulted in the creation of driver-assistance systems (DASs). These systems are designed to help drivers by either providing warnings to prevent collisions or taking over certain control tasks to alleviate the burden of repetitive and monotonous tasks. Consequently, a DAS can assume some of the driver's decision-making and actions in routine scenarios, minimizing the risk of human error that could lead to accidents, while also achieving more consistent and smoother vehicle control [22].

This approach offers three main advantages: increased driver comfort, enhanced traffic capacity, and energy and environmental benefits. According to Piao and McDonald [22], driver-assistance systems (DASs) cover three key areas: adaptive cruise control (ACC) and collision warning and avoidance (CWA); legal considerations; and implementation aspects. ACC is designed to alleviate the driver from manual control adjustments to maintain safe cruising, while CWA focuses on reducing rear-end collisions by issuing appropriate warnings. The legal aspects involve studying the regulatory framework and market introduction of DASs by evaluating their various functions. Lastly, implementing a DAS is a complex task that involves a wide range of technologies, user preferences, and government policies.

Both ACC and CWA can be classified into two types: autonomous systems and cooperative systems. In an autonomous system, the vehicle's control mechanism relies solely on information gathered by its own sensors. In contrast, cooperative systems require communication with nearby vehicles or transportation infrastructure. This communication can occur from vehicle to vehicle (V2V) or from the road to a vehicle (R2V). V2V communication enables a group of equipped vehicles to form a "virtual" network connected by wireless ad hoc communication. R2V communication can be achieved through various technical methods, such as visible light, optical beacons, or the 5.9-GHz Dedicated Short-Range Communications (DSRC) standard. Analytical solutions to control problems such as ACC or cooperative adaptive cruise control (CACC) are often difficult to obtain due to the nonlinear dynamics and high-dimensional state spaces involved.

Linearization generally provides limited assistance in these scenarios, making it more beneficial to explore alternative approaches, particularly reinforcement learning (RL), which does not require knowledge of the underlying Markov decision process (MDP). [23].

New technologies that actively intervene in and control car driving can significantly enhance comfort, safety, and traffic flow. Adaptive Cruise Control (ACC) and Collision Warning Systems (CWA) are examples of such technologies. ACC, in particular, is increasingly being integrated into passenger cars. The primary goal of ACC is to automatically ensure safe cruising, thus alleviating the driver from the monotonous and repetitive task of manual speed adjustment. In free-flowing traffic, the ACC system maintains a preset speed, similar to conventional cruise control systems. However, when following another vehicle, the ACC system automatically adjusts to maintain a desired time gap from the vehicle ahead [24].

An Adaptive Cruise Control (ACC) system can be designed using either an autonomous approach, which employs ranging sensors, or a cooperative approach, which utilizes vehicle-to-vehicle (V2V) and/or roadside-to-vehicle (R2V) communication. Ranging sensors, such as radars or lasers, are typically used to measure the distance and the rate of change of this distance from the preceding vehicle. Generally, ACC systems deactivate at speeds below 30 km/h, as they are primarily designed for highway traffic. Autonomous ACC systems relying on ranging sensors have limited predictive capabilities because they cannot respond to events occurring beyond the immediate preceding vehicle [23].

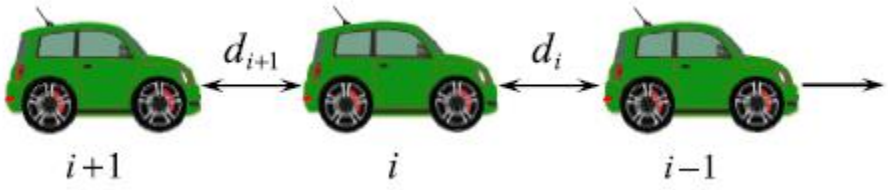

Figure 4. Adaptive cruise control vehicles [25]

In the development of Cooperative Adaptive Cruise Control (CACC) systems, de Bruin et al. [26] provided a detailed overview of the design and implementation of such systems. They identified four critical components necessary for effective CACC functionality:

Positioning System: This component is essential for accurately determining the location of each vehicle within the platoon. By knowing the precise positions, vehicles can maintain optimal spacing and alignment, which is crucial for coordinated movement and safety.

World Model: The world model serves as a comprehensive representation of the driving environment. It includes information about the road layout, traffic conditions, and the positions of surrounding vehicles. This model helps each vehicle to make informed decisions based on a holistic view of the environment.

Controller System: The controller system is responsible for managing the vehicle's speed and acceleration. It processes input from the positioning system and the world model to ensure that the vehicle maintains a safe following distance, adjusts speed appropriately, and executes smooth acceleration and deceleration.

Inter-Vehicle Communication System: This system facilitates the exchange of information between vehicles, particularly preview information from vehicles further ahead. By sharing data about speed, position, and upcoming traffic conditions, vehicles can anticipate changes and react more efficiently. This results in smoother braking and acceleration, reducing the severity of braking actions and enhancing overall traffic flow.

Their initial test results highlighted the effectiveness of this approach. Specifically, they found that the anticipatory braking enabled by the inter-vehicle communication system allowed upstream vehicles to brake more gently when a downstream vehicle decelerated. This contrasts sharply with scenarios where such communication was absent, leading to abrupt braking and potential safety hazards.

In a parallel study, Naus et al. [27] concentrated on a practical implementation of CACC, with a strong emphasis on feasibility and real-world application. They proposed a streamlined communication strategy that focused on interactions with the directly preceding vehicle, rather than attempting to communicate with multiple vehicles or a central platoon leader. This approach simplifies the system, reducing the complexity and potential points of failure.

Additionally, Naus et al. [27] implemented the communication as a feedforward signal. This means that the information from the preceding vehicle is used to inform the actions of the following vehicle in real-time. Importantly, this design includes a fallback mechanism: if the communication

link is interrupted or unavailable, the vehicle defaults to standard Adaptive Cruise Control (ACC) functionality. This ensures that the vehicle can continue to operate safely, even in the absence of cooperative communication, thus maintaining safety and reliability under varying conditions.

Together, these studies underscore the potential of CACC systems to significantly enhance vehicle safety, comfort, and traffic efficiency by leveraging advanced communication and control technologies.

2.2.2 Case Studies of Reinforcement Learning in ACC Systems

Most projects on CACC have traditionally relied on classical control theory to develop autonomous controllers. However, recent research from the machine learning community has shown promising theoretical and practical results in addressing control problems in uncertain and partially observable environments. It would be beneficial to test these machine learning approaches on CACC systems. One of the pioneering research efforts to utilize machine learning for autonomous vehicle control was Pomerleau's autonomous land vehicle. This project involved a neural network-based computer vision system that learned to correlate road observations with the appropriate actions. Remarkably, this autonomous controller successfully drove a real vehicle for over 30 miles independently [28].

Yu [29] was the pioneer in proposing the use of reinforcement learning (RL) for steering control. According to Yu, RL enables control designers to dispense with the need for external supervision while providing continuous learning capabilities. RL is a machine learning approach framed as the adaptive optimal control of a process P, where the controller, known as the agent, interacts with P and learns how to control it. Through trial-and-error interactions, the agent learns to behave optimally. It perceives the state of P and acts to maximize the cumulative return, which is based on a real-valued reward signal received after each action from P. Consequently, RL modifies the control policy, which links an action A to a state S, based on feedback from the environment. This approach is closely related to adaptive control, a well-regarded set of techniques within the control systems community [30].

Se-Young et al. [31] explored road following using reinforcement learning (RL) in conjunction with vision. With RL, the control system acquires knowledge of the dynamics of vehicle-road interaction indirectly, a critical aspect for effectively maintaining course on high-speed roadways.

Moriarti et al. [32] introduced an approach that combines supervised learning with RL to develop lane-selection strategies through trial-and-error interactions with the traffic environment. Through simulations, the authors assessed their method and observed that, in comparison to both a selfish strategy and the conventional "yield to the right" strategy, their intelligent vehicles maintained speeds closely aligned with their drivers' preferences while minimizing the frequency of lane changes.

Forbes's research [33]focused on developing a vehicle controller using instance-based reinforcement learning (RL). The study utilized stored instances of previous observations as value estimates for controlling autonomous vehicles, which were then extended to handle automobile control tasks. An adaptable simulation environment and a hierarchical control architecture for autonomous vehicles were established. Controllers derived from this architecture underwent evaluation and refinement within the simulator, with a focus on addressing challenging traffic scenarios across various simulated highway networks. However, this approach is constrained by memory length, which can quickly escalate when dealing with real-world applications.

Ng et al. [34] introduced an adaptive control system utilizing gain scheduling acquired through reinforcement learning (RL). This approach aims to preserve the nonlinear characteristics of vehicle dynamics, contrasting with a simplistic linearization of the longitudinal model, which may lack suitability across the vehicle's entire operational range. Evaluations of the proposed controller at specific operating points demonstrated precise tracking of both velocity and position in most instances. However, when deployed within a convoy or platoon, the tracking performance exhibited slight oscillations, particularly as the second vehicle endeavored to follow the lead vehicle. While these oscillations were transmitted to subsequent vehicles in the platoon, they diminished as the distance from the lead vehicle increased, indicating stability. Consequently, this approach appears more favorable for platooning control compared to Cooperative Adaptive Cruise Control (CACC), as the latter may induce minor oscillations.

Chapter 3:

3.1 Reinforcement Learning Applied in This Study

Reinforcement learning (RL) tackles sequential decision-making problems by enabling an RL agent to interact with an environment. At each time step, the agent observes a state s_t and selects an action a_t from an action space A according to a policy that maps states to actions. The system then provides a reward r_t to the agent and transitions to the next state s_{t+1}. This process repeats until a terminal state is reached, after which the agent starts over. The objective of the agent is to maximize the discounted accumulated reward $R_t = \sum_{k=0} \gamma^k t_{t=k}$, where γ is the discount factor in the range (0, 1] [17].

3.1.1 DDPG Agent Architecture

The deep deterministic policy gradient (DDPG) algorithm is a model-free, online, off-policy reinforcement learning method that has proven effective for continuous control tasks. In this study, the focus is on developing a robust DDPG agent, which operates as an actor-critic reinforcement learning agent. The DDPG agent is designed to explore and refine an optimal policy that maximizes the expected cumulative long-term reward. By leveraging the actor-critic framework, the DDPG agent simultaneously learns a policy (actor) and a value function (critic), enabling efficient and stable learning in complex environments. This dual-network approach allows the agent to make precise decisions and adjustments, facilitating the achievement of superior performance in dynamic and uncertain settings.

During training, the DDPG agent:

1- Continuously updates the actor and critic parameters at each time step.

2- Utilizes a circular experience buffer to store past experiences, from which it randomly samples a mini-batch of experiences to update the actor and critic.

3- Applies stochastic noise to the action selected by the policy at each training step to ensure exploration.

The comprehensive DDPG algorithm implemented in this study is outlined in Algorithm 1. The process begins with initializing the replay buffer, actor and critic networks, and their corresponding target networks. At each time step, an action a is selected based on the current exploratory policy, incorporating stochastic noise for exploration. The environment then provides a reward r_t and transitions to a new state s_{t+1}. This transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer D, which is continuously updated to maintain a diverse set of experiences.

Training involves sampling mini-batches of transitions from the replay buffer. These mini-batches are used to train the critic network by minimizing the loss between the predicted Q-values and the target Q-values, which are computed using the target critic network. The actor network is then updated by performing a gradient ascent step to maximize the expected return, utilizing the sampled policy gradient from the critic's feedback.

To ensure stability in training, the target networks for both the actor and critic are updated slowly to track the parameters of the main networks. This is achieved by a soft update mechanism where the parameters of the target networks are incrementally adjusted towards the parameters of the main networks. This algorithm effectively combines the strengths of experience replay and target networks to address the challenges of training in high-dimensional continuous action spaces, ensuring robust and stable learning outcomes.

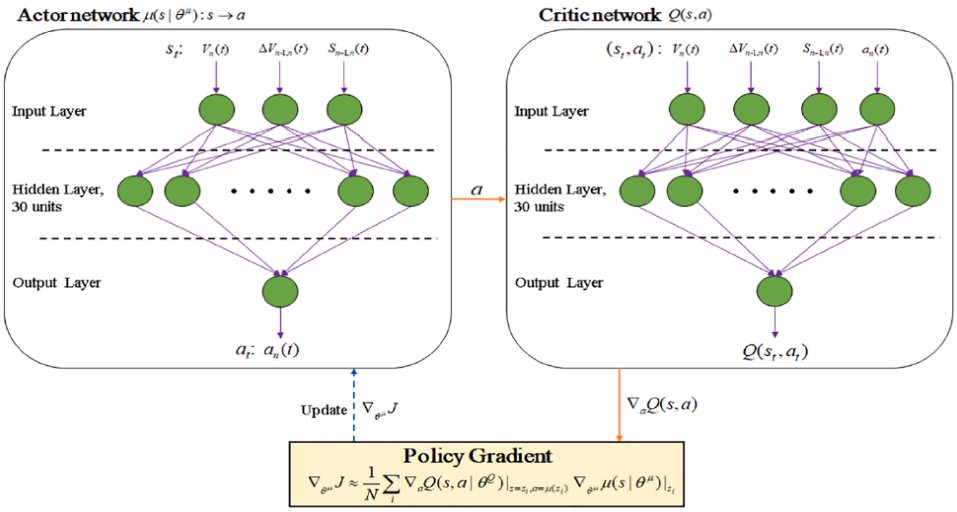

Figure 5. Architecture of the actor and critic networks [3].

Algorithm 1. DDPG: Deep deterministic policy gradient for car-following velocity control

Randomly initialize critic $Q(s, a|\theta^Q)$ and actor $\mu(s, \theta^\mu)$ networks with weights θ^Q and θ^μ

Initialize target network $Q'(s, a|\theta^{Q'})$ and $\mu'(s, \theta^{\mu'})$ with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$

Set up empty replay buffer D

for episode = 1 to M **do**

    Begin with a random process N for action exploration

    Observe initial car-following state: initial gap, follower speed, and relative speed

    for t = 1 to T **do**

        Calculate reward r_t

        Choose follower acceleration $a_t = \mu(s_t, \theta^\mu) + N_t$ based on current actor network and exploration noise N_t

        Implement acceleration at and transfer to new state s_{t+1} based on kinematic point-mass model

        Save transition $(s_t,\ a_t,\ r_t, s_{t+1})$ into replay buffer D

        Sample random minibatch of N transitions $(s_i,\ a_i,\ r_i, s_{i+1})$ from D

        Set $y_i = r_i + \gamma Q'(s_{i+1},\ \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic through minimizing loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update actor policy using sampled policy gradient: $\nabla_\theta \mu J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$

        Update target networks: $\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$

        $$\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for

end for

To implement this agent in MATLAB and Simulink, MathWorks [35] Reinforcement Learning Toolbox was utilized. This toolbox provides the necessary tools and functions to create, train, and simulate reinforcement learning agents. In this study, the DDPG agent was designed and integrated using this toolbox. Figure 6 illustrates the agent block utilized in the Simulink model. This block includes the configuration of the actor and critic networks, the replay buffer, and the target networks, all of which are essential components for the DDPG algorithm. The integration with Simulink allows for seamless simulation and testing of the agent within a dynamic system environment, facilitating the development and evaluation of the reinforcement learning strategy. The flexibility and powerful features of the Reinforcement Learning Toolbox enable efficient implementation and experimentation with advanced RL algorithms like DDPG.

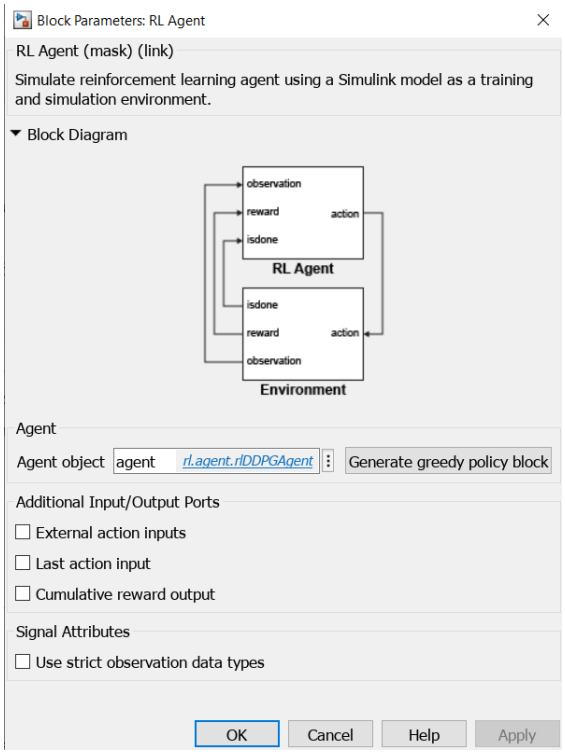

Figure 6. DDPG agent Simulink block

The inputs for this DDPG agent consist of the observation data, which includes the velocity error and the ego vehicle's velocity. Additionally, the agent receives the reward value generated by the reward function at each step. These inputs are crucial for the agent to understand the current state of the environment and the effectiveness of its previous actions. The observation data helps the

agent to assess the difference between the desired and actual velocities, while the reward value provides feedback on the performance of the agent's actions.

The output of the DDPG agent is the controlled action, which is a decision made by the agent on how to adjust the vehicle's control variables to optimize performance. This action is aimed at minimizing the velocity error and maximizing the cumulative reward over time, ensuring that the vehicle operates smoothly and efficiently according to the defined objectives. By continuously adjusting its actions based on the observations and rewards, the agent learns to improve its control strategy and achieve better results in managing the vehicle's behavior.

3.1.2 Environment Configuration

The reinforcement learning environment designed for this study simulates the longitudinal dynamics of an ego car and a lead car. The primary training objective for the agent is to ensure that the ego car maintains a set velocity while keeping a safe distance from the lead car. This is achieved by controlling the longitudinal acceleration and braking of the ego car.

After creating the DDPG agent, it is crucial to establish an environment where the agent can be trained. This environment is designed to simulate the conditions under which the agent will operate and interact. Figure 7 illustrates the environment block, which accepts an acceleration command input from the agent and consists of several key subsystems.

The Lead Car Subsystem simulates the lead car's behavior. It takes inputs such as initial velocity, position, and acceleration and outputs the actual position and velocity of the lead car. This allows the ego car to respond dynamically to the lead car's movements.

The Ego Car Subsystem represents the ego car, which is controlled by the DDPG agent. The agent's goal is to make the ego car travel at a desired velocity while maintaining a safe distance from the lead car. The ego car's actions are determined by the agent based on the observations and rewards it receives.

The Signal Processing for ACC block processes signals necessary for adaptive cruise control (ACC). It integrates various observations and computes the required signals to assist the agent in making informed decisions.

The environment block produces several outputs:

1- RL Observations: These include the velocity error (the difference between the desired and actual velocities) and the ego car's velocity. These observations are fed into the agent, enabling it to evaluate the current state and decide on the next action.

2- Constraint Observations: These observations include:

- Relative Distance: The distance between the ego and lead cars.
- Lead Car Velocity: The current speed of the lead car.
- Ego Car Velocity: The current speed of the ego car.
- Ego Car Acceleration: The current acceleration of the ego car.

These constraint observations are critical for ensuring the agent operates within safe and realistic bounds, preventing collisions and unsafe driving behaviors.

3- Reward Value: The reward value is calculated at each time step based on the reward function, which incentivizes desirable behaviors, such as maintaining a safe distance and achieving the desired velocity. The reward guides the agent's learning process, encouraging actions that maximize long-term benefits.

4- isDone Signal: This output is a signal that indicates whether a training episode should be terminated. If certain conditions are not met (e.g., if the ego car violates safety constraints), the isDone signal triggers the end of the current training step. This ensures that the agent only learns from valid and safe experiences.

5- New Acceleration Value: This output represents the updated acceleration command given to the agent. It completes the feedback loop, allowing the agent to continuously interact with the environment and learn from its actions.

Overall, this environment setup is designed to provide a realistic and controlled training ground for the DDPG agent. By simulating the dynamics of both the ego and lead cars and processing relevant signals, the environment enables the agent to learn effective strategies for longitudinal control, ensuring safe and efficient driving behavior.

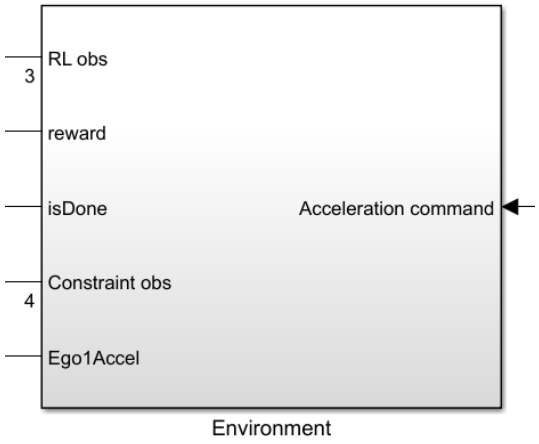

Figure 7. Environment block

Initially, the velocities of both the lead car and the ego car can vary between 1 and 25 m/s. The initial relative distance between the two cars is set to any value greater than 5 meters. The acceleration limits for the ego car range between 2 m/s^2 and -3 m/s^2, allowing for both acceleration and braking.

Figure 8 illustrates the lead car block, where the inputs include the initial velocity, initial position, and acceleration. The output from this block consists of the actual position and actual velocity of the lead car, which are calculated based on the provided inputs. These outputs are essential for the ego car to adjust its speed and maintain the desired safe distance, ensuring a smooth and controlled driving experience.

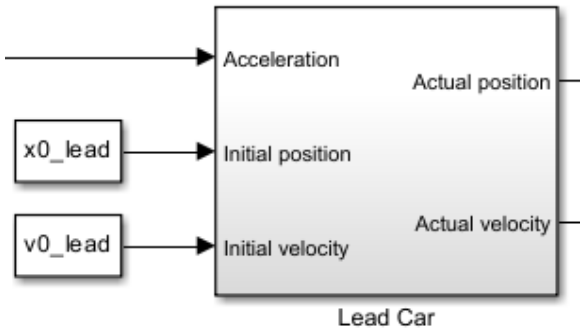

Figure 8. Lead Car block

Figure 9 depicts the subsystem model of the lead car, which processes the car's acceleration to determine its velocity and position. The subsystem starts with receiving the acceleration input, which is then passed through an integrator block. This integrator calculates the integral of the acceleration, effectively converting it into a change in velocity over time. By doing so, it simulates how the car's speed increases or decreases based on the acceleration.

Following the integrator block, the output is processed through a transfer function, specifically 1/(0.5s+1). This transfer function models the dynamic response of the lead car, smoothing the acceleration input to better replicate real-world conditions. The use of the transfer function is crucial as it accounts for the physical properties of the car, ensuring that the acceleration changes are applied in a realistic manner.

The smoothed output from the transfer function is then added to the initial velocity of the lead car. This summation incorporates both the initial speed and the acceleration-induced change in speed, yielding the actual velocity of the lead car. The resultant value is provided as the actual velocity output, which reflects the current speed of the car as it moves.

To determine the distance covered by the lead car, the actual velocity is integrated once more. This integration calculates the displacement or distance traveled over time, converting the velocity data into positional data. The calculated distance is then added to the initial position of the lead car. This summation includes the starting point of the car and the distance it has traveled, resulting in the actual position output.

In summary, the subsystem for the lead car takes the acceleration input and processes it through integration and a transfer function to determine the velocity. It then integrates the velocity to

calculate the position. By adding the initial velocity and position at respective stages, the model provides accurate real-time outputs for both the velocity and position of the lead car. This detailed modeling ensures that the dynamics of the lead car are realistically represented, which is crucial for training and evaluating the DDPG agent in a simulated environment.

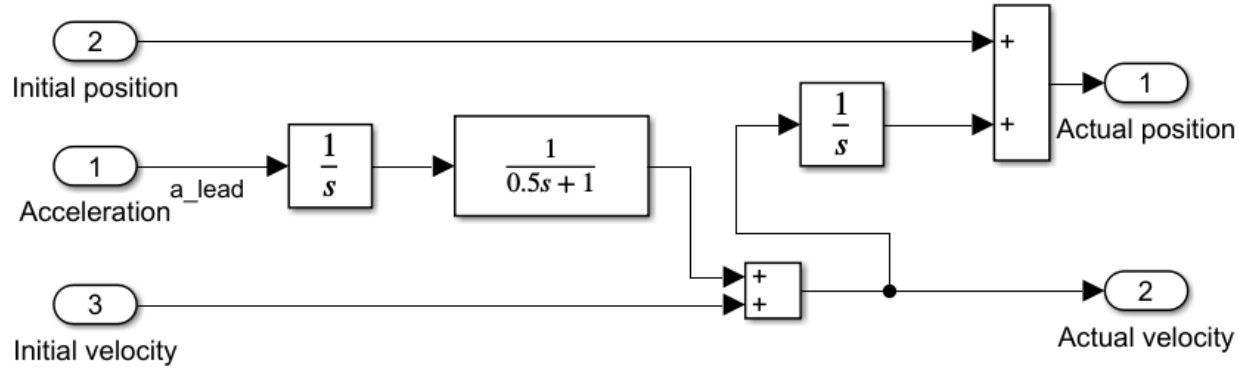

Figure 9. Lead Car subsystem

Figure 10 illustrates the subsystem model for the ego car. The inputs to this block include the initial velocity, initial position, and the acceleration command provided by the agent. This subsystem processes these inputs to produce the actual position, actual velocity, and actual acceleration of the ego car. The calculations within this block are performed similarly to those in the lead car subsystem. The acceleration command from the agent is integrated and processed through a transfer function to determine the current velocity. This velocity is then integrated to calculate the distance traveled by the ego car. By adding the initial velocity and position at the respective stages, the model generates real-time outputs for the car's actual velocity and position, along with the current acceleration, accurately reflecting the car's dynamic state.

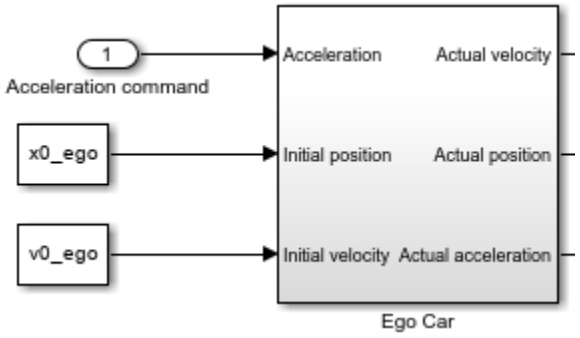

Figure 10. Ego Car block

The Signal Processing for ACC block processes multiple inputs, including the ego car's velocity, the lead car's velocity, the relative distance between the two vehicles, the ego car's acceleration, and the acceleration command from the agent. Using these inputs, it computes several outputs: the RL observation, the reward value, the isDone function, and the constraint observation.

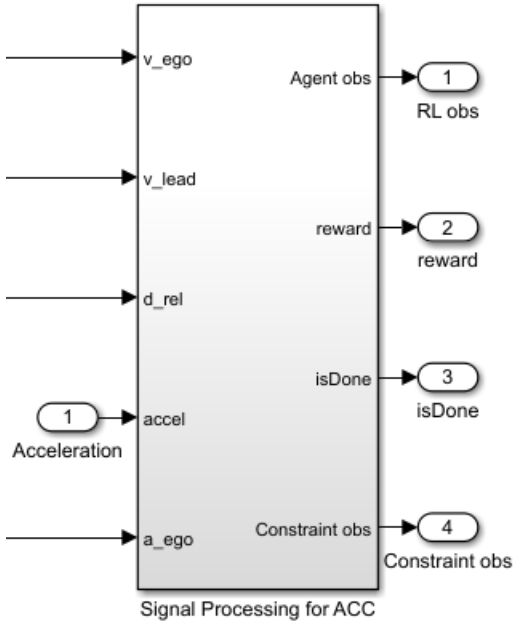

Figure 11. Signal Processing for ACC block

3.1.3 DDPG Agent with Constraint Enforcement

In the ACC application, the safety signals monitored are the ego car's velocity (v) and the relative distance (d) between the ego car and the lead car. The constraints for these signals are set as $10 \leq v \leq 30.5$ for the velocity and $d \geq 5$ for the distance. These constraints are determined by the following state variables in x: the ego car's actual acceleration, the ego car's velocity, the relative distance, and the lead car's velocity.

The action (u) represents the acceleration command for the ego car. The relationship between the safety signals and the action, along with the state variables, is described by the following equation:

$$\begin{bmatrix} v_{k+1} \\ d_{k+1} \end{bmatrix} = \begin{bmatrix} f_1(x_k) \\ f_2(x_k) \end{bmatrix} + \begin{bmatrix} g_1(x_k) \\ g_2(x_k) \end{bmatrix} u_k \tag{11}$$

The form's constraints are accepted by the Constraint Enforcement block $f_x + g_x u \leq c$. the coefficients of this constraint function are as follows:

$$f_x = \begin{bmatrix} -f_1(x_k) \\ -f_2(x_k) \\ f_1(x_k) \end{bmatrix}, g_x = \begin{bmatrix} -g_1(x_k) \\ -g_2(x_k) \\ g_1(x_k) \end{bmatrix}, c = \begin{bmatrix} -10 \\ -5 \\ 30.5 \end{bmatrix} \quad (12)$$

To learn the unknown functions f_i and g_i, training data was collected from the environment.

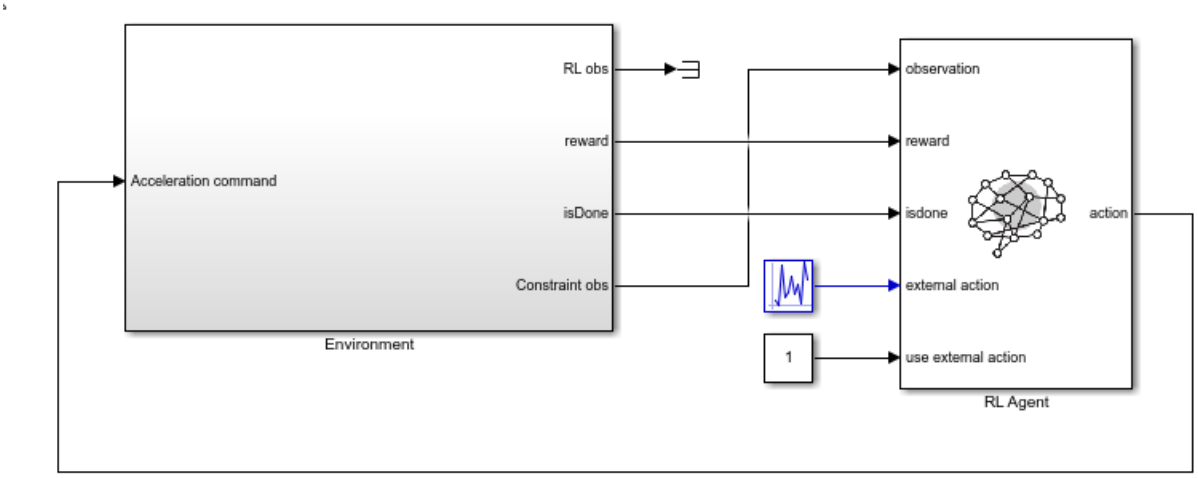

Figure 12. Learn Constraint Simulink Model

In this particular model configuration, the RL Agent block assumes a unique role by abstaining from the generation of actions. Instead, it operates in a manner where it facilitates the transmission of random external actions into the environment. This deliberate design choice serves a crucial purpose: to maintain consistency between the data collection phase and subsequent agent training sessions. By ensuring that the environment model, action and observation signal configurations, and model reset function employed during data collection align precisely with those utilized during agent training, the integrity of the training process is preserved.

The random external action injected into the environment adheres to a uniform distribution range spanning from -10 to 6 m/s. This range essentially dictates the behavior of the ego car within the simulated environment. Specifically, it signifies that the ego car is capable of exerting a maximum braking power of -10 m/s² or a maximum acceleration power of 6 m/s², depending on the specific action sampled from this distribution.

As for the training regimen, the environment furnishes the RL agent with four fundamental observations. These include the relative distance between vehicles, the velocities of both the leading and ego cars, and the acceleration of the ego car itself. Each of these observations plays a pivotal role in shaping the agent's understanding of the environment's dynamics. To delineate a continuous observation space for these values, we can establish distinct variables for each: the relative distance, the lead car velocity, the ego car velocity, and the ego car acceleration.

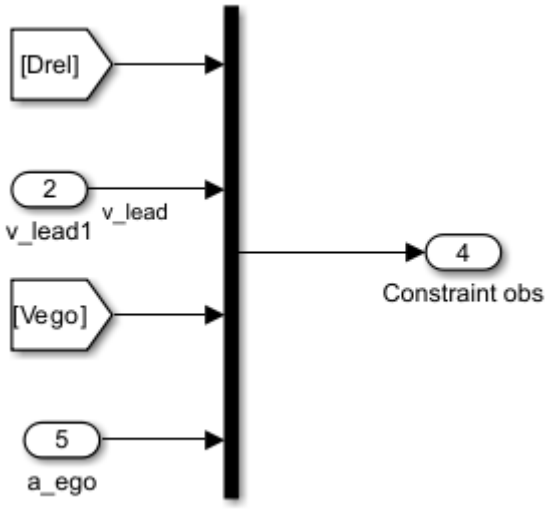

Figure 13. Constraint observation

By encompassing these four observations within a continuous space, the RL agent gains a comprehensive understanding of the environment's state. This holistic view enables the agent to make informed decisions and effectively learn optimal strategies during the training process.

With this model (figure 14), the acceleration command from the agent is constrained before applying it to the environment.

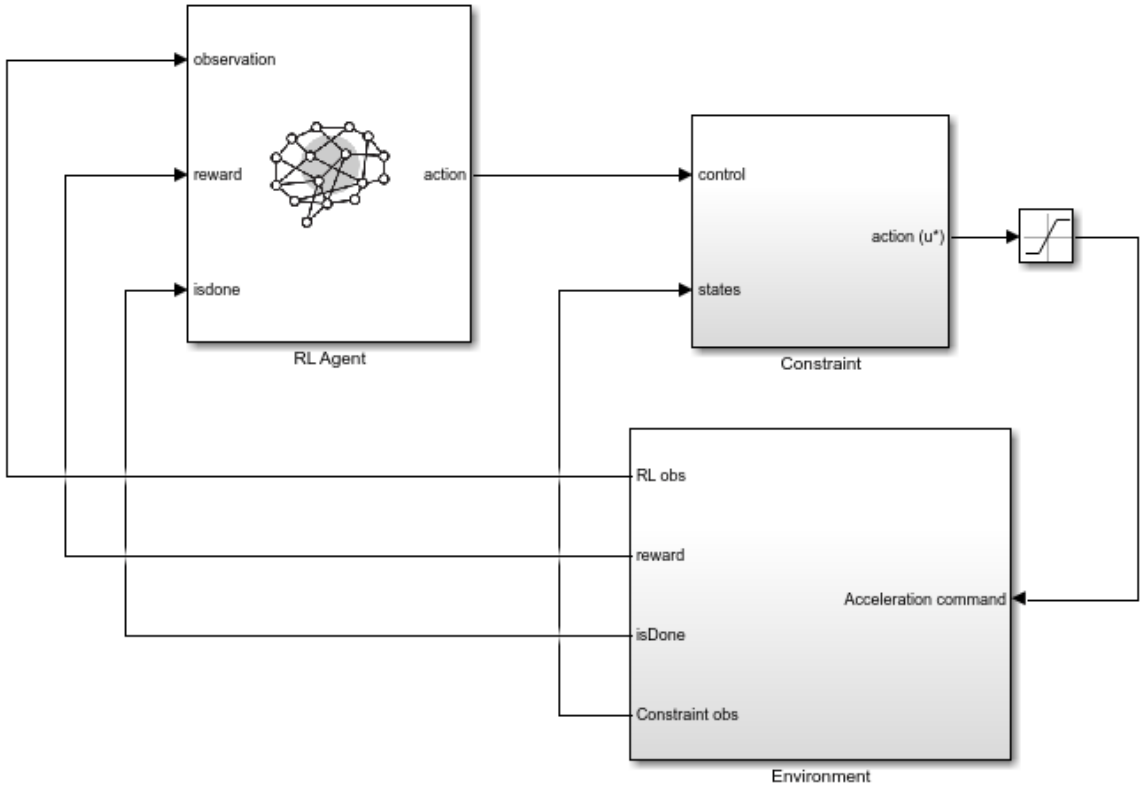

Figure 14.RL Model with Constraints

In the Constraint subsystem, the model generates the values of f_i and g_i from the linear constraint relations. The model sends these values along with the constraint bounds to the Constraint Enforcement block.

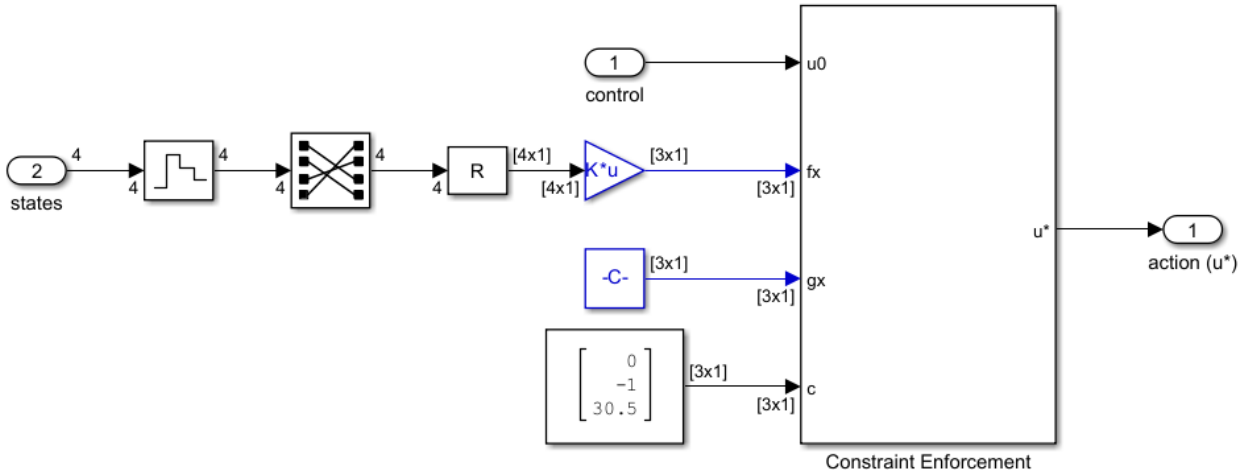

Figure 15. Constraint subsystem

In the RL environment using this model, for training, the environment produces three observations: the integral of the velocity error, the velocity error, and the ego-car velocity.

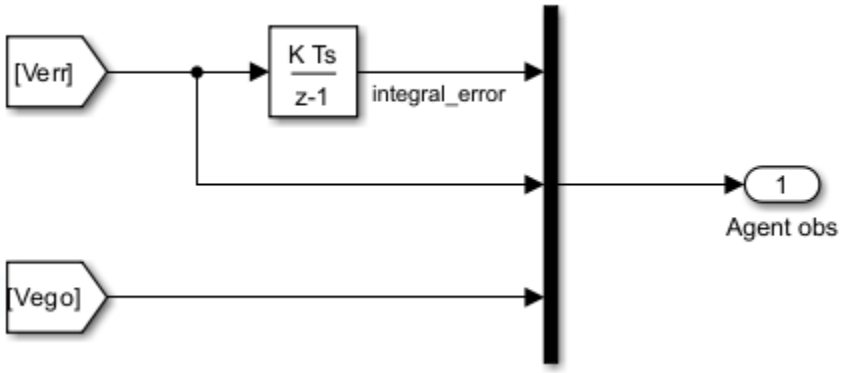

Figure 16. Agent observations

The isDone signal, serves as a trigger indicating when certain critical constraints are breached. These constraints primarily revolve around two key scenarios: firstly, if the ego car experiences negative velocity, effectively moving backward; and secondly, if the relative distance between the ego car and the lead car falls below zero, indicating a collision between them.

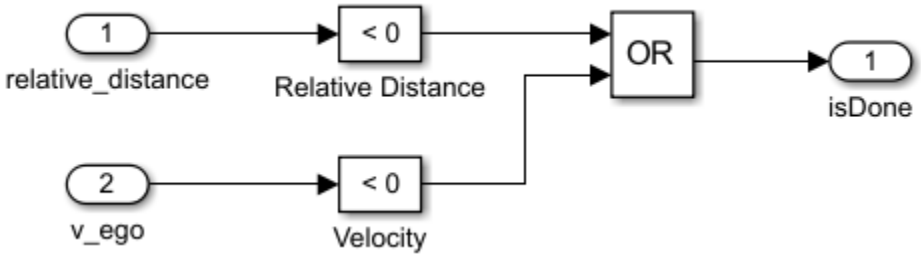

Figure 17. isDone subsystem

The RL Agent block relies on this isDone signal as a vital cue during training. It serves as a mechanism to prematurely terminate training episodes when these critical constraints are violated. By halting the training process early in such scenarios, the RL Agent can avoid potentially detrimental situations, allowing for more efficient learning and preventing the reinforcement learning algorithm from continuing in states where undesirable outcomes have already occurred. This termination signal acts as a safeguard, ensuring that the agent doesn't persist in exploring actions that lead to violations of safety or performance criteria, ultimately contributing to more effective and safer training.

3.1.4 Implementation of Spacing Policy

The reference velocity for the ego car, denoted as V_{ref}, is determined by specific conditions related to the relative distance between the ego car and the lead car. This reference velocity guides how the ego car adjusts its speed in relation to the lead car, ensuring safe and efficient navigation.

If the relative distance between the ego car and the lead car falls below a predefined safe distance threshold, the ego car's velocity is adjusted to track the minimum of two factors: the lead car's velocity and a driver-set velocity. This adjustment ensures that the ego car maintains a safe distance from the lead car, thereby reducing the risk of collisions.

Conversely, if the relative distance exceeds the safe distance threshold, the ego car's velocity is set to track the driver-set velocity independently. In this scenario, where the distance to the lead car is considered safe, the ego car can operate at its intended speed without constraint by the lead car's velocity. Figure 18 represents the spacing policy in the Simulink model.

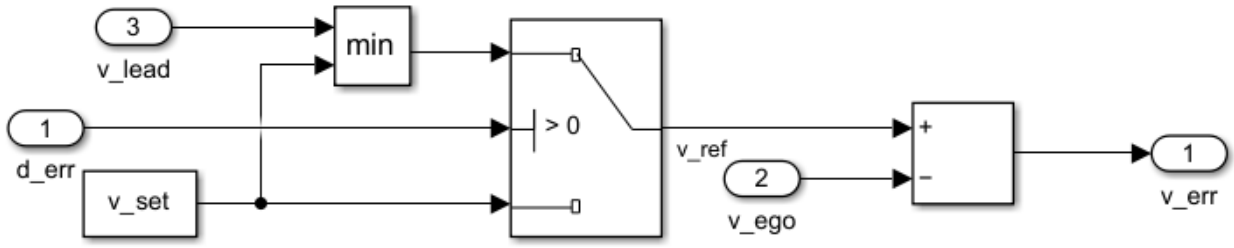

Figure 18. Spacing policy subsystem

In this specific setup, the safe distance or desired distance threshold is calculated as a linear function of the ego car's longitudinal velocity, expressed as:

$$d_{des} = th \times v_h + d_{min} \tag{13}$$

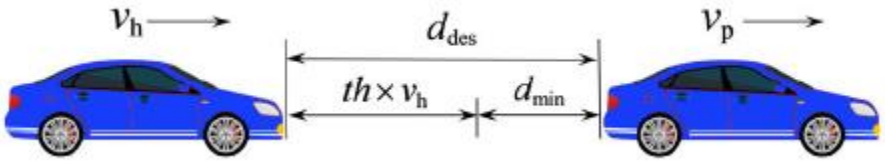

Figure 19. Time headway-based spacing policy [25]

Here, th represents a time gap factor, d_{min} denotes a default distance offset, and v_h signifies the lead car's current longitudinal velocity. This computation ensures that the safe distance dynamically adjusts based on the ego car's speed, providing a more nuanced approach to maintaining safe spacing between vehicles.

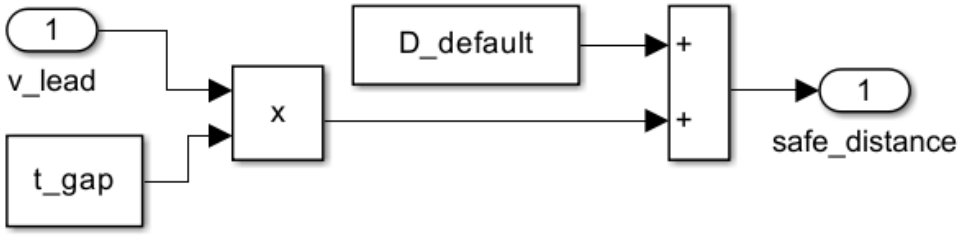

Figure 20. Safe distance subsystem

In summary, the reference velocity for the ego car is intricately tied to the relative distance between it and the lead car. By dynamically adjusting its velocity based on predefined conditions and the safe distance threshold, the ego car can navigate the environment safely and efficiently, mitigating the risk of collisions while maintaining optimal speeds.

3.1.5 Formulation of the Reward Function

A strategically designed reward function was created to guide the agent towards human-like driving behavior, particularly in terms of recognizing and mitigating collision risks. This reward system provides partial supervision, enabling the agent to develop an understanding of how to perceive potential dangers and take appropriate actions to avoid them.

To promote smooth driving, the reward function incorporates several key components. Firstly, the error in acceleration is squared. This squared acceleration error penalizes rapid changes in acceleration, encouraging the agent to opt for smoother, more gradual adjustments in speed. Secondly, the velocity error is also squared and included in the reward calculation. This term aims to improve the agent's ability to follow the desired velocity accurately, ensuring that the ego car maintains a speed that is consistent with either the driver-set velocity or the conditions dictated by the relative distance to the lead car.

Additionally, the reward function features a term for the distance error. This component is crucial as it incentivizes the agent to maintain a safe distance from the lead car. By penalizing deviations from the safe distance, the agent is encouraged to avoid getting too close to the lead car, thereby reducing the risk of collisions.

The equation below is the reward function.

$$r(s,a) = -w_1 r_a - w_2 r_v - w_3 r_d + M \tag{14}$$

- r_a is the punishment for any nonzero acceleration and is equal to a_t^2.

- r_v is the squared difference between reference velocity and ego velocity: $(v_{ref} - v_{ego})^2$

- r_d is the punishment for any deviation from the desired distance: $\left|\frac{d_{rel}}{d_{des}} - 1\right|$

- M is the positive reward for desired action and is calculated as:

$$M = \begin{cases} 1 & if \quad 0 < r_d \leq 0.1 \\ 1 & if \quad 0 < r_v \leq 0.1 \end{cases}$$

- w_1, w_2 and w_3 are weights that determine the tradeoff between smoothness and safety.

Figure 21 represents the Reward function block utilized within a Simulink model.

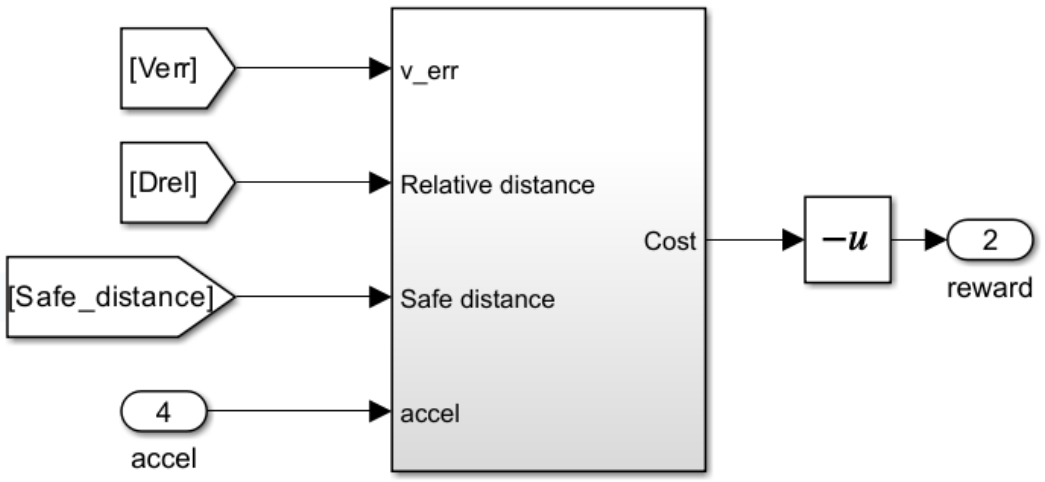

Figure 21. Reward function block

Figure 22 represents the Reward function subsystem utilized within a Simulink model with weights values of:

$$\begin{cases} w_1 = 1 \\ w_2 = 0.1 \\ w_3 = 1 \end{cases}$$

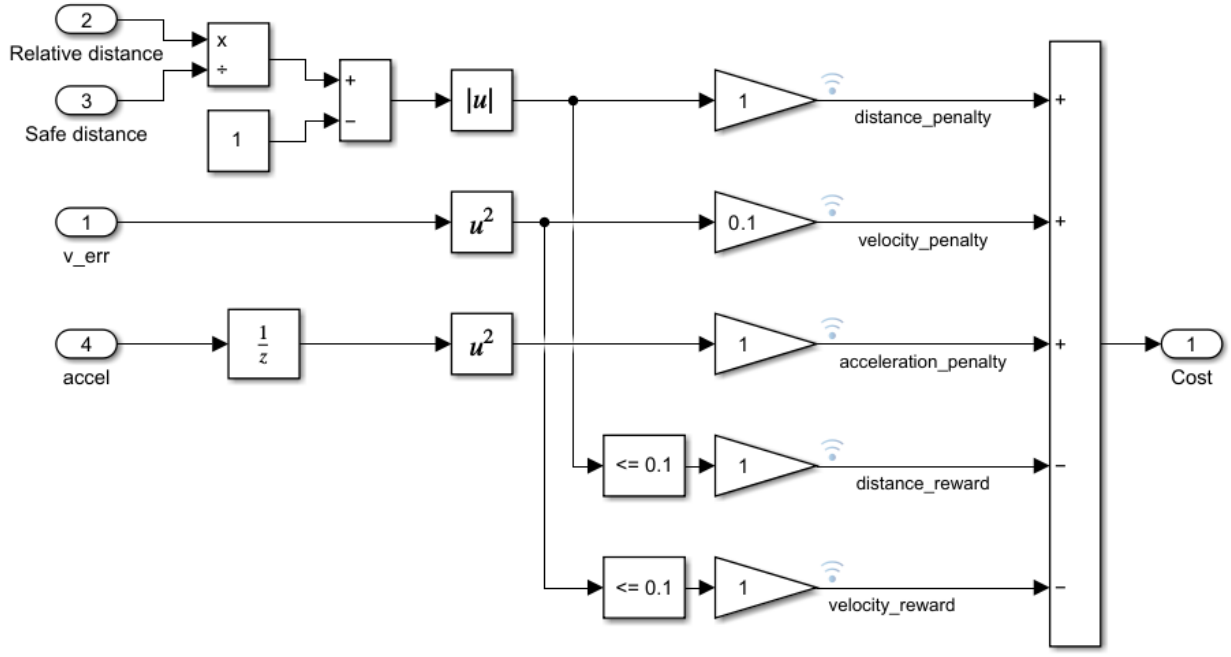

Figure 22. Reward function subsystem

Overall, the reward function is designed to balance multiple aspects of driving behavior. By squaring the errors in both acceleration and velocity, it promotes smoother driving and better adherence to target speeds. The inclusion of the distance error term further ensures that the agent consistently maintains a safe following distance, fostering a driving style that is both safe and efficient. This comprehensive approach helps the agent to develop driving skills that closely resemble those of a cautious and skilled human driver.

Chapter 4:

4.1 Training the DDPG Agent

The training process for the agent was designed to simulate a realistic driving scenario, focusing on interactions between a lead car and an ego car. The initial velocities for both vehicles were randomly set within a range of 1 to 25 m/s, providing a diverse set of starting conditions for the agent to learn from. To ensure safety and adherence to realistic driving limits, the ego car's set velocity was capped at 30 m/s, preventing it from exceeding this speed during training.

Several critical parameters were used to model the driving environment and ensure that the agent's training reflected real-world conditions. These parameters include:

- Default Spacing (d_{min}): Set to 5 meter, this value represents the minimum distance that should be maintained between the ego car and the lead car, serving as a basic safety measure.

- Time Gap (th): Set to 1.4 seconds, this parameter dictates the safe following distance based on the ego car's speed, ensuring that the ego car has adequate time to react to the lead car's movements.

- Driver-Set Velocity (v_{set}): Set at 30 m/s, this value serves as the maximum speed limit for the ego car, ensuring it does not drive too fast and remains within safe operating conditions.

- Minimum and Maximum Acceleration (a_{min} and a_{max}): These values, set at -3 m/s² and 2 m/s² respectively, define the range of acceleration for the ego car, promoting smooth and comfortable driving by preventing sudden, jerky movements.

- Sample Time (T_s): Set at 0.1 seconds, this parameter determines the frequency at which the system updates, providing a balance between computational efficiency and the granularity of the agent's actions.

To add a layer of realism to the training environment, the acceleration of the lead car was simulated using a sine wave with an amplitude of 0.6 and a frequency of 0.2 rad/s. This introduced variability in the lead car's speed, challenging the ego car to adapt to changing conditions and improving the robustness of the agent's learning process.

The hyperparameters for the DDPG (Deep Deterministic Policy Gradient) agent, detailed in Table 1, were meticulously chosen to optimize the training process. The agent underwent training for a maximum of 5000 episodes, with the objective of reaching an episode reward of 500. Achieving this reward threshold indicated that the agent had successfully learned the desired driving behaviors, such as maintaining safe distances, following the set velocity, and reacting smoothly to the lead car's movements.

Hyperparameter	**Value**
Time step T_s	0.1
Discount Factor γ	0.99
Experience Buffer Length	2×10^5
Mini-Batch Size	128
Maximum Training Episode	5000
Target Update Frequency	10
Target Smooth Factor τ	0.005
Critic Learning Rate α_c	10^{-3}
Actor Learning Rate α_a	10^{-4}
Exploration Noise ε	0.1
Exploration Model	Ornstein-Uhlenbeck
Optimizer	Adam

Table 1. DDPG agent hyperparameters

Figure 16 presents the training progression, illustrating how the agent's performance improved over time as it learned from its experiences. The graph shows a steady increase in the episode rewards, indicating the agent's growing proficiency in navigating the simulated driving environment.

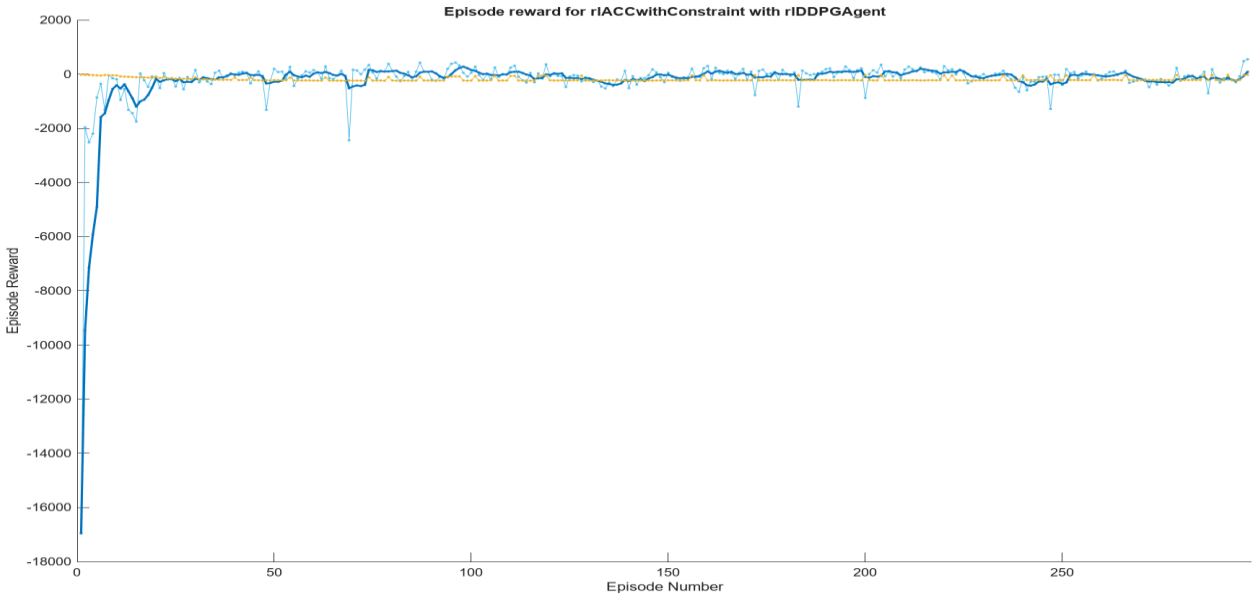

Figure 23. DDPG agent training progression

Overall, the structured training environment and well-defined parameters were crucial in developing an agent capable of human-like driving behavior. By ensuring the training process reflected real-world conditions, the agent was able to learn effective strategies for safe and efficient driving, preparing it for deployment in more complex and dynamic scenarios.

4.2 Validating the DDPG Agent

4.2.1 WLTP Cycle Evaluation

Following the completion of the training phase, the agent's performance was rigorously validated using the Worldwide Harmonized Light Vehicles Test Procedure (WLTP) cycle. The WLTP cycle provides a comprehensive testing framework that simulates a wide range of driving scenarios, including different speeds, acceleration patterns, and stop-and-go conditions typically encountered in everyday driving. By subjecting the agent to this standardized test cycle, we aimed to evaluate its ability to maintain safe distances, adhere to speed limits, and execute smooth acceleration and deceleration maneuvers in varied and unpredictable traffic conditions.

During the validation process, the agent's actions were closely monitored to determine how effectively it could replicate human-like driving behavior. Key performance metrics, such as speed adherence, following distance, and smoothness of driving, were analyzed to gauge the agent's proficiency.

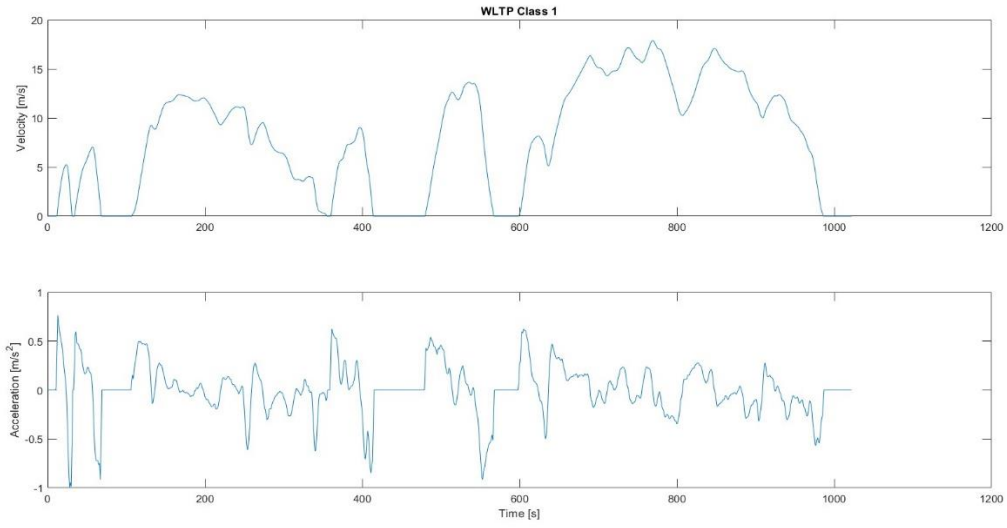

Figure 24. WLTP class 1 cycle

The simulation was conducted with initial velocities set at 25 m/s for the lead car and 20 m/s for the ego car.

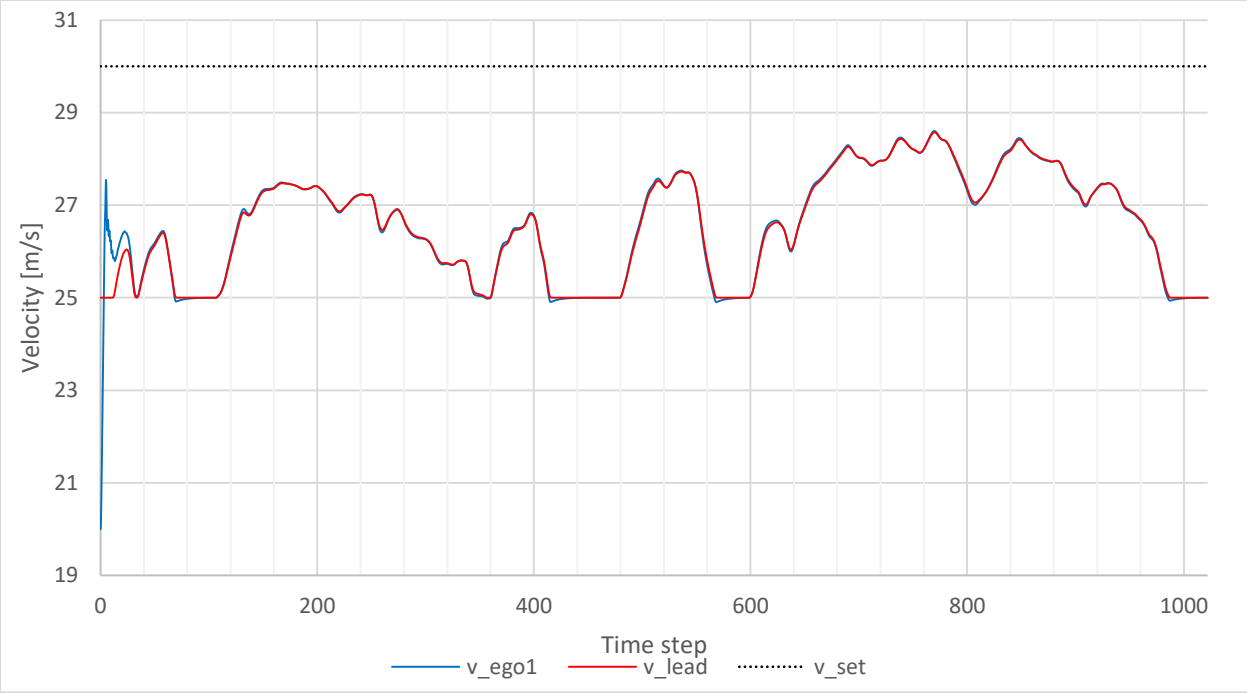

Figure 25. Leading and ego velocity on WLTP

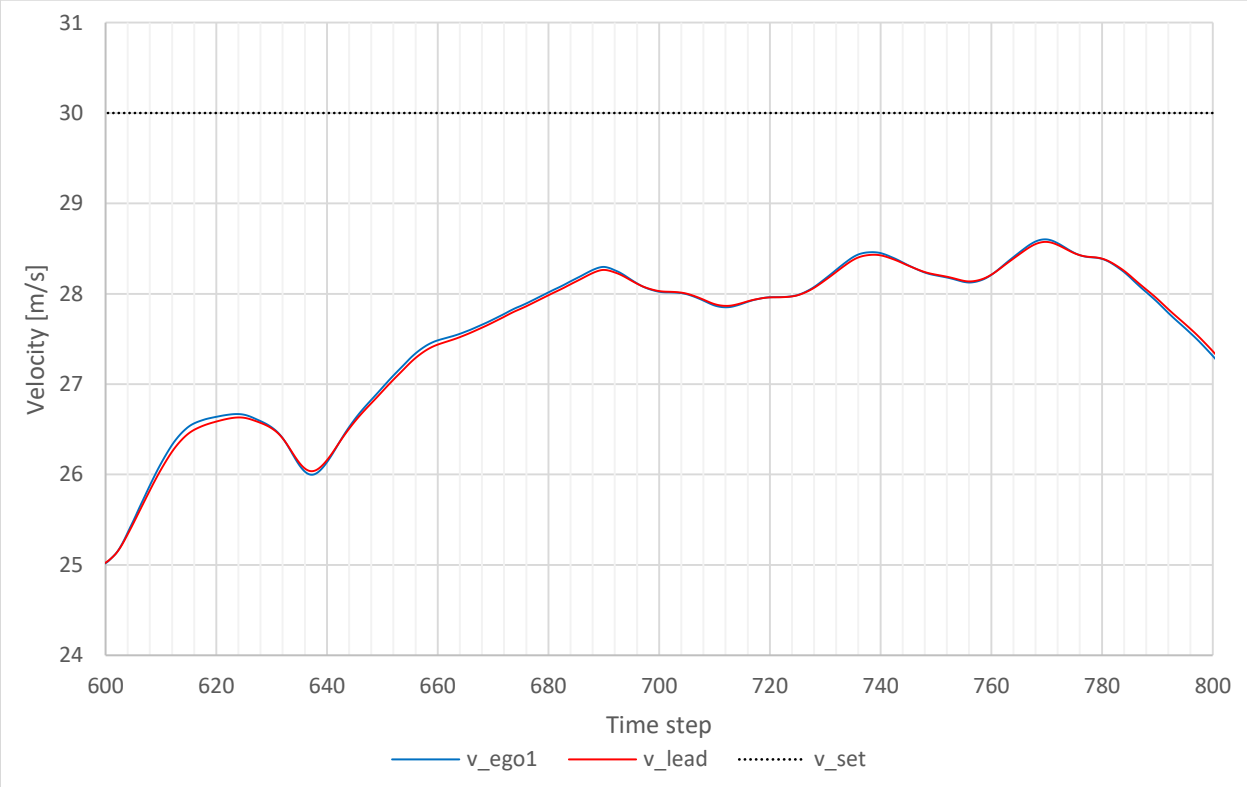

Figure 26. Zoomed-In view of lead and ego velocities during WLTP

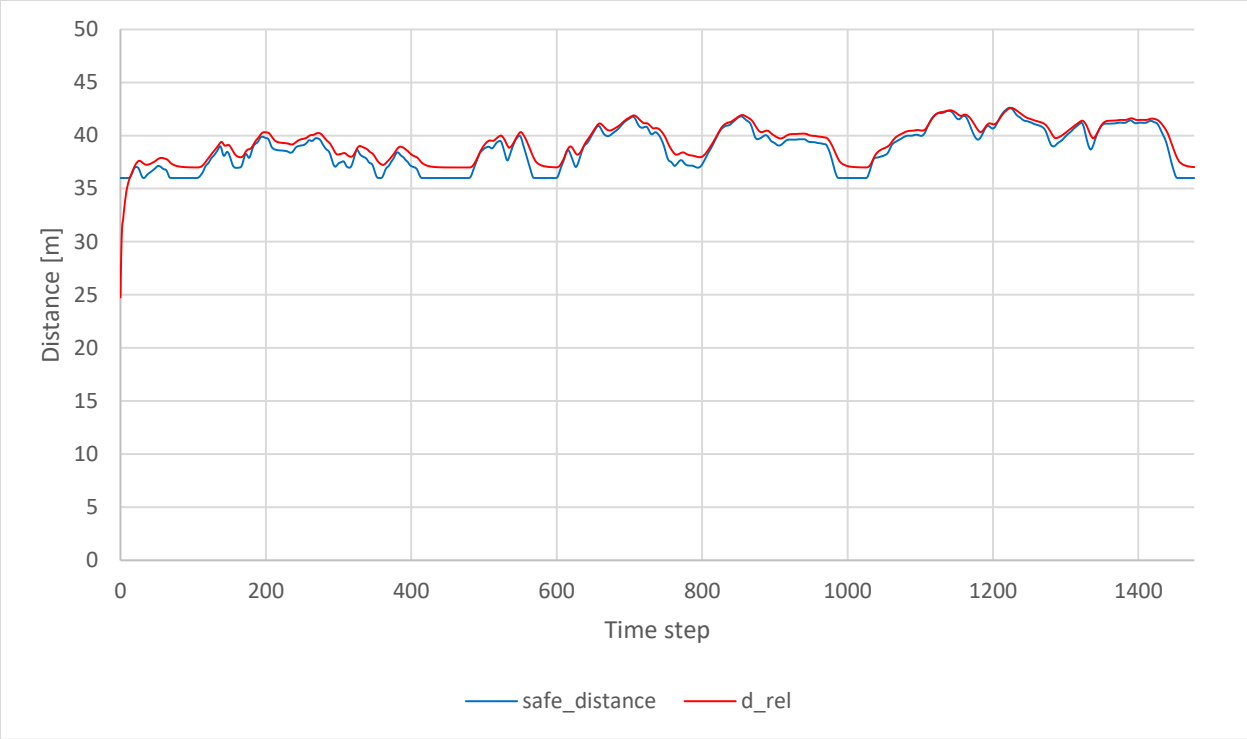

Figure 27. Rleative and safe distance on WLTP

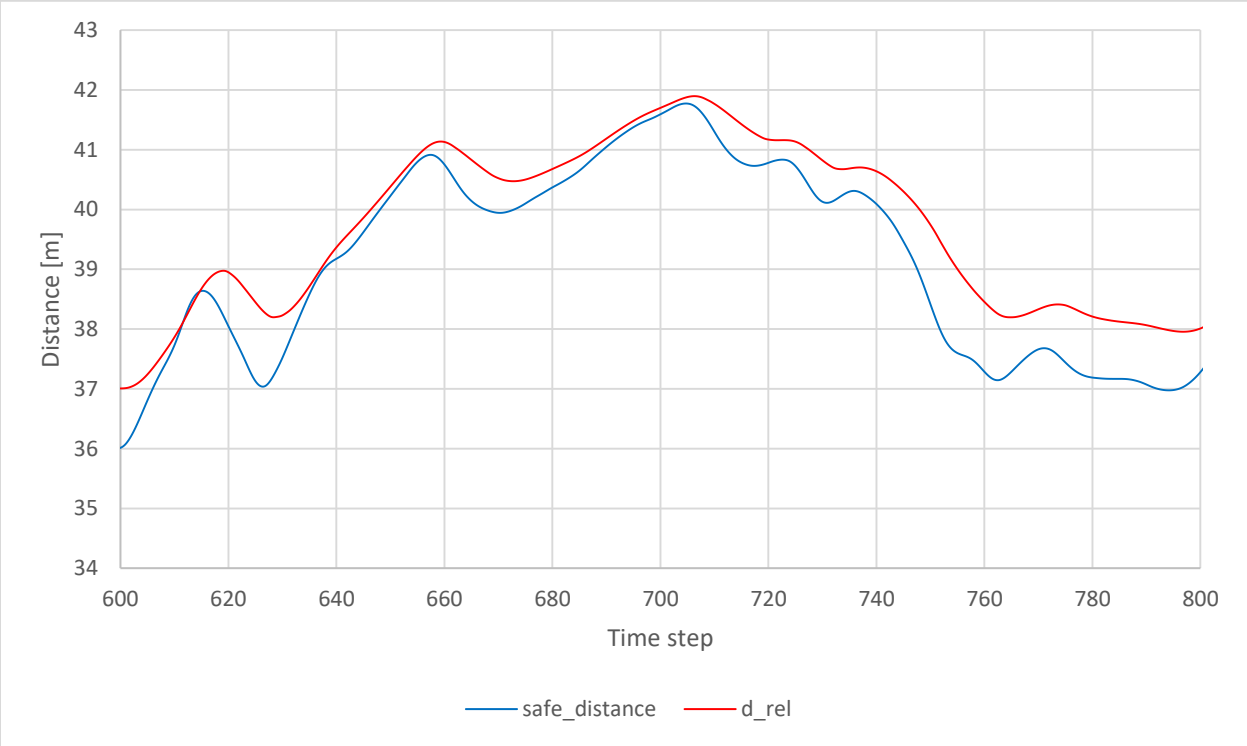

Figure 28. Zoomed-In view of relative and safe distance during WLTP

Through this comprehensive validation process using the WLTP cycle, we were able to identify any potential weaknesses in the agent's behavior and make necessary adjustments. This ensured that the agent not only performed well in the controlled training environment but also demonstrated reliable and safe driving behavior in real-world scenarios. The insights gained from this validation step were critical in refining the agent's algorithms and enhancing its overall performance, paving the way for its deployment in practical applications.

4.2.2 WLTP Cycle Evaluation with Delay

To accurately reflect real-world conditions where there are inherent delays in the signals sent to vehicles, different delay values were incorporated into the simulation. This approach helps in understanding how such delays impact vehicle performance and interaction. In the subsequent figures, the results of the simulation with delay values of 0, 0.1, and 0.2 seconds are presented. These delays were specifically added to the acceleration signals transmitted between the cars, mimicking the latency that can occur in real driving scenarios. By varying the delay values, we can analyze how the ego car's response to the lead car's movements is affected, providing valuable insights into the robustness and reliability of the trained agent in handling real-world communication delays.

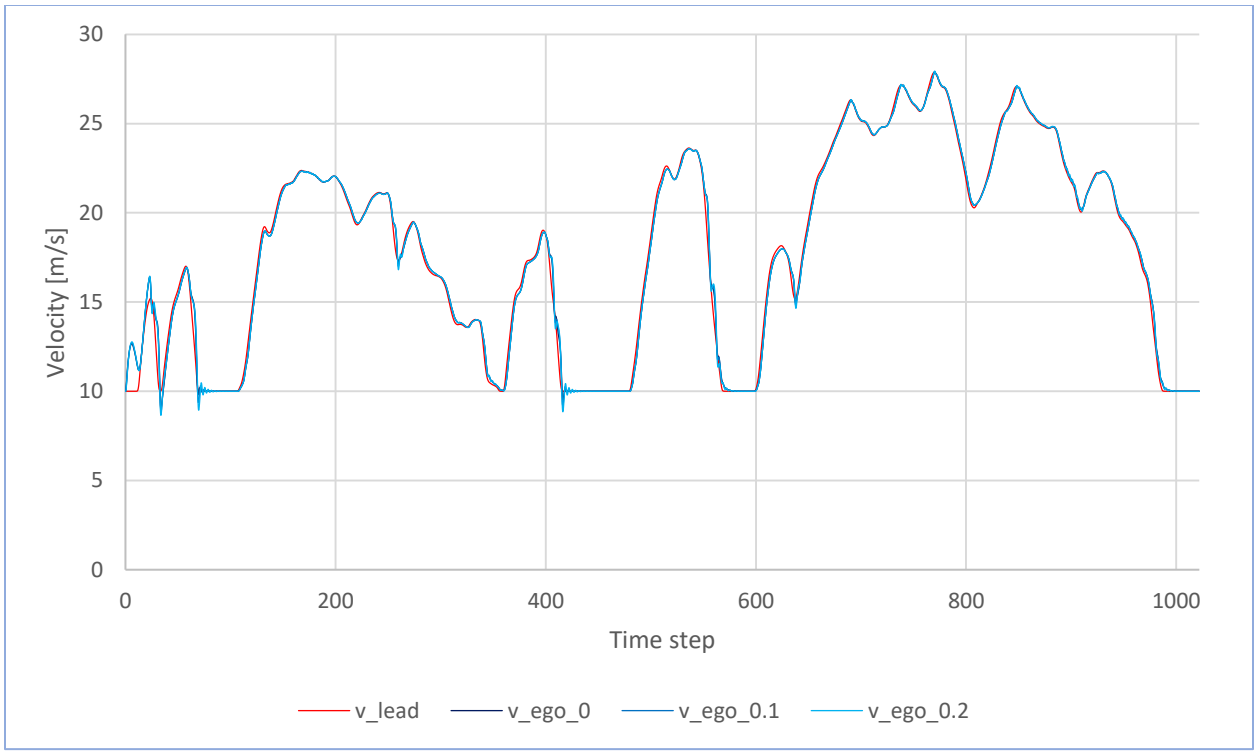

Figure 29. Lead and ego signal delay simulation

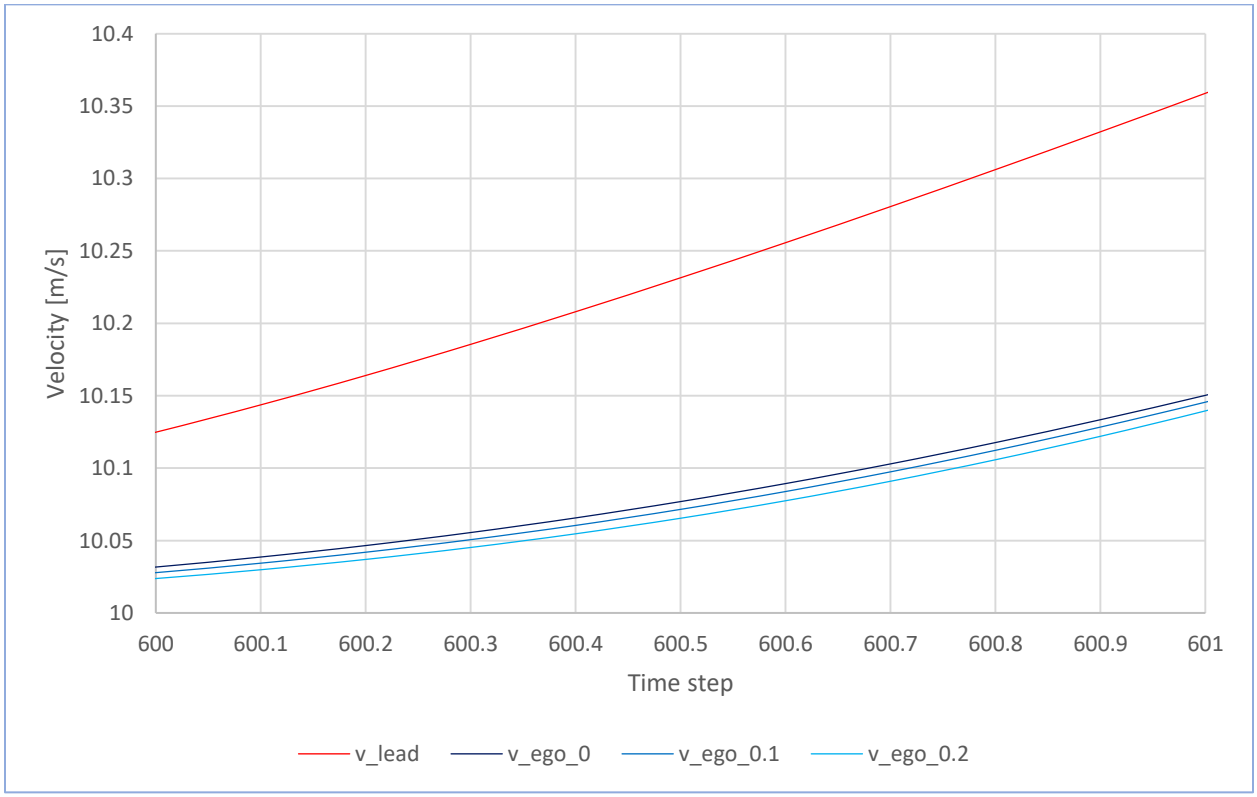

Figure 30. Zoom on lead and ego signal delay simulation

The Root Mean Square Error (RMSE) for the spacing error, which quantifies the difference between the relative distance and the safe distance, was calculated. The analysis revealed that as more delay is introduced into the system, the RMSE value increases. This indicates that larger delays in signal transmission negatively affect the ability to maintain the desired safe distance between vehicles, leading to greater deviations from the optimal spacing.

Delay	RMSE
0	5.53571
0.1	5.53599
0.2	5.53873

Table 2. RMSE of spacing error with different delay values

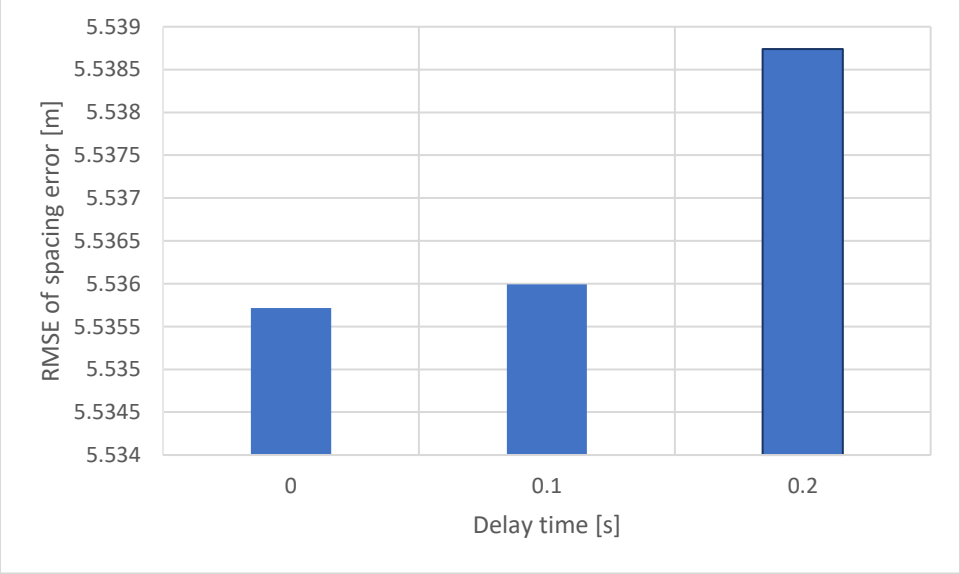

Figure 31. RMSE of spacing error with different delay values

4.3 Simulation of Multi-Vehicle Platoon

To thoroughly evaluate the performance of the cooperative cruise control model, simulations were conducted involving multiple ego cars, rather than a single ego car. This setup allows for a more comprehensive assessment of the model's effectiveness in a more complex and realistic driving environment.

By including multiple ego cars in the simulations, the interactions between different vehicles can be observed and analyzed. This approach helps in understanding how well the cooperative cruise control model manages spacing, speed adjustments, and overall traffic flow when multiple cars are involved. It also provides insights into the model's scalability and robustness in handling scenarios with several vehicles operating under the same control strategy.

The results from these simulations are crucial for determining the model's capability to maintain safe distances between all vehicles, ensure smooth acceleration and deceleration, and optimize overall traffic efficiency. This evaluation is a critical step in validating the practical applicability of the cooperative cruise control model in real-world driving conditions, where multiple vehicles must operate in harmony to ensure safety and efficiency.

4.3.1 Three-Vehicle Platoon

In this phase, the cooperative cruise control model was analyzed by introducing one lead car and two ego cars into the simulation. This setup was chosen to evaluate the model's performance in a more dynamic and complex environment. To ensure the robustness and adaptability of the model, various initial velocities for both the lead car and the ego cars were tested.

Testing with different initial velocities allows us to examine how well the model handles a range of starting conditions. It helps in verifying that the model can efficiently synchronize the movements of the lead and ego cars, regardless of their initial speeds. This is crucial for real-world applications, where vehicles often start from different speeds and need to adjust accordingly to maintain safe distances and optimal flow.

By analyzing the model under these conditions, we can assess its ability to manage the relative distances between multiple vehicles, ensure smooth acceleration and deceleration patterns, and adapt to varying traffic scenarios. The insights gained from these tests are essential for confirming that the cooperative cruise control model is effective in diverse driving situations, enhancing its reliability and practical usability in real-world deployments.

In the first scenario, the lead car begins with an initial velocity of 25 m/s, while the two ego cars start at 20 m/s and 15 m/s, respectively. The initial relative distance between each car is set to 50 meters.

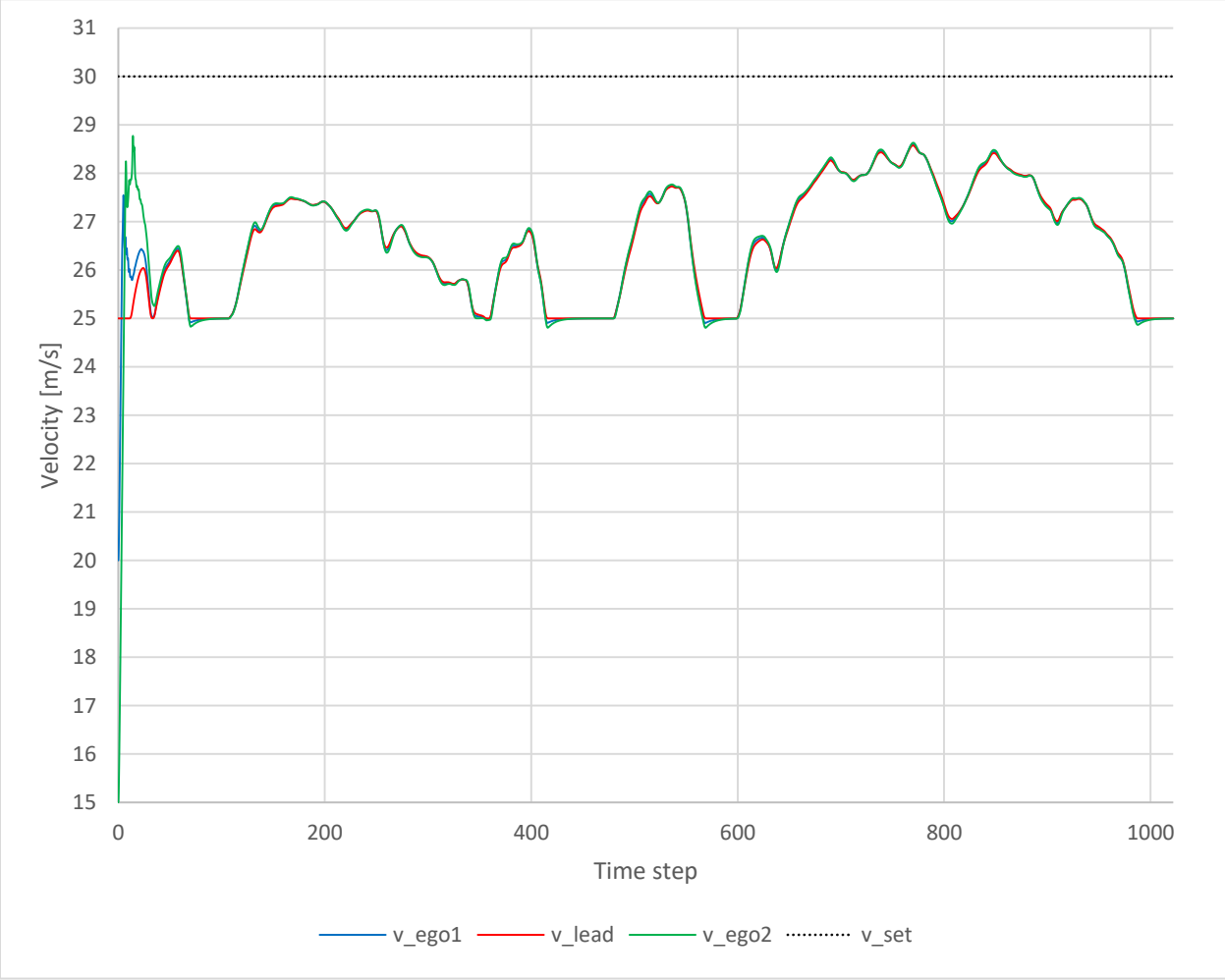

Figure 32. Three-Vehicle Platoon velocity control first scenario

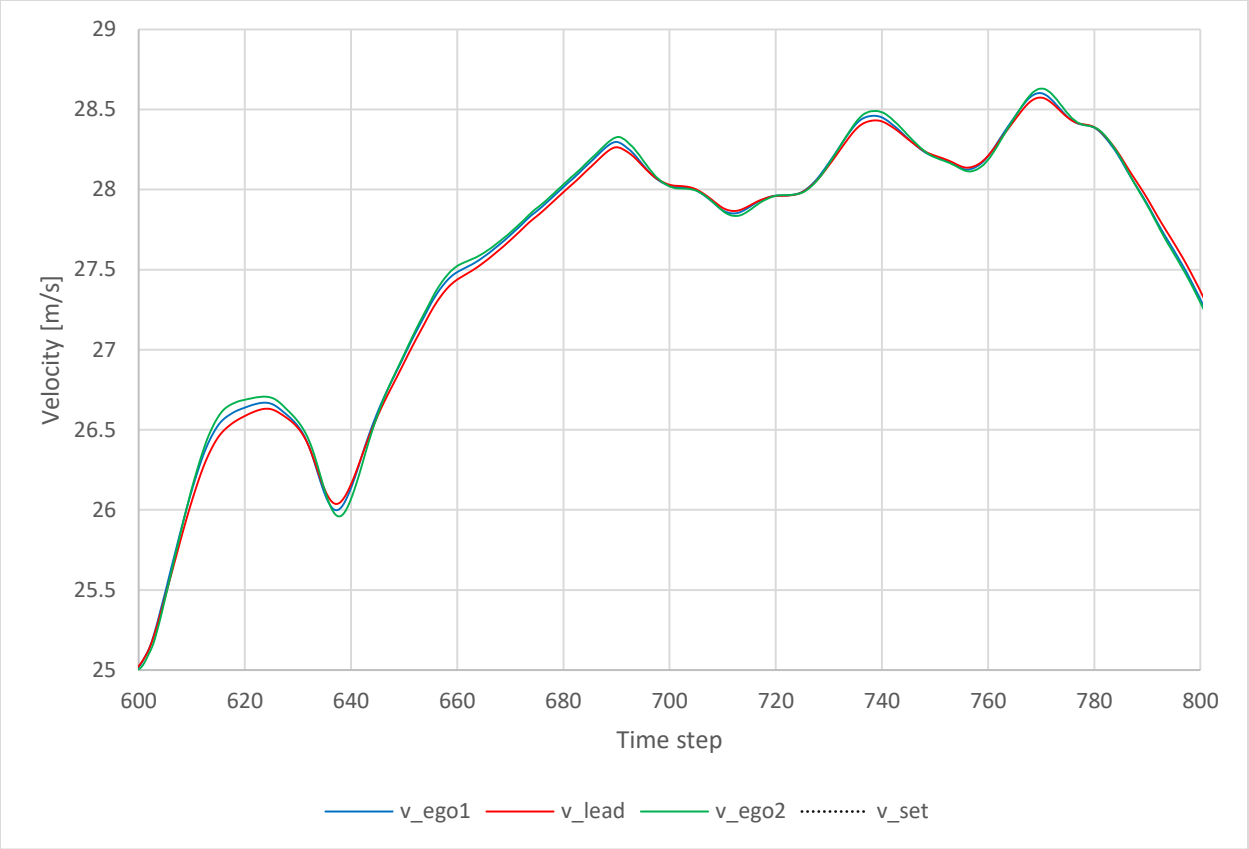

Figure 33. Zoom on Three-Vehicle Platoon velocity control first scenario

In the second scenario, contrasting and reduced initial velocity values have been selected in comparison to the initial scenario.

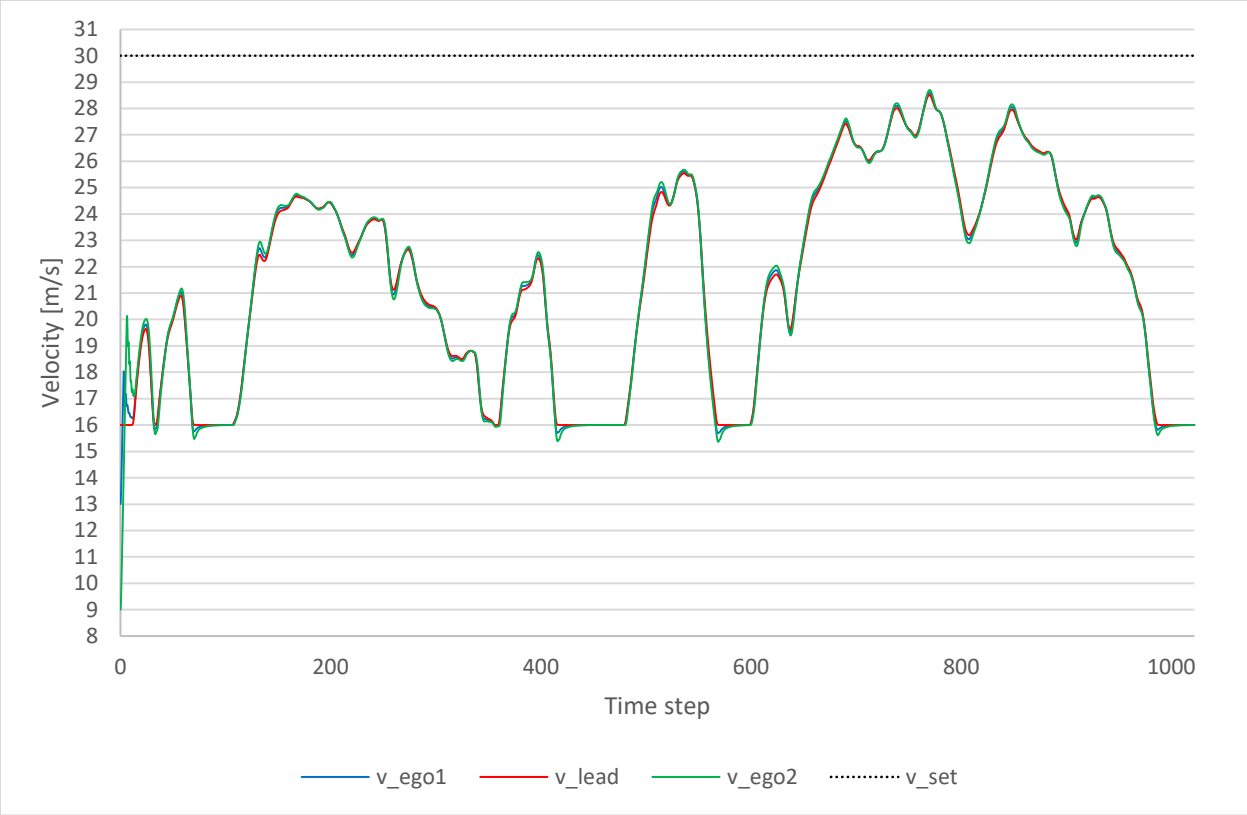

Figure 34. Three-Vehicle Platoon velocity control second scenario

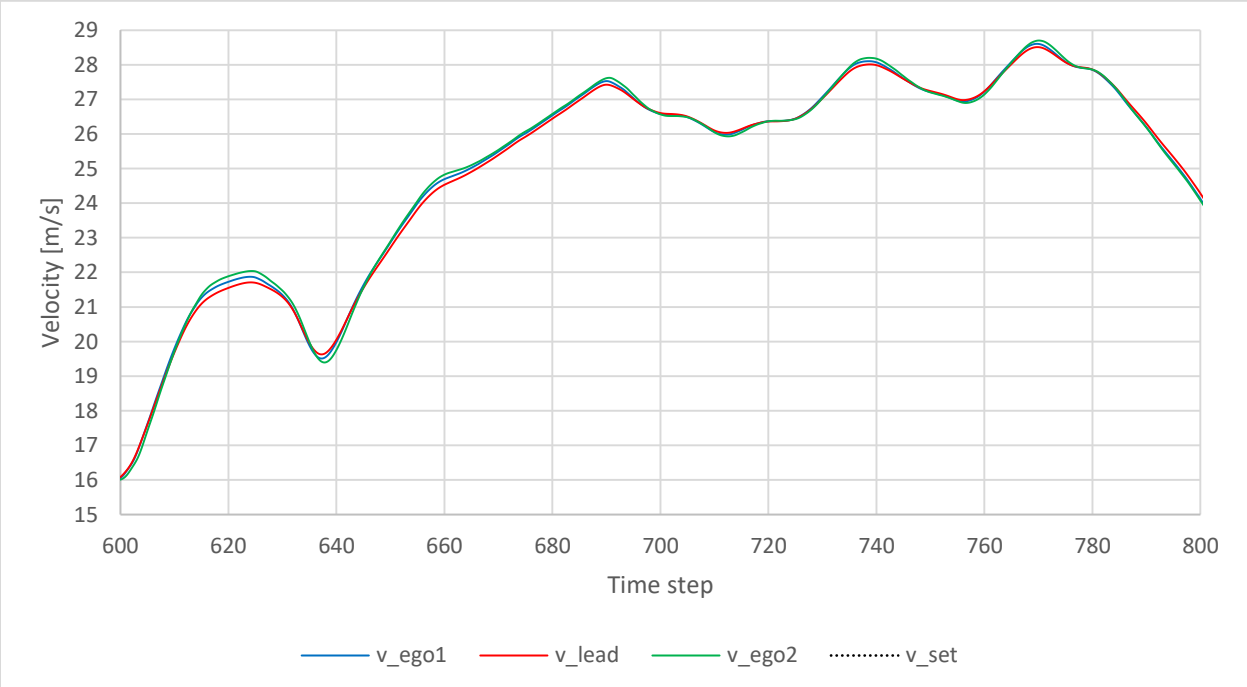

Figure 35. Zoom on Three-Vehicle Platoon velocity control second scenario

In the third scenario, the lead car's initial velocity (10 m/s) is lower than that of the ego cars (15 and 20 m/s). This simulation is conducted to ensure that the model remains effective even when the ego cars have higher initial velocities than the lead car. By testing this scenario, we aim to verify that the model's performance is not biased towards situations where the lead car maintains a higher speed. This helps in assessing the model's ability to handle diverse traffic conditions and confirms its reliability in scenarios where the ego cars may surpass the lead car's velocity.

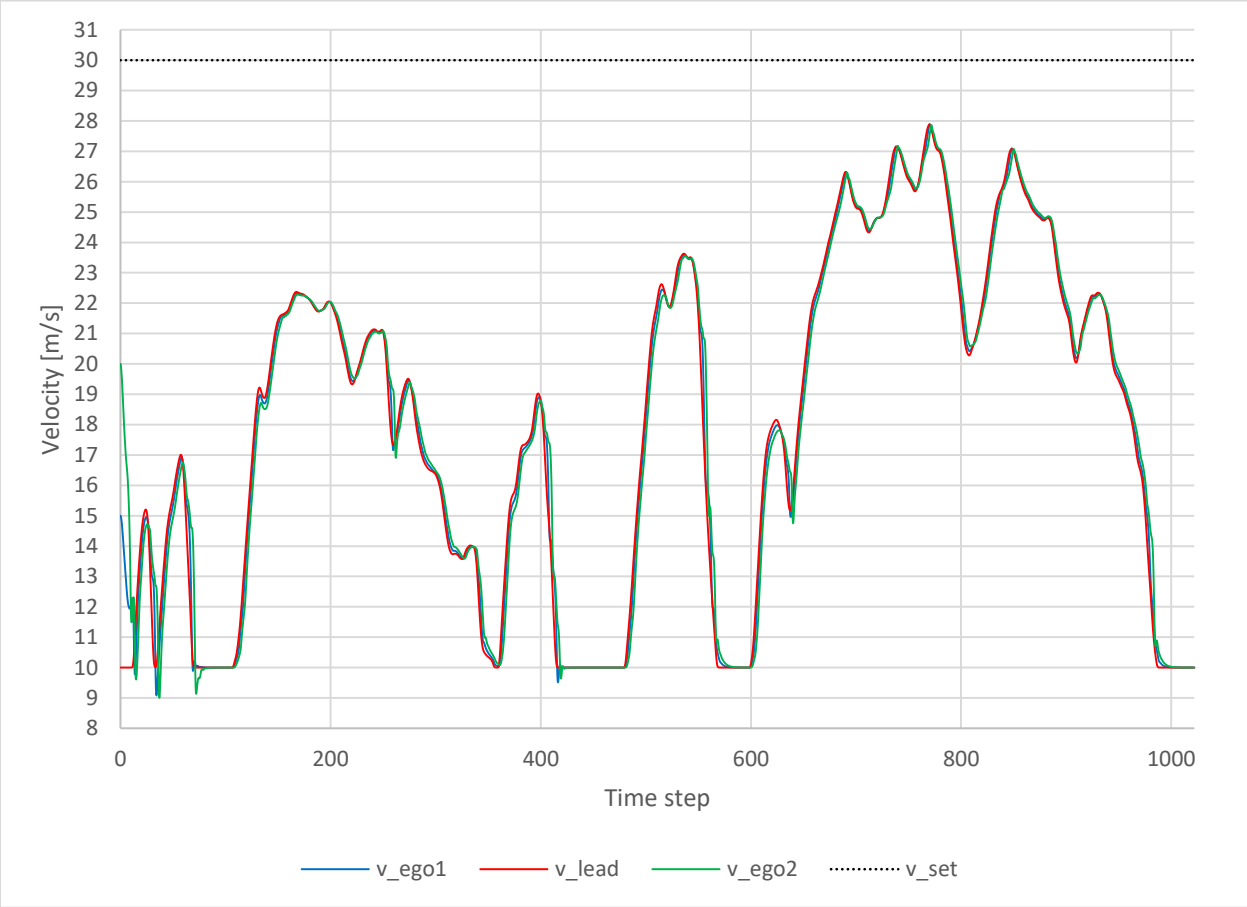

Figure 36. Three-Vehicle Platoon velocity control third scenario

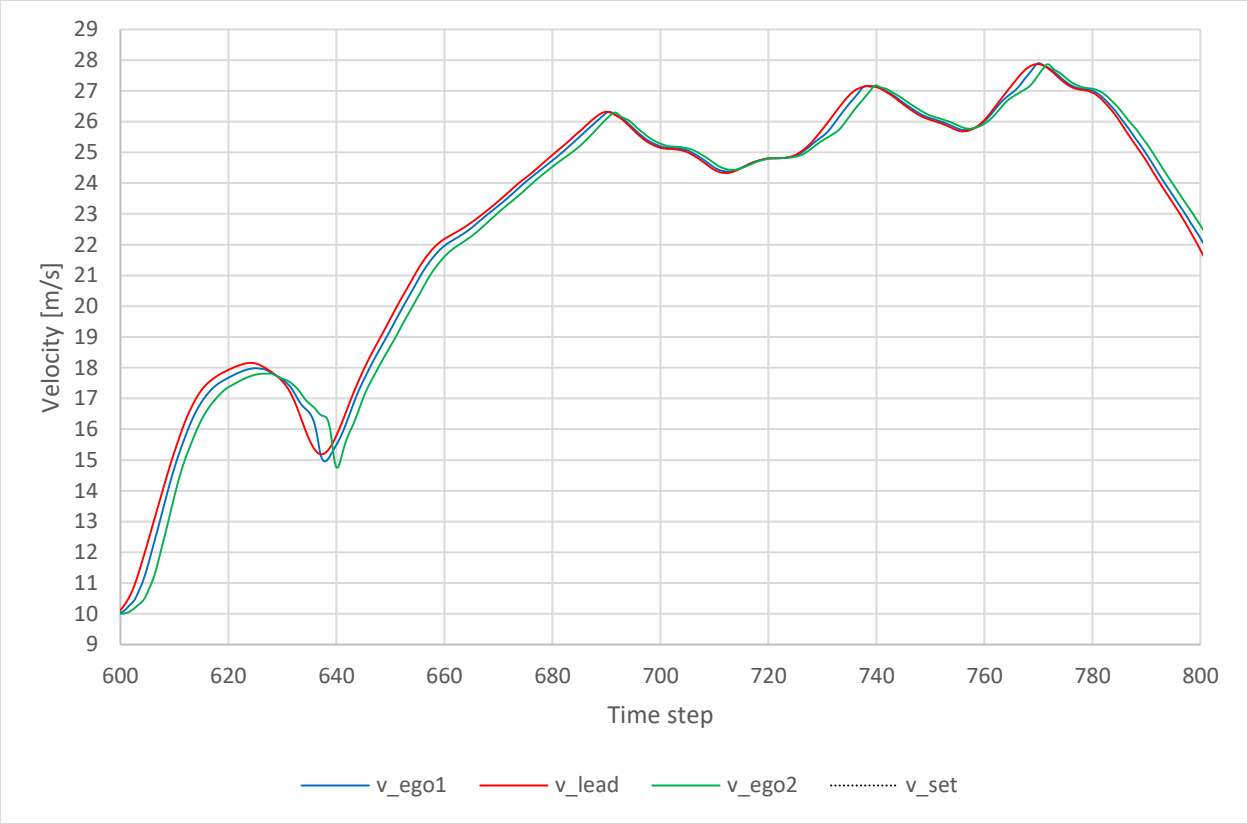

Figure 37. Zoom on velocity control Three-Vehicle Platoon velocity control third scenario

In the fourth and final scenario, all vehicles—the lead car and the two ego cars—start with identical velocities. Additionally, the initial relative distances between each pair of vehicles are set to 50 meters. This scenario serves to evaluate the model's performance in a uniform velocity setup, where all cars begin at the same speed. By maintaining equal velocities, we can observe how the model manages spacing and synchronization when no vehicle has a speed advantage. This test is crucial for verifying that the model can maintain consistent and safe distances, ensuring smooth operation even when starting conditions are completely uniform.

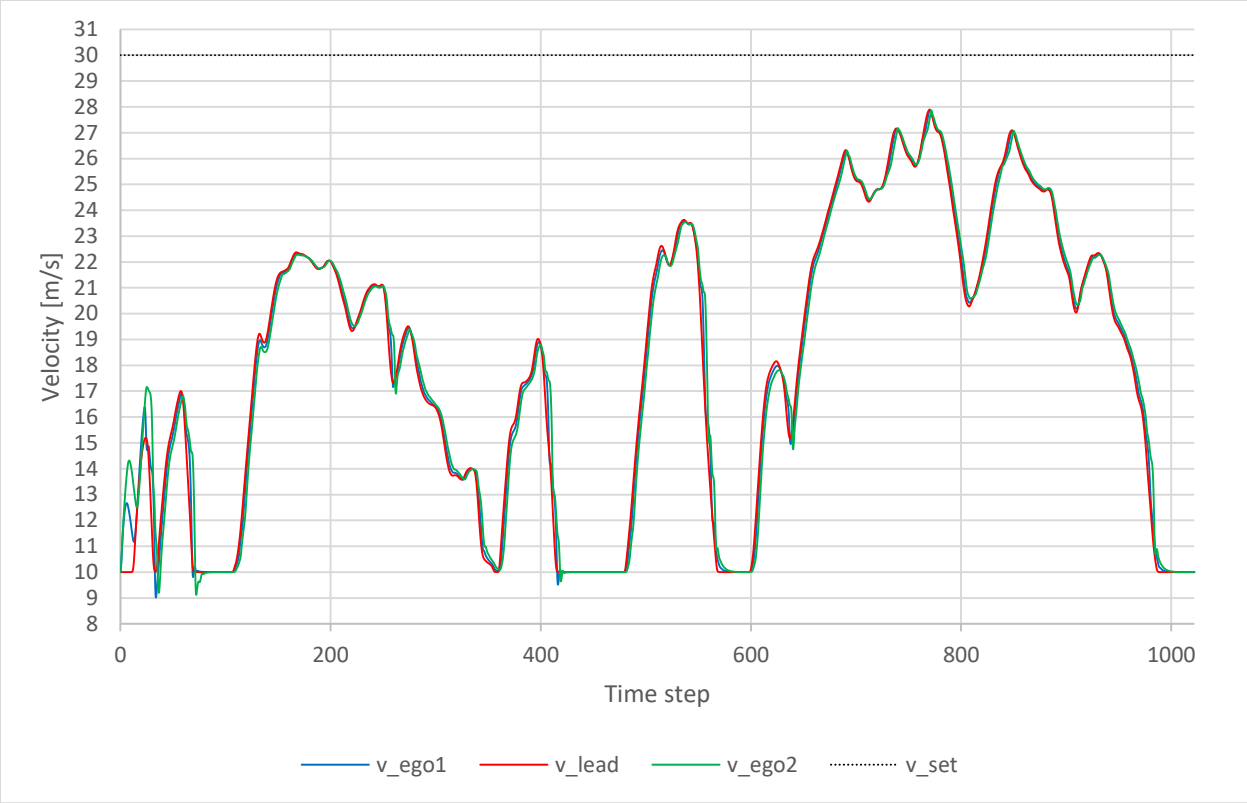

Figure 38. Three-Vehicle Platoon velocity control fourth scenario

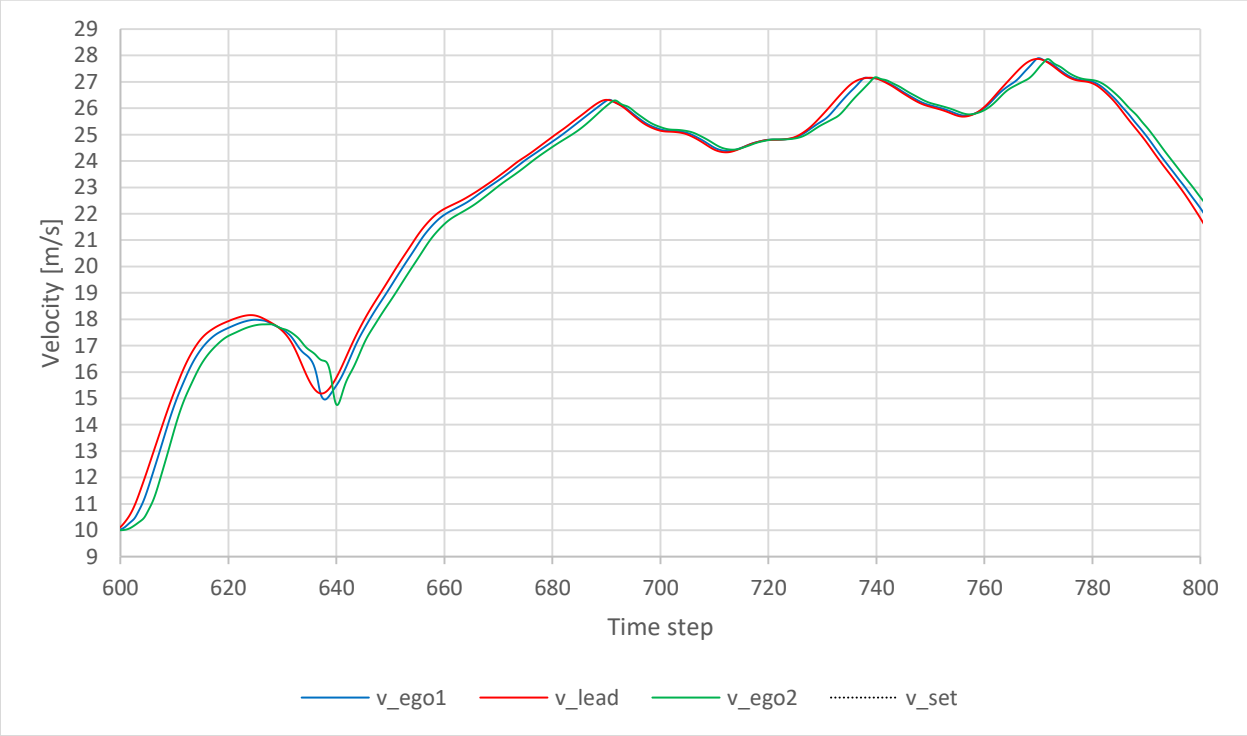

Figure 39. Zoom on Three-Vehicle Platoon velocity control fourth scenario

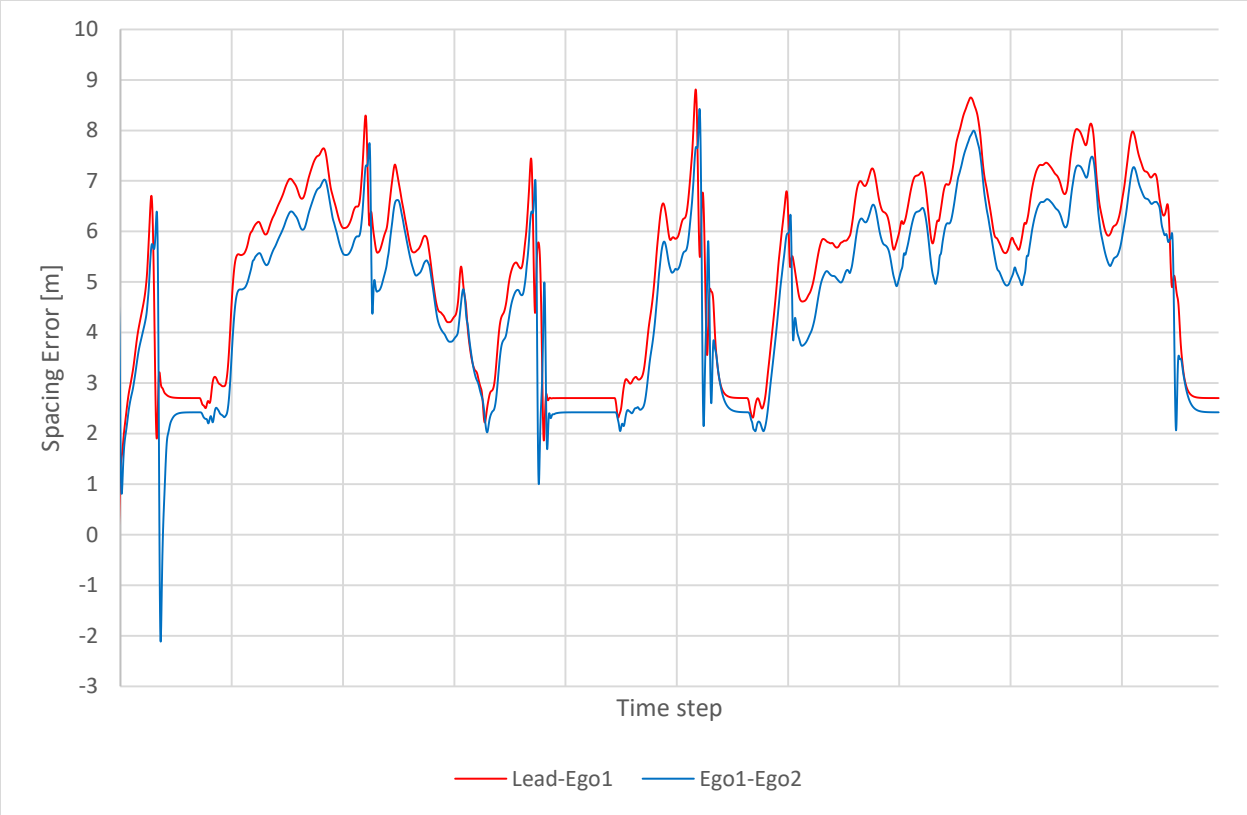

Figure 40. Spacing error of Three-Vehicle Platoon fourth scenario

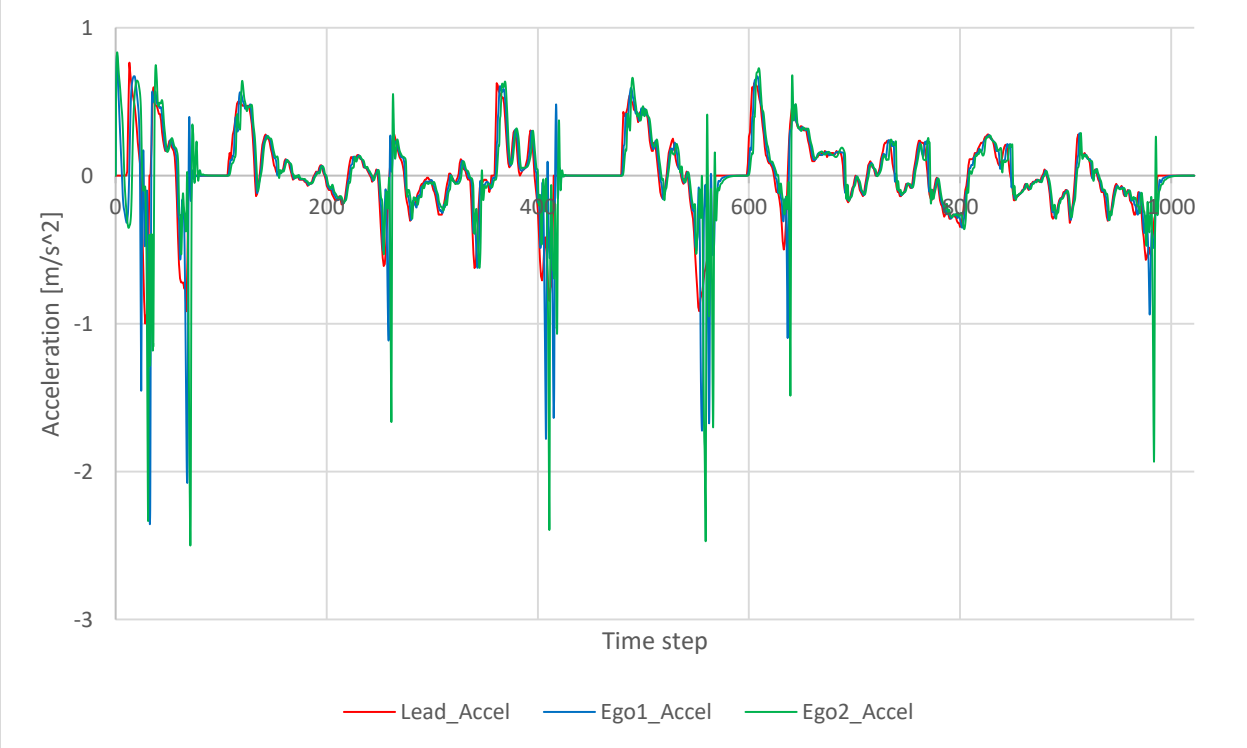

Figure 41. Acceleration of Three-Vehicle Platoon fourth scenario

4.3.2 Four-Vehicle Platoon

In this phase, a simulation involving four vehicles was conducted, incorporating a 0.1-second delay in the acceleration signal transmission between the cars. This delay was introduced to mimic real-world conditions where signal transmission is not instantaneous. The results of this simulation indicate that, despite the delay, the model effectively reduces the spacing error between the vehicles. This means that the distances between the cars become more consistent and closer to the desired safe distances, demonstrating the model's robustness and ability to maintain proper vehicle spacing even with communication delays.

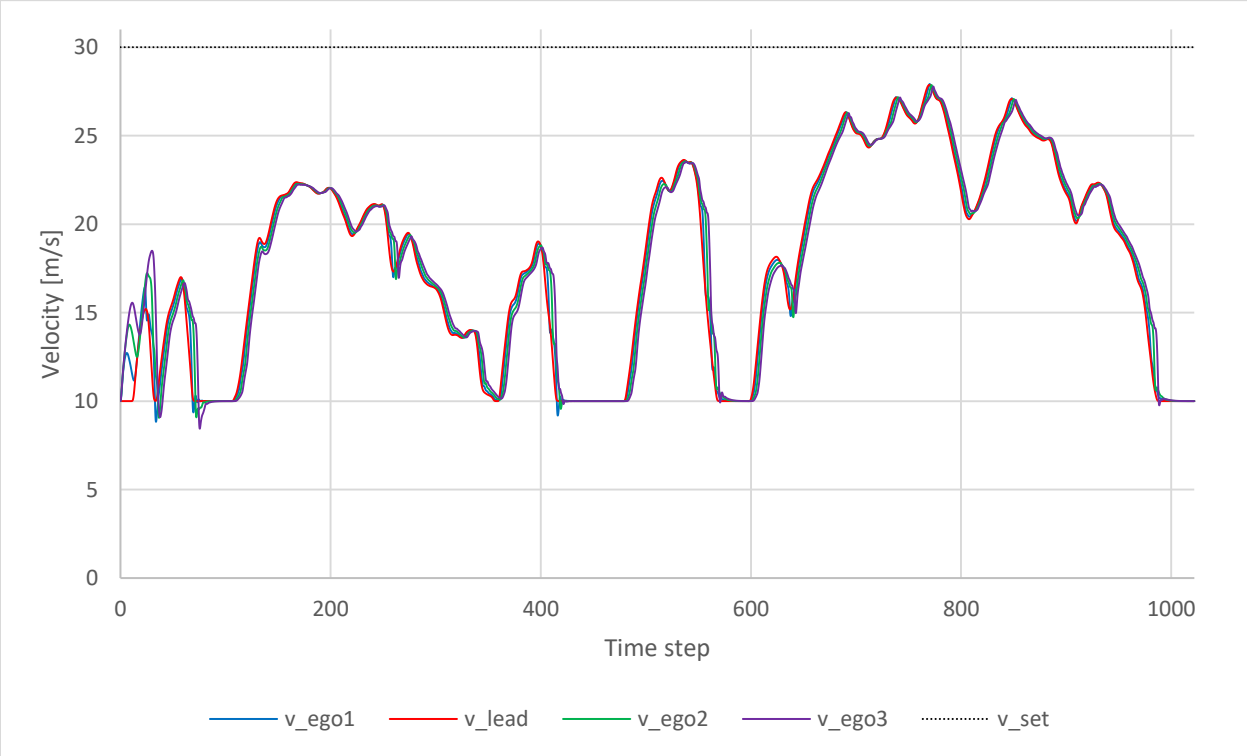

Figure 42. Four-Vehicle Platoon velocity control with 0.1 second delay

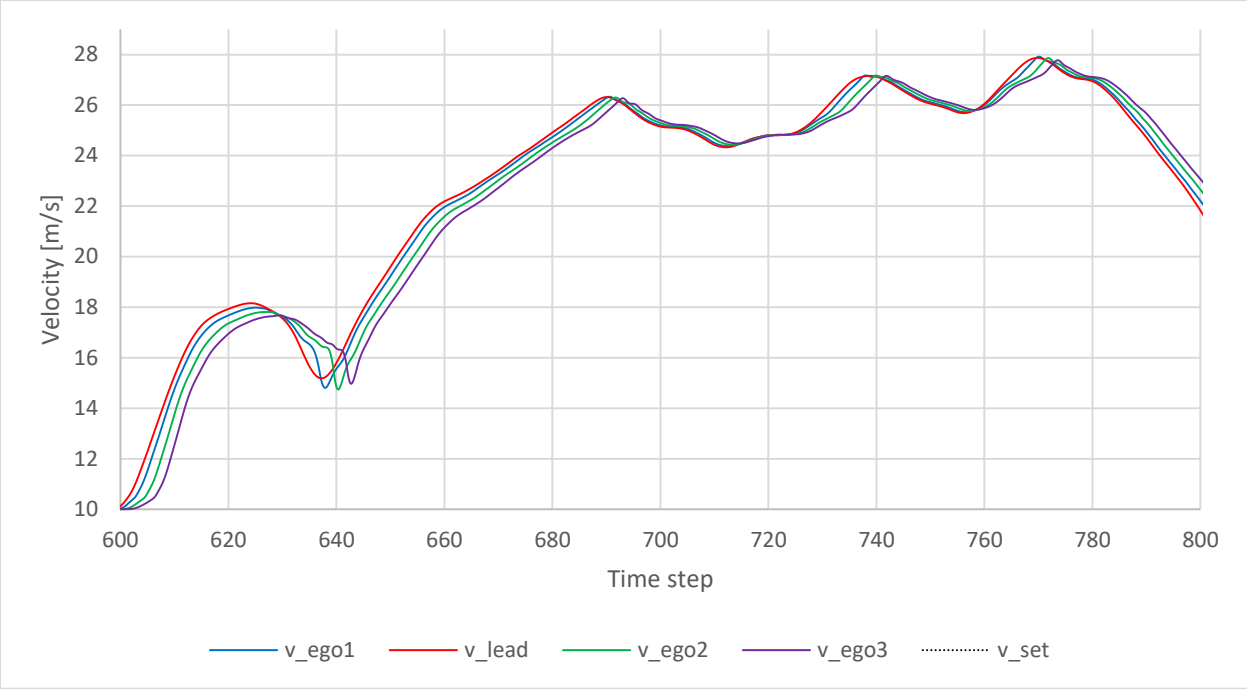

Figure 43. Zoomed on Four-Vehicle Platoon velocity control with 0.1 second delay

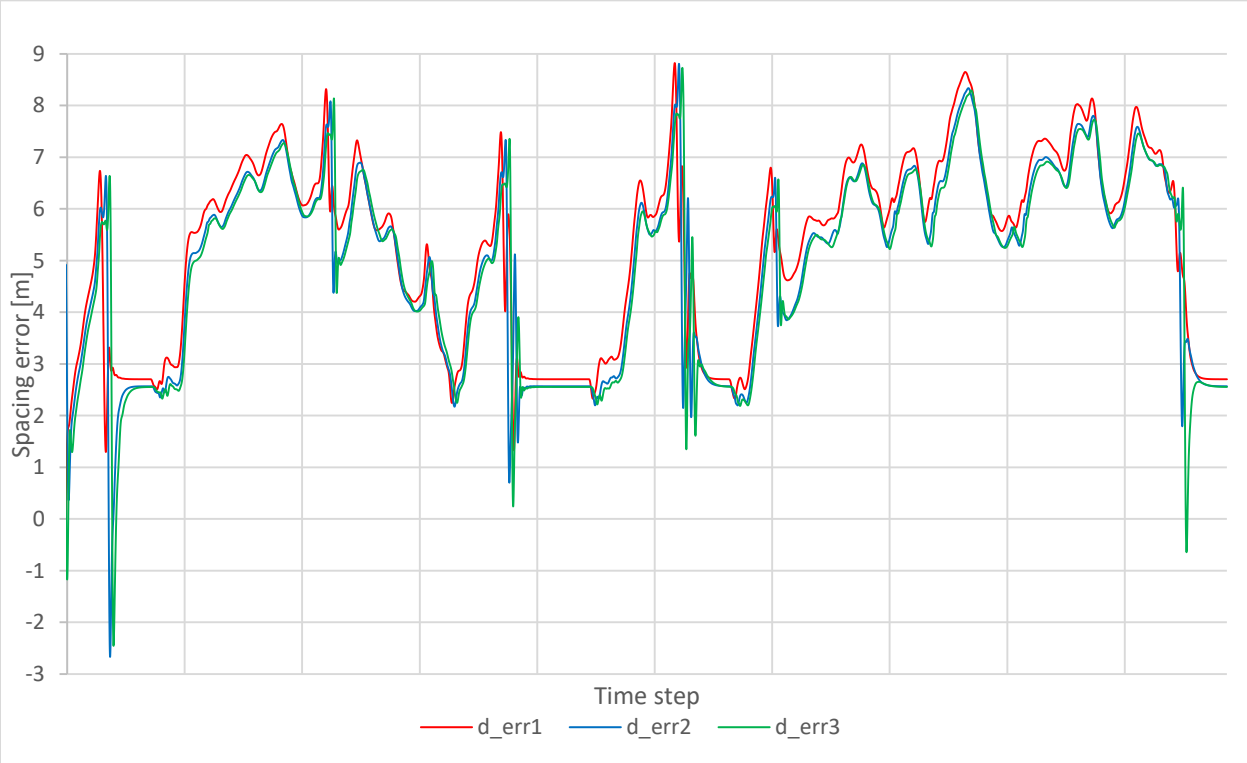

Figure 44. Spacing error of Four-Vehicle Platoon with 0.1 second delay

Table 3 presents the Root Mean Square Error (RMSE) values for spacing errors between consecutive cars, with a consistent signal delay of 0.1 seconds. The RMSE values indicate the deviation from the desired spacing between the vehicles, with specific pairs of cars being evaluated. Notably, there is a percentage reduction in RMSE from 5.04% between the lead car and Ego Car 1 to Ego Car 1 and Ego Car 2, and a further 0.4% reduction between Ego Car 1 and Ego Car 2 to Ego Car 2 and Ego Car 3. These reductions demonstrate that the model effectively decreases the spacing error between the vehicles as they follow each other, improving overall synchronization and maintaining safer distances.

Signal Delay (s)	0.1
RMSE: Lead Car to Ego Car 1 (m)	5.53
RMSE: Ego Car 1 to Ego Car 2 (m)	5.25
RMSE: Ego Car 2 to Ego Car 3 (m)	5.23
Percentage Reduction: Lead-Ego 1 to Ego 1-2 (%)	5.04
Percentage Reduction: Ego 1-2 to Ego 2-3 (%)	0.4

Table 3. RMSE and percentage reduction of spacing error between lead and ego cars

Chapter 5:

5.1 Conclusion

In this thesis, a sophisticated Reinforcement Learning (RL) approach was applied to enhance Cooperative Adaptive Cruise Control (CACC) strategies. The controller was designed using the MATLAB/Simulink Reinforcement Learning toolbox, with a particular focus on deploying a Deep Deterministic Policy Gradient (DDPG) agent. Within this framework, a comprehensive environment was established, incorporating tailored policies for both spacing and speed control.

The training of the model was carried out using a crafted reward function, designed to ensure optimal tracking performance while maintaining a safe distance from surrounding vehicles. The results were promising, demonstrating that the controller not only achieved precise speed and spacing adherence but also maintained robust safety standards across diverse driving scenarios. An element in the model's effectiveness was the integration of Vehicle-to-Vehicle (V2V) communication. To evaluate the model's resilience, different delay values in the communication signals were tested, simulating real-world conditions and challenges.

To further substantiate the safety and responsiveness of the controller, simulations were conducted. These included scenarios with varying initial velocities for the vehicles, as well as configurations involving a platoon of three and four vehicles. These simulations adhered to real drive cycles, providing a rigorous test of the model's capabilities. The RL controller consistently demonstrated its ability to adapt to changing conditions, maintain safe distances, and prevent collisions. The controller's performance in managing spacing errors and relative distances was exceptional, ensuring both safety and high levels of speed tracking accuracy.

In conclusion, this work underscores the significant potential of RL-based controllers in revolutionizing CACC systems. The developed model not only ensures safe and efficient vehicle operation but also exhibits robustness in handling varying traffic conditions and communication delays. These findings represent a substantial contribution to the field of autonomous driving, highlighting the critical role of adaptive and intelligent control strategies in modern vehicular

networks. This research paves the way for future advancements, offering a promising outlook for the integration of RL in autonomous driving systems, ultimately enhancing the safety, efficiency, and reliability of vehicular operations.

5.2 Future Work

There are several avenues for future research and development to further enhance the model presented in this thesis. One key area for improvement is the enhancement of passenger comfort through refined acceleration control. By optimizing the model to achieve smoother acceleration and deceleration, the system can provide a more comfortable and pleasant driving experience. This could involve incorporating more advanced control strategies or adjusting the reward function to prioritize passenger comfort alongside safety and efficiency.

Another significant area for potential improvement is the further reduction of spacing errors. While the current model demonstrates effective management of vehicle spacing, optimizing the model to minimize these errors even further could enhance overall traffic flow and safety. This could be achieved through advanced machine learning techniques, such as fine-tuning hyperparameters, employing more sophisticated algorithms, or leveraging larger and more diverse training datasets to improve the model's accuracy and robustness.

In summary, while the current model offers significant advancements in Cooperative Adaptive Cruise Control, there is substantial potential for further research and development. By focusing on improving passenger comfort, reducing spacing errors, integrating additional information sources, and expanding the model's applicability to more complex scenarios, future work can continue to push the boundaries of autonomous driving technology.

References

[1] D. M. P. a. M. M. C. E. Garcia, "Model predictive control: theory and practice - a survey," *Automatica,* pp. 335-348, 1989.

[2] E. B. C. Camacho, Constrained Model Predictive Control, London: Springer, 2007.

[3] Y. W. Z. P. J. H. X. W. R. K. Meixin Zhu, "Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving," *Transportation Research Part C: Emerging Technologies,* vol. 117, 2020.

[4] L. L. H. L. P. e. a. Luo, "Model predictive control for adaptive cruise control with multi-objectives: comfort, fuel-economy, safety and car-following," *Journal of Zhejiang University SCIENCE A,* vol. 11, 2010.

[5] D. A. Taku Takahama, "Model Predictive Control Approach to Design Practical Adaptive Cruise Control for Traffic Jam," *International Journal of Automotive Engineering,* vol. 9, no. 3, pp. 99-104, 2018.

[6] K. L. R. R. J. W. Shengbo Li, "Model Predictive Multi-Objective Vehicular Adaptive Cruise Control," *IEEE,* vol. 19, no. 3, pp. 556-566, 2011.

[7] A. C. a. F. B. S. Lefevre, "Autonomous car following: A learning-based approach," *IEEE Intelligent Vehicles Symposium (IV), Seoul, Korea (South),* pp. 920-926, 2015.

[8] M. G. F. C. a. L. W. D. Ernst, "Reinforcement Learning Versus Model Predictive Control: A Comparison on a Power System Problem," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 39, no. 2, pp. 517-529, 2009.

[9] J. M. a. N. L. A. Y. Lin, "Comparison of Deep Reinforcement Learning and Model Predictive Control for Adaptive Cruise Control," *IEEE Transactions on Intelligent Vehicles,* vol. 6, no. 2, pp. 221-231, 2021.

[10] Wikipedia, "Proportional–integral–derivative controller," 31 May 2024. [Online]. [Accessed 1 June 2024].

[11] S. VM, "https://medium.com/@svm161265," When and why to use P, PI, PD and PID Controller?, October 2021. [Online]. [Accessed 1 June 2024].

[12] U. o. York, "What is reinforcement learning?," University of York, York, 2023.

[13] D. S. L. V. N. P. Todd Mummert, "What is reinforcement learning?," IBM, 14 September 2022. [Online]. [Accessed 1 June 2024].

[14] G. P. S. C. Ashish Kumar Shakya, "Reinforcement learning algorithms: A brief survey," *Expert Systems with Applications,* vol. 231, 2023.

[15] MathWorks, "What Is Reinforcement Learning?," 20224. [Online]. Available: https://www.mathworks.com. [Accessed June 2 2024].

[16] R. A. Howard, Dynamic programming and Markov processes, Cambridge, MA, USA: MIT Press, 1960.

[17] R. S. &. B. A. G. Sutton, Reinforcement Learning An Introduction, Cambridge, MA, USA: MIT Press, 2018.

[18] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274.,* 2017.

[19] I. B. L. L. G. B. R. Grondman, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Trans. Syst. Man, Cybernet.,* vol. 6, p. 1291–1307, 2012.

[20] V. K. K. S. D. R. A. V. J. B. M. G. A. R. M. F. A. O. G. e. a. Mnih, "Human level control through deep reinforcement learning," *Nature,* 2015.

[21] T. H. J. P. A. H. N. E. T. T. Y. S. D. W. D. Lillicrap, "Continuous Control with Deep Reinforcement Learning," *arXiv:1509.02971,* 2015.

[22] J. P. a. M. McDonald, "Advanced driver assistance systems from autonomous to cooperative approach," *Transp. Rev,* vol. 28, no. 5, p. 659–684, 2008.

[23] C. D. a. B. Chaib-draa, "Cooperative Adaptive Cruise Control: A Reinforcement Learning Approach," *IEEE Transactions on Intelligent Transportation Systems,* vol. 12, no. 4, pp. 1248-1260, 2011.

[24] Z. B. a. R. E. P. Fancher, "Human-centered design of an ACC with braking and forward-crash-warning system," *Vehicle Syst. Dyn,* vol. 36, no. 2, p. 203–223, 2001.

[25] Z. X. Y. L. CUNXUE WU, "Spacing Policies for Adaptive Cruise Control: A Survey," *IEEE,* vol. 8, pp. 50149-50162, 2020.

[26] J. K. R. V. K. a. M. N. D. De Bruin, "esign and test of a cooperative adaptive cruise control system," *IEEE Intell Vehicles Symp,* p. 392–396, 2004.

[27] R. V. J. P. M. V. d. M. a. M. S. G. Naus, "Towards on-the-road implementation of cooperative adaptive cruise control," in *in Proc. 16th World Congr. Exhib. Intell. Transp. Syst. Serv.*, Stockholm, Sweden, 2009.

[28] D. Pomerleau, Neural network vision for robot driving, Cambridge, MA: MIT Press, 1995.

[29] G. Y. a. I. Sethi, "Road following with continuous learning," *in Proc. Intell. Vehicles Symp.,* pp. 412-417, 1995.

[30] F. L. a. D. Vrabie, "einforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits Syst,* vol. 9, no. 3, p. 32–50, 2009.

[31] J. L. a. D. C. S. Oh, "A new reinforcement learning vehicle control architecture for vision-based road following," *IEEE Trans. Veh. Technol.,* vol. 49, no. 3, p. 997–1005, 2000.

[32] D. M. a. P. Langley, Distributed learning of lane-selection strategies for traffic management, Palo Alto, CA: Tech. Rep. 98-2, 1998.

[33] J. R. Forbes, "Reinforcement learning for autonomous vehicles," Univ. California, Berkeley, 2002.

[34] C. C. a. J. H. L. Ng, "Reinforcement learning of dynamic collaborative driving—Part I: Longitudinal adaptive control," *Int. J. Vehicle Inf. Commun. Syst.,* vol. 1, no. 0, p. 208–228, 2008.

[35] MathWorks, "Train RL Agent for Adaptive Cruise Control with Constraint Enforcement," [Online]. Available: https://www.mathworks.com/.