OST
Eastern Switzerland
University of Applied Sciences

Benjamin Plattner

Vehicle Platooning using Multi-Agent Reinforcement Learning

A Study on Autonomous Driving in the CARLA Simulator

Bachelor's Thesis

OST - Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Advisor: Prof. Dr. Mitra Purandare

Co-Examiner: Dr. Sam Blakeman, Sony AI

June 2023

Abstract

We successfully trained a model using reinforcement learning that enables a car to identify and follow a leader vehicle, therefore creating a vehicle platoon in a simulation. For this thesis, we are using CARLA, an open-source 3D environment simulator for autonomous driving research based on the Unreal Engine.

Specifically, we implemented the Double Deep Q-Network algorithm to train the policy. The state is solely represented as the sensor input taken from the follower car's front camera. From that, an Atari-style Convolutional Neural Network predicts the actions' Q-values. A custom reward function, that combines the relative longitudinal and lateral positions of the two vehicles in the platoon, provides the necessary feedback of each action's performance. Actions, discrete by definition of a DQN, comprise left and right steering, acceleration and slowing down, as well as braking.

The resulting model is capable of forming and maintaining a platoon. While navigating in a multi-lane scenario, it even manages to cross intersections successfully. It enables the follower car to identify the leader vehicle and imitate its actions in a previously unseen environment.

Keywords: Multi-Agent Reinforcement Learning, Autonomous Driving, Simulation, CARLA, Platooning.

Executive Summary

The idea of autonomously driving vehicles has been around for many decades. While partial autonomy has been achieved, driverless vehicles on public roads emerged only in recent years, although in limited settings. In addition, the number of vehicles on public roads is steadily increasing, often leading to congested areas. One of the possibilities to improve the safety and to reduce congestion on public roads is platooning.

In this setting, vehicles drive behind each other and form a moving queue. While platooning has been tested on highways, it mainly relies on control algorithmic approaches to mathematically model the world. In a complex traffic scenario often found in cities, not all real-life conditions can be modelled as a mathematical formula.

A different approach is to use machine learning, specifically reinforcement learning, where an agent learns a task by repeating actions and receiving rewards depending on how good the outcome was.

Training a model using reinforcement learning requires many thousands of iterations to make any significant progress. A simulated 3D environment is best suited to repeatedly train and test such a model. This thesis uses a state-of-the-art 3D environment simulator for autonomous driving, suitably called CARLA (“Car Learning to Act”). It is an open-source simulator especially designed for research that provides realistic maps of towns, real world physics, various vehicle types, and simulation of real-world traffic scenarios.

In this thesis, we investigate how to train a model that enables a car to drive in a platoon.

We successfully trained a model using reinforcement learning that allows the follower to identify the leader vehicle and follow it in a previously unseen simulated environment. It can adapt its behavior depending on the leader vehicle’s actions, such as steering, slowing down and accelerating. It can operate in a multi-lane scenario and even manages to cross intersections. Building on this initial success, extensions such as obstacle avoidance, speed sign recognition, heavy traffic situations or a transfer to a real car can be further explored.

Figure 1: Vehicle platooning situation in the CARLA simulator with the leader and follower car.

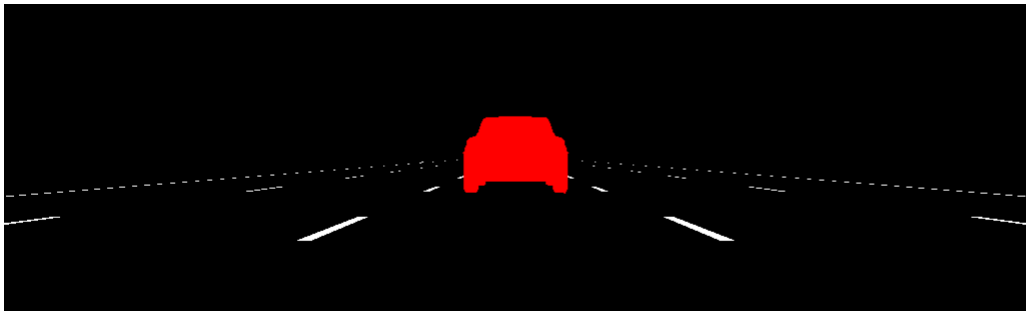

Figure 2: How the follower perceives the leader car through its camera.

Figure 3: Various metrics are collected during a model evaluation. In this scenario, the follower adjusted its actions often, but it followed very well since the scalar-projected distance to the leader is mostly around 40 meters. The point-line distance increased when the leader made a turn, but the follower imitated the action, and the distance was reduced again. Overall, a positive reward resulted.

Contents

Chapter 1

Introduction

The idea of autonomously driving vehicles has been around for many decades [1]. While partial autonomy has been achieved, driverless vehicles on public roads emerged only in recent years, although in limited settings (for example, [2], [3], [4]).

Until level 5 autonomy [5] can be reached, many years of research, testing and public policy changes in addition to educating the public are required [6]. Furthermore, the number of vehicles on public roads has been steadily increasing [7], often leading to congested areas [8].

One of the possibilities to improve the safety and to reduce congestion on public roads is Cooperative Adaptive Cruise Control (CACC), more generally known as platooning [9]. In this setting, vehicles drive behind each other and form a moving queue. While platooning has been tested on highways, it mainly relies on control algorithmic approaches to mathematically model the world [10].

In a complex traffic scenario often found in cities, not all real-life conditions can be modelled as a mathematical formula. A different approach is to use machine learning, specifically reinforcement learning, where an agent learns a task by repeating actions and receiving rewards depending on how good the outcome was.

In this thesis, we investigate how to train a model that enables a car to drive in a platoon.

Structure of the Thesis

The thesis is organized as follows:

Chapter 2 – Objective specifies the vision and the high-level goals for this thesis.

Chapter 3 – Problem Analysis looks at the different elements of this thesis that are analyzed in subsequent chapters.

Chapter 4 – Reinforcement Learning Theory introduces the concept of reinforcement learning.

Chapter 5 – Research provides an overview of previous work that is relevant for this thesis.

Chapter 6 – Design defines the important elements of reinforcement learning and how they are to be implemented.

Chapter 7 – Implementation explains in detail how the various parts are implemented.

Chapter 8 – Experiments and Results describes the various experiments and their outcomes.

Chapter 9 – Conclusion summarizes the findings and provides an outlook.

Chapter 2

Objective

2.1 Vision

The vision is to have multiple cars, each with the aim to join and maintain a platoon by locating the leader car and attaching themselves to the existing queue or forming a new one, thereby becoming a follower car.

The cars are on a multi-lane road where the platoon is driving on one lane that is chosen by the leader car. The street scenario is composed a grid road with multiple intersections which may break up the platoon and a new leader car has to be defined.

Such a platoon moves ideally at a constant speed to be most efficient, however, intersections or obstacles require it to slow down or to accelerate. Situations like turning left or right at intersections or passing obstacles on the road by either switching lanes or safely pass on the oncoming lane can be handled by the leader car of a platoon. These maneuvers have to be adapted by the follower cars so that the platoon is maintained, however, they do not violate the rules of the road or drive insecurely.

The OST is in possession of a fleet of four wheeled remote-control toy cars that are equipped with a camera. A sim-to-real transfer could enable such a fleet to organize themselves into a platoon. The leader car either drives along a pre-defined route or is manually controlled by a user.

2.2 Thesis Goals

While this vision is ultimately a desired state, the time limits of this project constrain the achievement of certain elements.

Training a model using reinforcement learning requires many thousands of iterations to make any significant progress. A simulated 3D environment is best suited to repeatedly train and test such a model. This thesis uses a state-of-the-art 3D environment simulator for autonomous driving, suitably called CARLA (“Car Learning to Act”). It is an open-source simulator especially designed for research that provides realistic maps of towns, real world physics, various vehicle types, and simulation of real-world traffic scenarios.

This thesis focuses on the simulation part with the goal of training a car to follow a leading vehicle, thus forming a platoon. The outcome, in the form of simulated cars that learn to form such a platoon, could be applied to the street setting with cars or delivery robots, but also to an industrial or even home setting with multiple autonomous robots.

To achieve this goal, the following components need to be defined:

- the Reinforcement Learning algorithm,
- the simulation environment,
- the simulation platform,
- the monitoring of progress.

For the simulation itself, various scenarios are defined in which the agents are trained. This comprises:

- straight and curved roads,
- intersections with dynamic lights,
- multiple lanes,
- obstacles.

Specifically, we focus on a one leader, one follower scenario, where the leader is controlled by a pre-defined algorithm (i.e., it behaves perfectly) and the follower is the one being trained.

2.3 Approach

In this thesis, these four components (RL algorithm, simulation environment, simulation platform, and montoring progress) are researched and evaluated in detail. We provide solutions along with comparisons of various aspects.

Chapter 3

Problem Analysis

The problem that is addressed in this thesis is how a simulated car can learn to participate in a vehicle platoon. Such a problem requires multiple components in order to produce a viable result.

3.1 Reinforcement Learning Algorithm

In order for a simulated car to learn, a learning algorithm is required. Since we are using a simulation, we can produce the data ourselves that is necessary since the agent can interact directly with the environment.

Such a setting is predestined for Reinforcement Learning (RL). However, there exists a multitude of RL algorithms that have benefits, such as sample efficiency or speed, but also disadvantages, such as long training times or instability.

In addition, a RL algorithm requires a reward function, which determines the reward that an agent receives when it interacts with the environment. Such a reward function is crucial for the speed of the training, but also for correctly incentivizing the agent.

Then, the algorithm needs to be implemented and a model (or more exactly, the agent's policy) has to be learned. One option is to implement everything from scratch. However, next to being quite time consuming and error prone, there exists a number of learning frameworks that implement the basic algorithms and are tested quite thoroughly. Therefore, one of these machine learning frameworks needs to be selected.

3.2 Simulation Environment

A simulation environment can range from extremely simple, such as a two dimensional (2D) grid where the only options are up, down, left or right. But it can also be a three dimensional (3D) environment which has the quality of a video game.

To simulate cars, there exist a number of environments, both 2D and 3D, which range from simple grid-like settings to high-resolution 3D simulations. They have different complexities, features and system requirements.

Another approach would be to build the simulation ourselves, therefore controlling every aspect of the simulation, but with the cost of spending time to build all the required simulation assets.

3.3 Simulation Platform

To run a simulation, a platform in the form of a computer with a graphics card is required.

While a powerful laptop could suffice for such a task, simulations are often running over multiple hours or even days, hence a more standalone solution would be ideal.

3.4 Monitoring Progress

To monitor progress during the training phase metrics are required that are regularly updated. They indicate whether the training is going in the right direction or if there is an issue with the parameters. Again, a few tools exist that provide such a service.

Chapter 4

Reinforcement Learning Theory

This chapter aims to provide the reader with some fundamental concepts of reinforcement learning (RL). It contains a high-level description of elements that are relevant for this thesis. An excellent and in-depth treatise of RL can be found in the seminal book from Sutton and Barto [11]. If the reader is already familiar with RL, then this chapter can be safely skipped.

4.1 Basic Elements

Reinforcement Learning is one of the three machine learning paradigms, next to supervised and unsupervised learning. Where supervised learning requires a labelled dataset to train a model effectively, and the unsupervised learning is often used for classification tasks (such as spam email classification), reinforcement learning generally generates the data itself and uses rewards and exploration for training a model [11, p. 1f].

However, to explain the RL learning process, we need to take a step back. A typical RL process consists of five elements: agent, environment, state, action and reward. These elements originate from the theory of Markov Decision Processes.

4.1.1 Markov Decision Process

A finite Markov Decision Process, or finite MDP, is about sequential decision making, where one step after the other an action is taken and a reward is received. However, this reward is not just the result of that single action, but all future rewards that would follow from it [11, p. 47].

The finite MDP is a mathematical description of this problem. It allows to calculate so-called value functions, which represent the expected reward of taking a specific action in

a given state [11, p. 58]. These expected rewards are what influences the decision making process of an agent. Ultimately, its goal is it to maximize the cumulative rewards, and the finite MDP is a fundamental element to calculate them.

The finite MDP is depicted in Figure 4.1, where the concept of RL is applied to a game, but it could be an arbitrary environment.

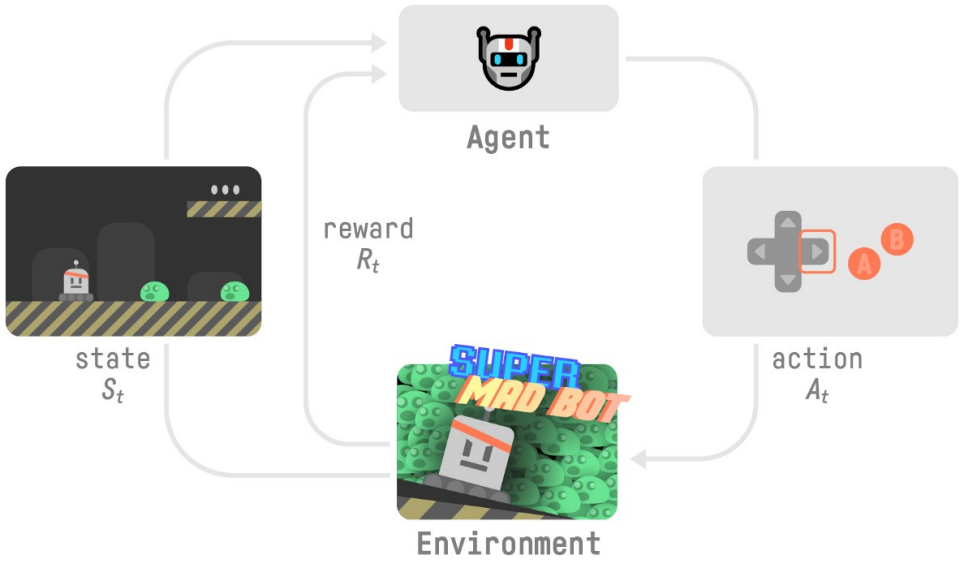

Figure 4.1: The reinforcement learning process applied to a game (source: [12]).

4.1.2 Agent

An agent is an entity that can observe the state of a given environment, take actions and perceive rewards. Depending on the rewards it learns to adjust the optimal action for a given situation [11, p. 47f].

The agent can be anything, from a robot, to a car, or a player in a game and even a human (examples: [13], [14], [15], [16]). It is the central entity in RL, since it learns from the interactions with the environment.

4.1.3 Environment

RL requires an environment to work. A classical example is the Atari game suite, but board games such as Go, or even multiplayer games like Dota 2 are examples where RL is used (examples: [17], [18], [19]).

To effectively use RL, it is necessary to have an environment available where an agent can interact with the environment by taking actions and receiving rewards that inform it how good or bad this action was [11, p. 48].

4.1.4 State

The state refers to the environment in which an agent is currently in. If the agent takes an action, it will transition from the current to the next state to perceive its consequences.

However, there are no past states in the sense that the finite MDP requires the current state to comprise all information that is relevant for the present and the future. This is a Markov property, which should not be violated [11, p. 49].

4.1.5 Action

An action is executed by the agent within an environment [11, p. 48]. It can be a discrete action, such as going left or right in a maze, or it can be continuous, like steering a wheel. The action has an influence on the next state, even if it executes nothing, but it is the element that triggers the transition to the next state along with the reward that is fed back to the actor.

4.1.6 Rewards

The reward is the result from an actor taking an action. The finite MDP uses a reward to calculate the expected cumulative rewards, based on which the agent adjusts its behavior since it tries to maximize it.

The reward is in the form of a numerical value which can be arbitrary, but it usually lies between -1 and +1 to avoid exploding cumulative values. The function that determines the level of the reward can be very simple, such as when an agent crosses a finishing line for which event a reward of +1 is given. It can also be used to penalize an action, such as returning a value of -1 in case there was a collision, for example in a car racing game [11, p. 53]. This function can also be more complex, where the resulting reward depends on a measure, such as the distance to a target.

The reward is an important feature of the learning process. While it can speed up the learning when it is properly defined with the correct incentives, it can also slow down or even prevent learning when it is mis-specified [20].

4.2 Reinforcement Learning Process

This far, we have not touched the fact how the agent is learning. What happens in every state is that the agent needs to decide which action is the one that maximizes the cumulative expected reward. The basis on which the agent supports its decision is called the policy.

The optimal policy is what the agent needs to learn. That means, at every state the policy informs the agent which action is best [11, p. 58]. The policy is usually updated when a new state is observed along with its reward. To achieve the optimal policy, the agent should observe many different states, which implies that the longer the agent learns, the better the policy should become in an ideal setting.

4.2.1 Policy

In classical reinforcement learning where, such a policy is often represented as a table. Simplified, this table consists of a state, potential actions in that state and the next state from an action along with the current and expected reward. Such a table can become quite complex when there are thousands or millions of different states. It becomes quite difficult and inefficient to use such a table to keep track of an optimal path through this way of representing the policy.

This is where approximation techniques come into play [11, p. 196]. Given the inputs of an observation along with its corresponding reward, a deep neural network is trained that learns to approximate the relationships between any given observation and their reward.

When the observation is an information in the form of one or more values, like x/y/z-coordinates of an agents position, then often a densely connected neural network is used. When the input form is of more complex structured values, like an image, then a convolutional neural network (CNN) is better suited. Both networks do learn the optimal weights that map the corresponding action values from any given input.

4.2.2 CNN

The state in a simulation is often an image input. This does not have to be the case, as other state values, such as position, velocity, distance to objects, etc. could be used. However, since the goal of this thesis is to train a model that enables a car to follow the leading car, and at some point transfer this model to a real car, the camera input seems to make most sense.

Since a single input image represents a single state, there are many thousands or millions of different states possible. While it is not feasible to create a state-action pair for

every input image, especially if only a single pixel value is different, the deep reinforcement learning approach is best suited to approximate such states. They can ingest an image with red, green and blue color channels (an RGB image), however, they require a convolutional neural network (CNN) to process such an image efficiently.

CNNs are the standard deep network architecture for extracting features from images. They use the concept of filters, which are small grid-like patches that are mostly applied to every area of an image. Various filter combined can identify shapes within an image. Filters in early stages of the network usually extract simple shapes, such as edges, whereas filters in later stages can extract more complex structures, such as faces or the shape of a car in our case.

CNNs can therefore identify features from the input image, and with the different actions at the last layer, can predict the Q-values for each action at that state (i.e., image). These Q-values can then be used in the RL algorithm that selects the best action given its policy.

Chapter 5

Research

This chapter presents a literature overview of research done in the fields relevant to this thesis.

5.1 Simulators

There exist a number of different simulators that cater for performing research in the autonomous driving space as presented in Table 5.1.

For this thesis, we choose the CARLA simulator, since it is well-known, established in the research community, actively developed and provides important features, such as a Python API.

5.2 CARLA

Only a handful of papers were identified that apply reinforcement learning using the CARLA simulator [22].

CARLA provides a leaderboard [48] which encourages teams to evaluate the driving proficiency of their algorithms. The top three placed teams devised their own algorithms ([49], [50], [51]), however, they use reinforcement learning only sparsely of for sub-components, if at all. They mainly leverage transformers to extract information from multi-modal sensor input.

A more recent publication [52] is implementing a Deep Q-Network (DQN), but also a Deep Deterministic Policy Gradient (DDPG) to train a model that enables a vehicle to follow a pre-defined path in the CARLA simulator. They were mainly using a segmented black-and-white image that contains a line of the optimal path. However, they also tried an approach

Table 5.1: Overview of Autonomous Driving Simulators

Name	Description	Active	Link	Paper
CARLA	Open-source simulator for autonomous driving	Yes	[21]	[22]
Donkey	Open-source self driving car platform for remote control cars	Yes	[23]	n/a
Duckietown	Open-source platform to teach and research autonomous driving.	Yes	[24]	[25]
FLOW	Open-source deep reinforcement learning framework for mixed autonomy traffic based on SUMO.	No	[26]	[27]
MACAD	Open-source multi-agent connected autonomous driving simulator based on CARLA.	Yes	[28]	[29]
MetaDrive	Open-source with infinite scenarios for generalizable RL	Yes	[30]	[31]
Microsoft AirSim	Platform for AI research with autonomous vehicles.	Yes[1]	[32]	[33]
Nocturne	Open-source 2D driving simulator for multi-agent coordination.	No	[34]	[35]
NVIDIA DRIVE Sim	End-to-end simulation platform for autonomous driving.	Yes	[36]	n/a
Plexe	Open-source 2D platooning simulation extension for SUMO and Veins.	Yes	[37]	[38]
SUMMIT	Open-source simulator for generating data for urban traffic based on CARLA.	Yes	[39]	[40]
SUMO	Open-source traffic simulator	Yes	[41]	[42]
Veins	Open-source 2D simulator for vehicular network simulations.	Yes	[43]	[44]
VISTA	Open-source data driven simulator for autonomous driving perception and control.	No	[45]	[46]

[1] The project re-launched with a simulator only for drones [47]; cars are no longer supported.

with a pre-trained CNN where they extract the waypoint information from an RGB image, however, they noted they faced some challenges during the training, but without providing more details.

As a reward, they provide the algorithm a positive incentive for staying close to the lane's center line along with the longitudinal velocity, while penalizing the lateral distance and velocity. The discrete action space for the DQN consists of 27 different actions, each has different discrete values for throttle and steering. In their evaluation, the DDPG seems to follow the path quite well, however, they had some challenges with the DQN which did not

perform well. Their conclusion is that the DDPG requires far fewer training episodes than a DQN in addition to being capable of a continuous action space.

5.3 Cooperative Adaptive Cruise Control

Much research is done on Cooperative Adaptive Cruise Control (or CACC). As previously noted in chapter 1, CACC has been tested for a few years now, however, mainly using control algorithms.

5.3.1 Control System Approach

A review of the CACC landscape has been done by [53] where they review three types of systems:

- Communications
- Driver characteristics
- Controls

They also identify three different types of following, or information flow topology as they call it:

- Predecessor following
- Predecessor-leader following
- Bidirectional

The predecessor following is the most widely used topology, where the follower receives information from the predecessor. [54] also notes, that they assume the same speed, which results in a homogeneous platoon. They consider a platoon system as a combination of vehicle longitudinal dynamics, information exchange flow, decentralized controllers and inter-vehicle spacing policies. In addition, all dynamics of the participating vehicles have to be the same to qualify as a homogeneous platoon, as noted by [55], otherwise it would be heterogeneous.

The problem with heterogeneous platoons is the inter-vehicle spacing, which may change over time and may differ within the platoon. Ideally, a stable inter-vehicle distance is achieved, a concept that is known under the term string stability. String stability is important for a platoon. However, if vehicles do not always drive at the same speed, string stability cannot be guaranteed.

[54] also note that the string stability of the vehicle platoon requires that either the vehicle's spacing error, states or the control input does not amplify upstream through the platoon. If the vehicle platoon is not string stable, a small perturbation on one vehicle can propagate through the platoon and cause uncomfortable driving experience or even dangerous situations.

This up- or downstream propagation of the spacing between vehicles was addressed by [56]. Heterogeneity in a platoon can result from vehicles with different acceleration dynamics due to their size, weight or motorization, but also the road features like curves or uneven terrain. [56] propose an algorithm that achieves string stability even in heterogeneous platoons.

5.3.2 Reinforcement Learning

Reinforcement Learning can be applied to the problem of string stability as proposed in [57]. They model the system as a Markov Decision Process and incorporate stochastic game theory. The string stability was proved where the proposed system was capable of damping small disturbances throughout the platoon. In particular, as the second car attempts to track the leader, slight oscillations result. This oscillation is passed onto the cars following, but as they move farther in the platoon, the oscillations decrease, implying stability.

Another application of RL in platooning considers the safety of such algorithms. Model-based [58] and model-free [59] algorithms are proposed, that use Lyapunov functions to guarantee stability in a system. However, finding a suitable Lyapunov function is not straight-forward, something that was addressed by the latter of the two papers, also proposing a Safe Deep Q-Network. However, as with DQN algorithms, this approach applies only to discrete action spaces. Originally, Lyapunov functions were first studied by [60].

The simulation of traffic scenarios and autonomously driving vehicles apart from platooning are a widely-researched area. As presented in the simulator overview in section 5.1, which only represents a subset of the available autonomous driving simulators, the field is quite an active area of research. There, many use RL to train an agent to act in the simulator (e.g., [29], [14], or [61]) In addition to simulators, benchmarks to evaluate RL algorithms in traffic situations have also proposed (e.g., [62]), but also using RL for improving the traffic flow in areas with bottlenecks (e.g., [63]).

[64] use a dueling deep Q-network in a platooning scenario. They analyze the platooning behavior in terms of delay of arrival due to congestion and reducing the overall fuel consumption. Mainly, their experiments are set in a 2D simulation and intend to align the traffic light state and the speed of a platoon using a 5G mobile network. Their goal is to train an

agent that controls the platooning and the traffic light state behavior using deep RL. Its incentivize is to reduce the number of times a platoon comes to a full stop but also to reduce unnecessary accelerations, hence reducing the fuel consumption. However, while a reduction in delay and fuel consumption are important and some of the main benefits of platooning, they are their sole focus in this paper.

[65] use a genetic algorithm in combination with RL to study the management of a platoon. By enabling the participating vehicles to use 5G to connect to the leader vehicle in a platoon, the leader vehicle from each platoon can aggregate information from various other platoons and their connected vehicles. Hence, the leader of a platoon has all information available to make decisions. For the learning, they use a replay buffer that is sorted according to a rank so that the most important experiences are sampled more frequently. They use the 2D Plexe [38] and SUMO simulators [42] to validate their model with a focus on string stability. However, their focus is on the management and organization of a connected platoon (i.e., a CACC), rather than on the vehicle itself participating in the platoon and learning the correct actions. Nonetheless, their paper makes very good observations about the previous research in this field, especially in inter-vehicle connected platoons.

5.3.3 Multi-Agent Reinforcement Learning

The field of platooning using reinforcement learning seems not very well researched. While a recent survey of deep reinforcement learning in the autonomous driving space [66] identified some applications, their survey did not mention platooning. However, they noted that the multi-agent RL (MARL) is a research topic, but under-represented in the autonomous driving space.

Another recent study combines MARL with delivery vehicles to optimize the path planning [67]. Especially, coordination among different agents in the same environment is analyzed so that they chose the optimal path to their destination. However, the simulation part is only focused on the path planning, but not on the vehicle learning to navigate autonomously.

In a similar vein, [68] use reinforcement learning to train the leader vehicles of two separate platoons to successfully lead them in a roundabout. They use the FLOW framework [27] that connects RL libraries to the SUMO simulator [42] for the training and evaluation of their approach. While they use the Trust Region Policy Optimization (TRPO) RL algorithm, it is again only for the leader vehicle, whereas the follower vehicles are controlled by the simulator's traffic manager. Interestingly, they perform a simulation-to-reality transfer, where they port the trained model, that had some noise added to states and actions, to a toy car, which seemed to work quite well. However, they note that a roundabout-loop-detector

and eventually vehicle-to-vehicle communication are required to provide the information about each state. In addition, they observed a reduction in travel time.

Chapter 6

Design

In this chapter, the design of the reinforcement learning is explained in more detail.

6.1 Algorithm

The selection of an appropriate RL algorithm has many implications, such as the number of actions an agent can take, the performance of the learning speed or the available computing power.

6.1.1 Double Deep Q-Network

For the reinforcement learning part, we use a Double Deep Q-Network, as proposed by [69]. The main reasons for using a Q-learning based network are:

1. sample efficiency due to the replay buffer
2. simpler to implement from scratch than more recent models (e.g., PPO)
3. discrete action space
4. more stabler than a simple DQN

The advantage of adding a second, so called target model, within the Double DQN is the reduced fluctuations from one weight update to the next. DQNs are generally prone to overestimating Q-values, since they only use the maximum Q-values from each state-action pair. While some overestimation is not a problem if it is uniformly distributed, concentrated higher values at some states pose an issue. This can be overcome by decoupling the prediction process during training from the one that updates the model. Eventually, the target model gets updated every so often with the weights form the other online model.

6.1.2 Other RL Algorithms

We decided against using a Proximal Policy Optimization (PPO), Soft Actor Critic (SAC), or other more recent algorithms since while the focus of this thesis is to train a model, it is also about setting up the simulation and training environment. Implementing another more complex algorithm from scratch can be a daunting task when embarking up on the reinforcement learning world (e.g., for PPO see [70]). However, by leveraging tried and tested RL algorithms from a library such as Stable-Baselines3, as described in subsection 7.3.2, such algorithms can be incorporated with a reasonable effort and are less error prone.

6.2 Actions

By using the Double DQN, the output of such a model is restricted to discrete values. This was an intentional choice since it simplifies the steering and acceleration to a certain degree.

For the first experiments we defined four different options from which the agent can choose. There, the steering was fixed to a value. However, this resulted in a zigzag course, since the vehicle is able to steer from left directly to right.

In a more realistic scenario that was introduced in the second part of the experiments, the fixed steering value was switched for a delta value. This implied that the steering value increased or decreased by a small value with each action that has an influence on the steering.

The different options that the agent can choose from are outlined in Table 6.1.

Table 6.1: Actions

Action	Description	Fixed Steering[1]	Delta Steering[1]	Throttle	Braking[2]
0	Straight ahead[3]	0.0	0.0	0.5	0.0
1	Left turn	-0.2	-0.005	0.5	0.0
2	Right turn	+0.2	+0.005	0.5	0.0
3	Slow down	previous value	0.0	0.0	0.0
4	Braking	previous value	0.0	0.0	0.25

[1] Only fixed or delta steering is allowed in an algorithm.
[2] Braking was introduced for the later part of the experiments.
[3] The description is a bit misleading for the delta steering.

6.3 State

The CNN, as briefly described in subsection 4.2.2, is used to process the state, or image input, and predict the Q-values of every action.

At first, we used an InceptionV3 CNN, which has about 22 million parameters to train for an input image of size 128x128 and four actions in the last (dense) layer. This proved to be excessive since the training is quite slow, and a much smaller CNN was needed.

Eventually, we settled with the CNN that was used in the Atari paper [71], and detailed in this post [72]. The network architecture is provided for a 64 by 64 pixel image in Figure 6.1, but described in more detail in Table 6.2.

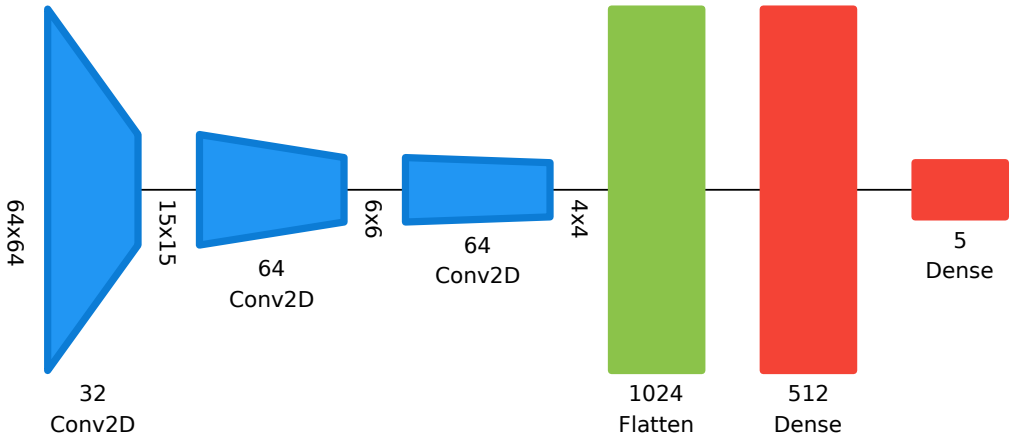

Figure 6.1: Atari CNN visual network architecture (build source: [73])

Table 6.2: Atari CNN

Layer	Type	Input Shape	Filters	Kernel Size[1]	Strides	Activation
0	Conv2d	64x64x12[1]	32	8x8	4x4	ReLU
1	Conv2d	32	64	4x4	2x2	ReLU
2	Conv2d	64	64	3x3	1x1	ReLU
3	Flatten					
4	Dense	1,024	512			ReLU
5	Dense	512	5[2]			ReLU

[1] The image input size's channel is 12 due to 3 color channels stacked 4 times.
[2] The number of actions, i.e. 5.

6.4 Reward Function

The design of the reward function requires significant attention as it provides crucial feedback to the learning algorithm how good or bad each action was.

6.4.1 Vector Geometrical Approach

In our setting, the follower car should be in line with the leader car. This means that the follower car should ideally be driving right behind the leader car. If we only calculate the Euclidean distance between the two cars, it could be anywhere within that distance, for example driving next to it while still receiving a positive reward.

Since we receive the 3D forward unit vector from the leader car from the CARLA simulator along with its current location as a 3D point, we can calculate the distance from the follower's 3D point to the leader car's center line.

Equation 6.1 provides the formula for calculating the distance d_{pl} between a point and a line in a three dimensional space.

$$d_{\mathrm{pl}} = \left\| \overline{LF} \times s \right\| \tag{6.1}$$

Where L is the 3D vector of the Leader, F the 3D vector of the follower, and s the forward vector of the Leader [74].

Another important factor is the location of the follower relative to the leader vehicle. The follower should ideally be placed behind the leader and not in front of it. One approach is to use scalar projection [75], as defined in Equation 6.2.

$$d_{\mathrm{sp}} = \overline{LF} \cdot F \tag{6.2}$$

In this case, the vector between the leader and follower is projected onto the follower forward vector using the dot product.

The resulting scalar d_{sp} indicates the distance between the follower and the leader on the forward vector:

- Negative scalar value: the follower is behind the leader
- Positive scalar value: the follower is in front of the leader
- Zero scalar value: the follower and leader are at the same location regarding forward directions

With these formulas, we can determine where the follower is located relative to the leader vehicle.

To summarize, the *point-line distance* is the orthogonal distance between the follower car and the leader's forward vector, independent of their left or right distance. The *dot product* is the cosine angle between the two vehicles' forward vectors which tells us if they are facing the same direction (i.e. $\cos(0) = 1$). Finally, the *scalar-projection distance* informs us if the follower is behind the leader vehicle (negative value) or in front of it (positive value). In fact, the Euclidean distance is no longer needed because the scalar projection has all that information in addition to the relative position (negative and positive).

An experiment to test the correctness of the described distance measures was conducted and the results are presented in Figure 6.2. In this setting, the follower car did a few full circles while the leader car just drove straight ahead. Therefore, the angle (dot product) between the two cars oscillates between ± 1, whereas the scalar-projection distance increases (so does the Euclidean distance). The point-line distance oscillates, too, since the follower car drove away from the leader's center line while circling.

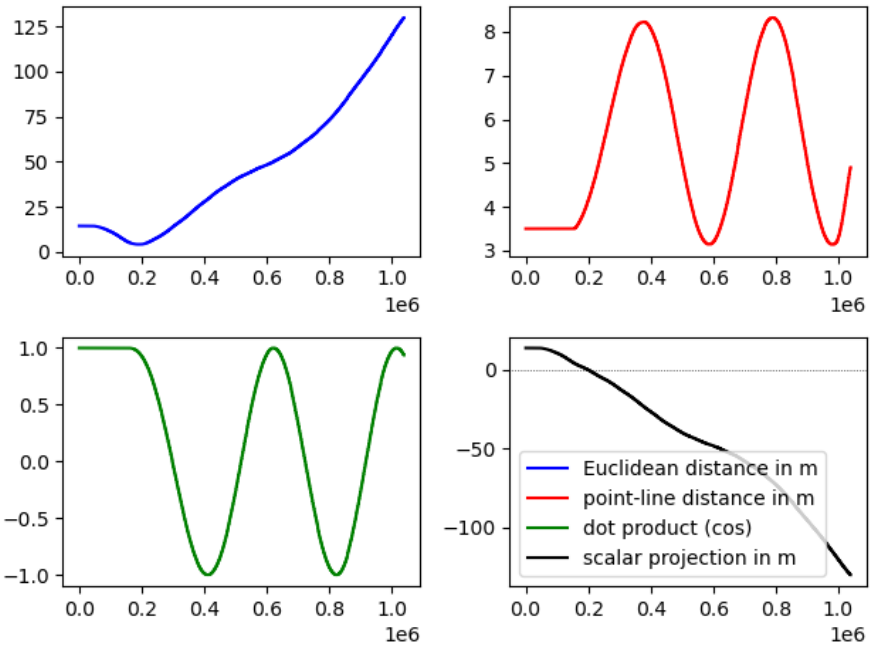

Figure 6.2: Comparison of various distance metrics for the reward function.

6.4.2 Reward Function Design

The reward functions should be generically applicable. An earlier version of it that was briefly explored for this thesis used linear functions where at some fixed point the reward started to change linearly. But, outside this sometimes narrow range there was no

indication in which direction an improve in rewards can be expected. Ideally, a Gaussian or a similar distribution function should provide some small reward increments even at the outside (i.e., a slightly positive marginal reward), while providing bigger increases the closer the metric is toward its pre-defined optimal value.

6.4.2.1 Apodization

There exist a group of functions called apodization functions, that are often used in signal processing to de-noising an input. They usually normalize the input around $\mu = 0$ with a response value of 1, and cut off the response value at slightly above 0 at around ± 1. The Gaussian apodization function is in the form of a bell-shaped curve (see an overview of such functions in: [76]).

This Gaussian apodization function seems to be an ideal form to normalize an input to the range of 0 to 1. With that, even infinite real values (such as distance) are reduced to a known range. The Gaussian apodization function also works with the usual parameters, μ and standard deviation, to influence its shape and position.

Point-Line Distance For the the point-line distance, which is used for generating a measure of the longitudinal distance of the follower car to the leader car's center line, an ideal value is 0. Hence, it does not even need to be scaled if a standard deviation of 1.0 meter is used, as can be seen in Figure 6.3.

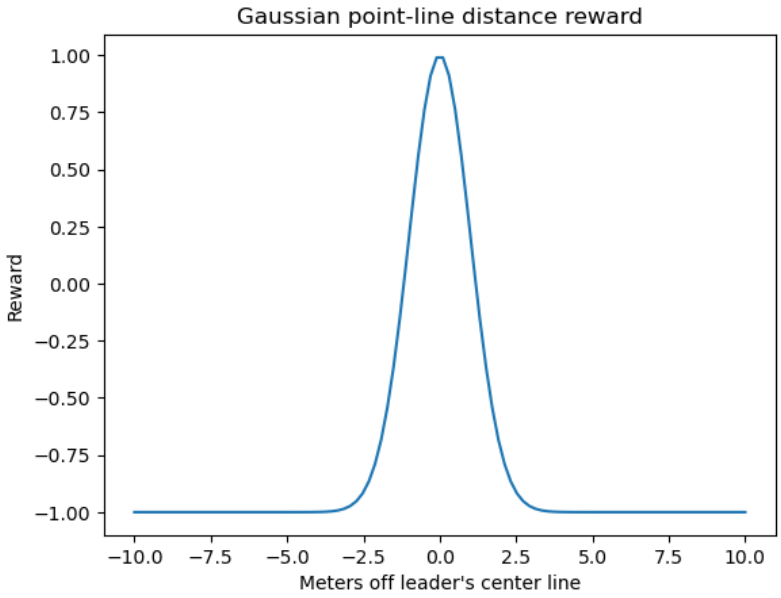

Figure 6.3: Point-line reward for various distances

Scalar Projection Distance The scalar projection distance, which is used to generate the metric of the lateral distance of the follower car along the leader's center line independent of the follower's longitudinal position, has an ideal distance around 20 meters. Since the position behind the leader car is in the negative space, the ideal metric is calculated as -20.0 meters. Using the Gaussian apodization function, the mean is corrected by that value. The standard deviation is chosen as 10.0 meters, hence the function returns a value of 1.0 when the follower is exactly 20.0 meters behind the leader. A value of 0.0 is returned if the cars are further than 50 meters apart, mainly due to the limitations of floating point precision, as can be seen in Figure 6.4.

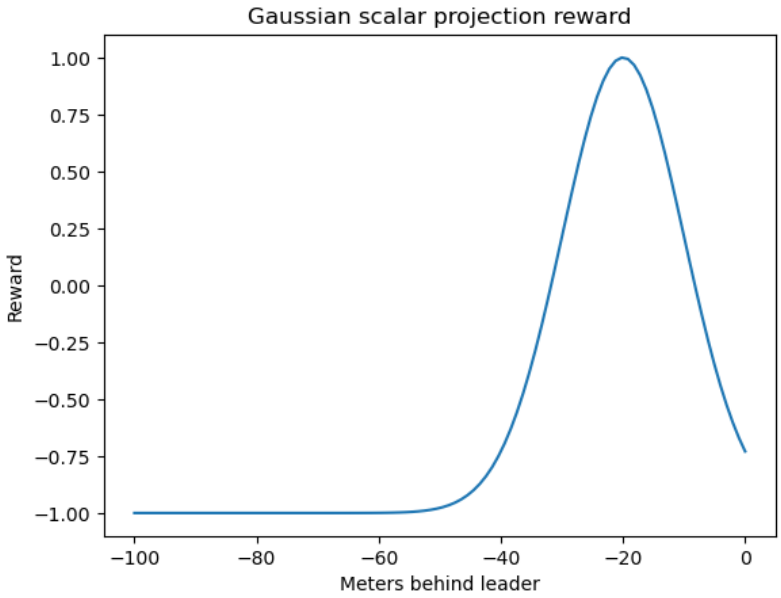

Figure 6.4: Scalar-projection reward for various distances

6.4.2.2 Min-Max-Range Normalization

To scale the returned values form the Gaussian apodization function to the range of -1.0 and +1.0, the min-max-range normalization function is used [77], as defined in Equation 6.3.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{6.3}$$

It scales the given input linearly to a desired range. Since an apodization function is always within the range of 0.0 to 1.0, the min-max-range function can be easily used for this case.

6.4.2.3 Polynomial Normalization

For the angle between the two cars, i.e., between each car's forward vector, the situation is different. The CARLA simulator returns a cosine value when using its embedded function to calculate the angle. Since the cosine saturates around +1 and -1, it defeats the purpose of indicating clearly (i.e., with a steeper slope) how the reward should develop around the metric's ideal value. When feeding the cosine into the Gaussian apodization function, the smoothness around its peak value (i.e., a cosine close to 1.0 indicates close to perfect alignment) is even further increased. Also, when decreasing the standard deviation drastically, there remains little of the function's response value for most of the remaining cosine values. Hence, a different function should be used to provide a steeper slope towards the ideal value.

Ideally, the function has a steep inclination, while also spreading out a bit. A simple polynomial function is actually sufficient for such a case. By tuning the degree of the polynomial function, the slope can be adjusted. After comparing different values, a degree of 20 shows promising features, while also keeping the intermediate values within a reasonable range. The two functions are shown in Figure 6.5.

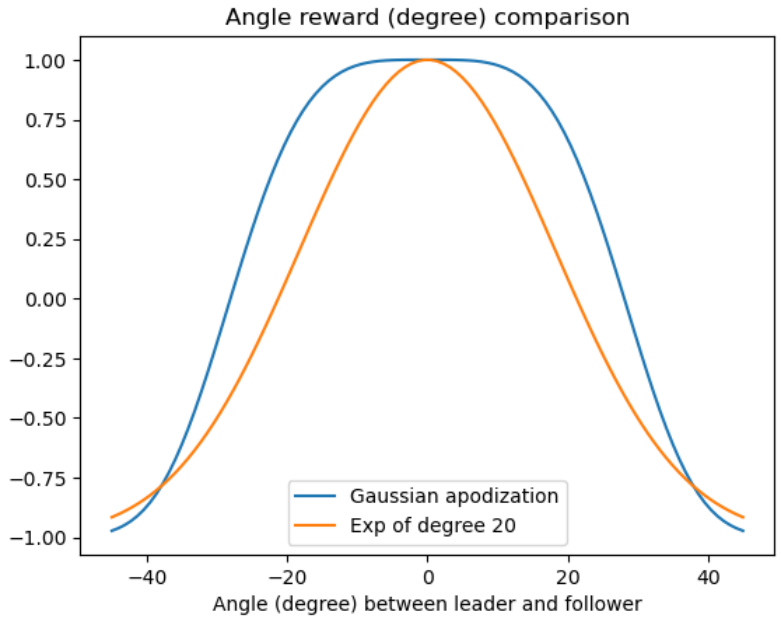

Figure 6.5: Comparison of the polynomial and the Gaussian apodization functions for the angle

With a polynomial function we want the minimum input value (in our case the cosine) to be at 0. Since the cosine has a range of -1 to +1, shifting it by +1 should do the trick. Therefore, the maximum value on which the polynomial is calculated is 2, resulting in $2^{20} = 1,048,576$

as a maximum intermediate value. This maximum value is then used to normalize it to a range of 0 to 1 using the min-max normalization function from Equation 6.3.

With these three continuous reward functions in place, the agent should receive sufficient feedback on its actions to learn efficiently.

6.4.3 Termination

In addition to the reward functions, a negative incentive when colliding with an object should be provided. Also, such a collision should terminate the learning process, since it is an undesired outcome.

Hence, a fixed negative reward of -1 is provided in case of a collision. This reward is designed in a way that reduces previous rewards in that step to -1 in case they were positive before, or adds -1 to an already negative reward. One can think of this behavior like a put option in finance, where the strike is -1 and a linear decrease starts at 0. For example, if the previously collected reward in this step is +1, the resulting reward in case of a collision is -1. However, if the previous reward is -0.5, then -1 is added, resulting in -1.5.

This design should penalize a high risk taking strategy that results in a positive reward, but with a high probability of colliding with an object.

6.5 Architecture

The individual components from the design part are an important foundation for this architecture section. Our architectural representation is in the form of a data and logic flow diagram.

We split the diagram into two parts, mainly to improve the readability. Figure 6.6 represents the prediction part, which ultimately uses the trained CNN to predict the Q-values and, using the learned policy via the Double DQN, choses the best action. Figure 6.7 visualizes the training part, which is only activated once the replay buffer is sufficiently filled.

An episode starts with a reset of the simulator to its initial state. There, the two vehicles are placed at locations so that the leader is in front of the follower car. Also, the whole environment is set up, for example, the follower's camera and lane invasion sensors are deployed.

Once the initialization is complete, the prediction part starts. It begins by capturing the image of the current state, then takes an action according to the epsilon-greedy policy and stores it along with the reward in the replay buffer.

If the replay buffer is sufficiently filled (according to a pre-defined hyperparameter), the training starts. Another hyperparameter, the minibatch size, defines the number of samples that are randomly drawn from the replay buffer. By iterating through each sample, a training run is performed. Hence, in a single episode, the model can be trained as often as the minibatch is in size. This makes the DQN algorithm sample efficient, since it re-uses previous experiences multiple times.

Once the training loop processed all samples from the replay buffer, it starts with the next episode (or step), and takes again an action. If a termination event occurred, for example the follwer collided with an object, then the simulator is reset to its initial state and the process starts over again. The elements that already exist in the replay buffer are kept and new observations are added with each episode. Only when the replay buffer reached its limit are older observations removed (a first-in, first-out principle).

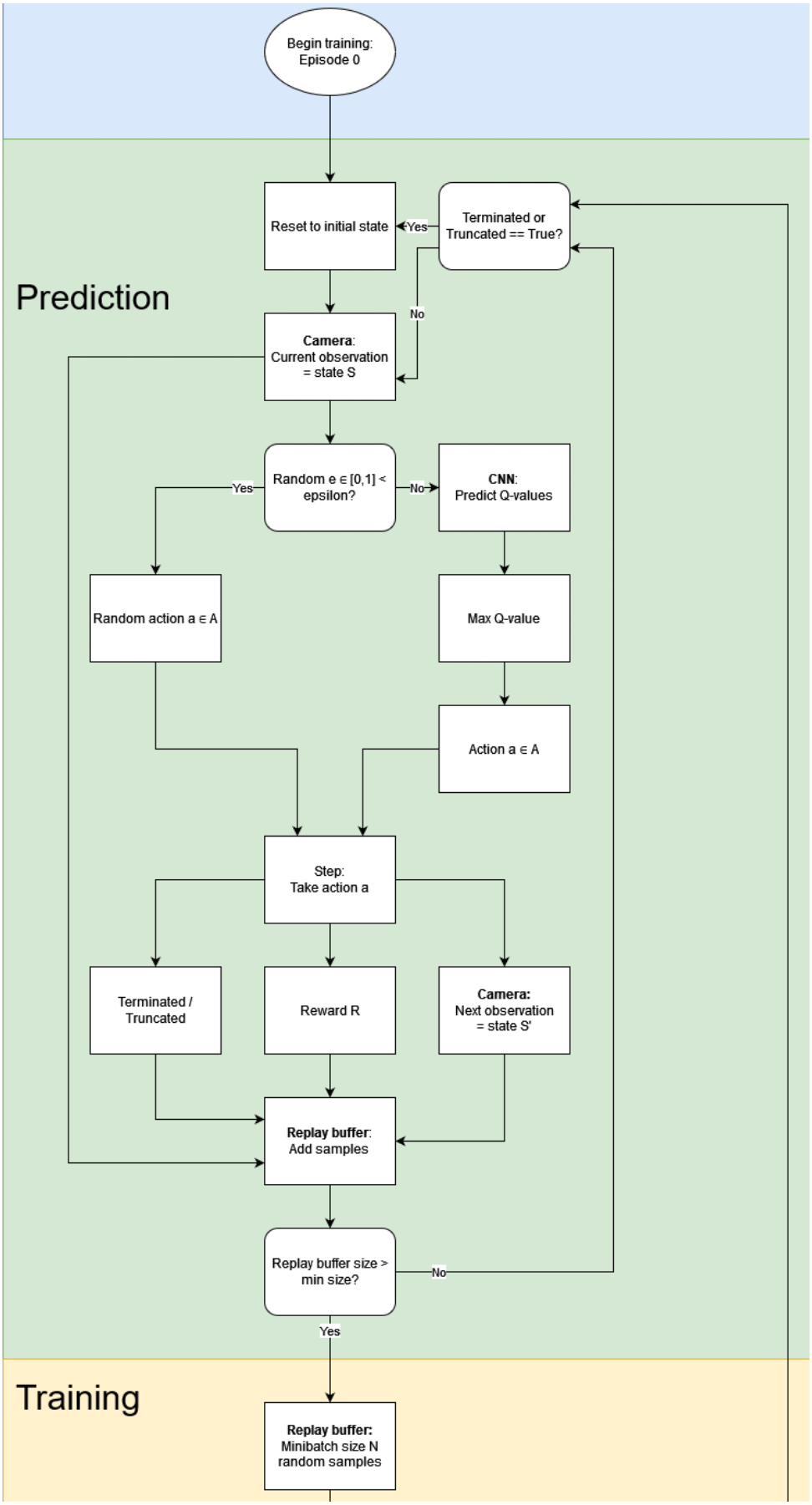

Figure 6.6: The data and logic flow architecture of the simulator and learning (part 1 of 2).

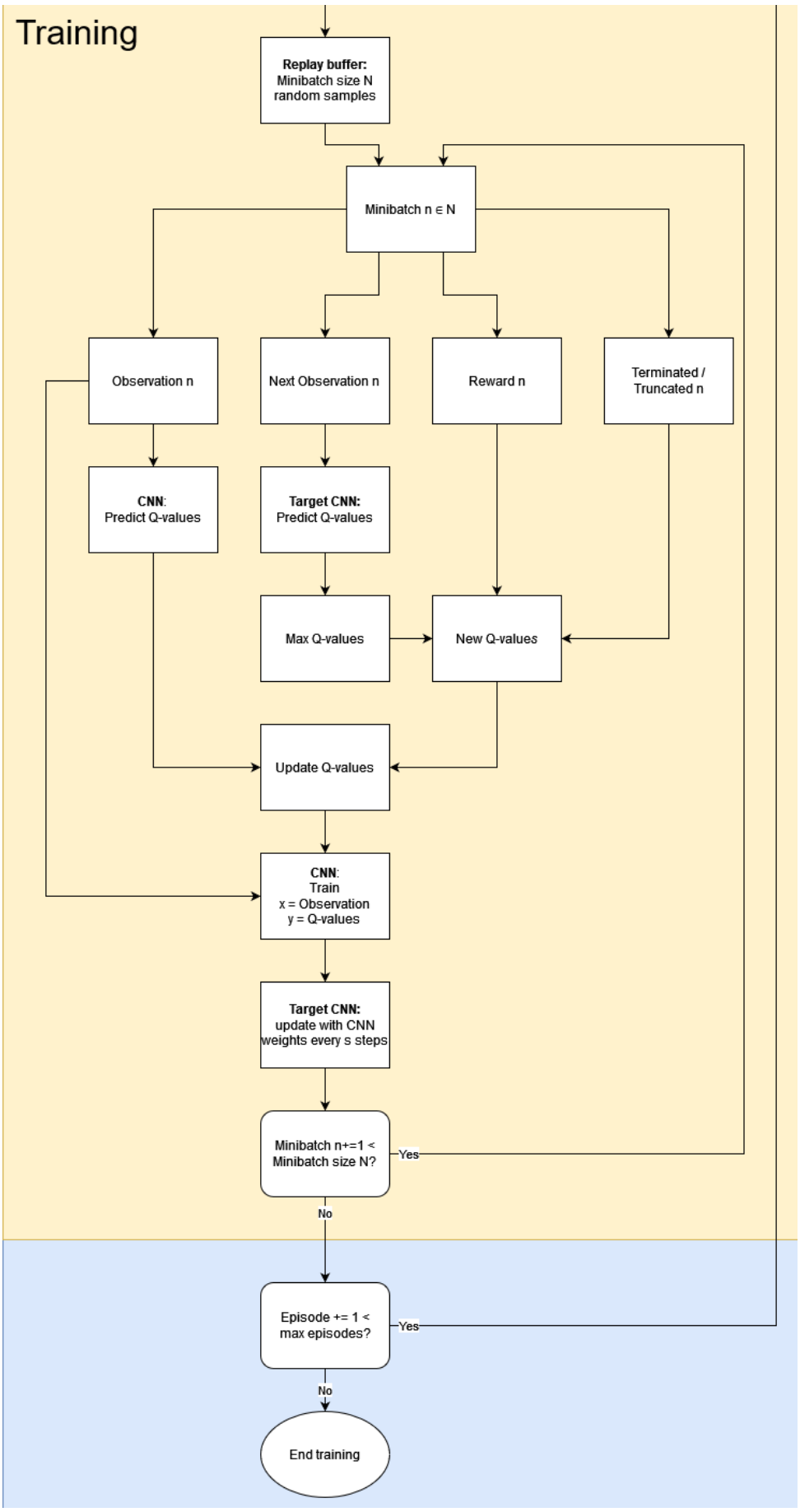

Figure 6.7: The data and logic flow architecture of the simulator and learning (part 2 of 2).

Chapter 7

Implementation

7.1 Setup

In this section, the setup for running the simulation and the reinforcement learning is described.

7.1.1 CARLA

The simulator is the main part of this thesis. While there are a number of simulators that target the autonomous driving space available, as described in section 5.1, we decided to work with the CARLA simulator [22], [21].

It is open-source, and specifically targeted to performing research on autonomously driving vehicles. Based on the Unreal Engine [78] and written in C++, it features realistic-looking 3D scenarios such as cars, pedestrians, towns and rural areas. In addition, they adopt the OpenDRIVE standard [79] that enables the import of custom maps, but also exposes a standard terminology when describing features of a road. Crucially for this project, they expose the simulator's functions through a well-documented Python API, that enables the controlling of much of the simulator's behavior and assets.

CARLA is designed as in a client-server architecture. The simulator is running as a server and opens a port on the local network, whereas the client (e.g., our Python-based reinforcement learning algorithm) accesses the simulator via this port. That design enables multiple clients to access the simulator, but also allows running it on a dedicated server with the clients running on local machines.

Finally, CARLA features a traffic manager, that runs as a separate client, also connecting to the simulator via its port. The traffic manager takes care of managing the vehicles and

traffic lights in the simulator along with the path planning, especially when the autopilot is activated.

CARLA is best run on Ubuntu 18.04 or 20.04 or on Windows 10. A package with all necessary files is provided on their website, and it can be installed and launched locally.

7.1.1.1 CARLA Build from Source

If customizations are planned, such as adding custom maps or changing the buildings in a town, another option is to build CARLA from source. Such an attempt was made, although unsuccessful and ultimately not required. We briefly describe the learnings from that experience.

It took about six hours in total to set everything up and complete all stages including the Unreal build. The build has very specific requirements, such as the Visual Studio version, cmake and make versions, too. The tutorial is quite good, but it lacks specific but important details, like the cmake or make installation path problems that we encountered. There, it is important to not have any spaces in the paths that reference cmake or make, otherwise it will fail. However, another error appeared (a missing separator), which we could not resolve. It could even be that a specific update in one of the libraries made this local build fail.

It may work much better on Ubuntu, however we think there is no longer a hard requirement to have a customizable version for our purpose. The provided maps should be sufficient for our research, and adding custom layouts would be quite labor intensive.

7.1.1.2 RLlib

When selecting CARLA, one feature that stood out was a library to use pre-implemented and tested reinforcement learning algorithms from RLlib [80]. Similar to using Gymnasium, as described in section 7.3, RLlib provides access to distributed training of more than 25 RL algorithms.

Unfortunately, CARLA's RLlib connector library is no longer supported, and even our attempt to get it working again has failed. We therefore implemented our own algorithm as described in chapter 6.

7.1.2 Hardware

From the beginning of this thesis until about ten weeks in, the hardware to train the models and running the CARLA simulator on were two desktop PCs with a dedicated graphics card.

7.1.2.1 Desktop PCs

We have access to two desktop PCs running Ubuntu 20.04 with an NVIDIA GeForce GTX 1060 that has 6GB of memory, along with 32GB RAM. The setup with two desktop PCs works quite well, even though in the beginning the simulations crashed unexpectedly. Resetting the display and running the simulation in a headless mode without any rendering proved to be a good approach, since then the experiments ran more stable, although still with frequent crashes.

Also, training the models on the PC's CPU worked well, even though some attempts were made to use the GPU, as described in subsection 7.1.4.

7.1.2.2 DGX-2

For the last few weeks in this thesis we received access to a high-performance computer that is specifically designed for running machine learning experiments. It is an NVIDIA DGX-2 [81] which is equipped with 16 Tesla V100 graphic cards, each comes with 32GB shared GPU memory and 1.5TB system memory. The 16 GPUs with a total of 512GB of GPU memory combined exceed 2 petaFLOPS (i.e., floating point operations).

The server runs on Red Hat Linux Enterprise and the Apptainer containerization software (see subsection 7.1.3) allows to run model training, but also the CARLA simulator in our case.

7.1.3 Apptainer

Apptainer [82] is the Linux Foundation version of SingularityCE, which is the community version from Sylabs. It is a container system for high-performance computing (HPC) and is deployed on the DGX-2 (see subsubsection 7.1.2.2). It has an improved security and user permission handling in the sense that a container can be created without requiring any root privileges. This allows building containers on systems where a user does not have any elevated privileges, such as OST's DGX-2.

An Apptainer container is best built using a definition file, which is similar to a Docker file. Importantly, containers can be built from existing Docker images that are for example stored on Docker Hub. In the definition file, the contents of a container can be defined.

There are two containers that are required for running experiments in the CARLA simulator:

1. one containing the CARLA distribution,
2. another with the code for the training along with the necessary Python libraries.

The CARLA container is easy to build from a CARLA Docker file The second container for the code for training and accessing the CARLA simulator over its Python API required careful installation of the necessary packages.

7.1.4 GPU Improvements

Getting the simulation and the code running on the DGX-2 is one thing, but optimizing it to utilize the available resources optimally is another important step. The previous versions of the code were running on PCs with a dedicated GPU. However, their resources were taken up mostly by the CARLA simulator. Therefore, the training and predictions happened mainly on the PCs' CPUs. While this code ran quite optimally on these two PCs, the different architecture and more powerful resources of the DGX-2 requires adjustments.

While the prediction method ran about twice as fast than before on the laptop's GPU, the training was about the same. There are a few issues that could prevent efficient training on such a machine.

1. There may be many sequential operations in the script that are more efficiently run on a CPU.
2. The data loading could be a bottleneck where a lot of data is transferred between the GPU and the system memory.
3. The batch size of a single prediction or training (i.e., fit) is too small to fully occupy the GPU.
4. A volatile GPU usage could hint at an inefficient data loading or pre-processing of images.
5. Some operations could be executed on the CPU rather than on the GPU.

The batch size was a bottleneck, since a by increasing the batch size the GPU utilization increased as well. A smaller image input size of 64 by 64 pixels allows to use a four times larger batch size while keeping the memory consumption the same. The benefit is that the GPU is busier processing a larger batch size and hence it increases its utilization. There are still other optimizations that should be considered, such as trying to decrease the calls to the CARLA simulator within runs, or the transfer of stored images from the replay buffer to the prediction and training batch.

7.2 Machine Learning Framework

We describe the main libraries that were considered for the training of the reinforcement learning algorithm in this thesis.

7.2.1 Tensorflow

Tensorflow is a well-known machine learning platform [83]. It was released in 2017 by the Google Brain team in version 1. In 2019, version 2 was release, which is still the major version in use today [84].

We adopted Tensorflow because it abstracts many layers of a machine learning process. It reduces the time to start training a model. Also, many tutorials are based on Tensorflow, especially the one which inspired some of the code for this thesis [85].

7.2.2 PyTorch

PyTorch would have been the alternative to Tensorflow. It is very popular among the research community, especially since many parts of the machine learning process can be controlled.

We tried to switch over to PyTorch at some point during the project, mainly to better control the usage of GPUs. However, it turned out that a significant effort is required to migrate existing code from Tensorflow to PyTorch.

Therefore, we decided not to use PyTorch. On the other hand, Stable-Baselines3 (see subsection 7.3.2) uses PyTorch, and if that migration were made, the switch would be somewhat easier.

7.3 Gymnasium

Gymnasium has become a standard for structuring code and environments in the reinforcement learning space. It provides a standardized API to create and call public and custom local environments for reinforcement learning tasks. An example of a public environment is CartPole [86] where the agent is a cart that should learn to balance a pole only from its position and the pole's angle.

In our case, the environment is the CARLA simulator. The default settings are two cars placed behind each other in Town06.

7.3.1 Environment

A Gymnasium environment consists of at least the following two methods:

- `reset()` - this method sets the environment to its initial state from which a new training run can be started.

- `step()` - once an action for an agent is defined, the step method advances the environment by a single step where the action is executed.

The environment should return the reward from that step along with new state (for example, the next image from the agent's camera, or the agent's new location).

Other methods can be defined, but are optional, such as:

- `render()` - a graphical representation of the environment along with the agent that is learning.
- `close()` - when the training is finished, the close method cleans up auxiliary resources, such as closing open ports or freeing up reserved memory.

Once these methods are implemented, the environment needs to be registered with a suitable name (e.g., `CarlaSim-v0`) in the local Gymnasium registry. With that name the environment can be found by the creation method `make()` and it takes that name as an argument. For example, `env = gymnasium.make('CarlaSim-v0')` would be the appropriate way to create a new instance of the CARLA environment.

7.3.2 Stable-Baselines3

Using the Gymnasium API, it is possible to use reinforcement learning libraries that also adhere to this standard. A well-known open source library is Stable-Baselines3 [87] (or SB3 for short). They emerged from a need to create standardized and peer-reviewed implementations of reinforcement algorithms. Since there are multiple ways to implement, and especially tweak, RL algorithms, SB3 set out to become a de-facto industry standard in defining default implementations. Another benefit is the feedback from the community that helps to improve and fix the algorithms' implementations.

Using the previously created Gymnasium-compatible environment, a SB3 provided RL algorithm, such as a DQN or PPO, can be accessed and used for training a policy. We briefly tested the SB3 connection, and it seemed to work, but ultimately, we did not implement the training in that way since the time-frame that would have been necessary to test this implementation was too long. However, in a future work, SB3 should definitely be integrated as it rules out RL algorithm implementation errors.

7.4 Testing

An important part of every software project is to include a suite of tests. Whether they are unit, integration or end-user tests, it is important to verify that critical assumptions about the functionality of the code hold.

7.4.1 CARLA Tests

We were running manual visual tests in CARLA. This mainly meant testing:

- where and if the spawn locations for vehicles exist,
- do the vehicles behave as expected while driving,
- does the camera input capture the images,
- are segmented input images correctly transformed,
- does the lane invasion sensor register road line crossings correctly.

These are some of the tests that were run, either by writing a Python script or iterating through some tests in a Jupyter notebook. Especially the lane invasion sensor testing was important, since it appeared that not all the line markings are registered correctly. Near intersections there seems to be an absence of metadata on solid line markings. This allows a vehicle to cross such a line and the lane invasion sensor does not register the event. Such a behavior was found in a manual test after the visual inspection of the vehicle behavior when it continued driving off-road whereas the episode should have terminated after a solid line crossing.

7.4.2 Python Unit Tests

A more automated way to run tests is Python's unittest library. It allows to define tests against functions, either as a unit test or even as an integration test where multiple components are evaluated simultaneously.

7.4.2.1 Test Driven Development

In addition, a Test Driven Development (TDD) approach was used to create such tests. Using TDD to design the reward function is an effective way to define the expected behavior of a function in a test and then investigate when the results did not match. It is also much easier to define and test corner cases when creating tests for outlier values. This way we were able to identify mistakes in the reward functions which would have been very difficult to find otherwise.

As an example: a collision should give a negative reward of -1.0. However, if the overall reward is already at e.g. -2.0, should the negative reward be only -1.0 or -2.0 or -3.0? But what about when the overall reward is significantly positive, e.g., +5.0. Should a collision only add a negative reward of -1.0 resulting in this case in an overall reward of +4.0?

TDD actually fores one to think about these scenarios in depth and what the expected outcome should be. Overall, more than 600 test cases were implemented and automatically tested.

Chapter 8

Experiments and Results

In this chapter, various experiments that were conducted as part of this thesis are presented. They include model evaluations, but also GPU improvements.

8.1 Fixed Steering with Improvements

After running many models that incorporated the improvements since the successful experiment of the first vehicle platooning as described in subsection 8.3.2, the results were not quite as impressive again in terms of following the leader vehicle (see for example, section 8.2). While the follower seemed to be capable of identifying the leader vehicle from its camera input and follow it to a certain degree, the steering always felt to be reduced to the very minimum. Also, when facing impossible situations, like being close to an obstacle from which only a sharp left or right turn would have allowed to escape, it simply could not do it due to the constraints of the small steering delta.

Hence, this last experiment of this thesis is leveraging all the improvements, such as frame stacking and frame skipping, Gaussian reward functions, an Atari-style CNN, and the braking action. However, the steering is reverted to the fixed values, as per Table 6.1, that were also used in the first vehicle platooning experiment.

The assumption is that the steering with delta values was too complex to learn and, therefore, steering was avoided whenever possible.

Hypothesis 1. *A fixed steering value allows the follower to better and faster learn to control the vehicle compared to the delta steering value.*

8.1.1 Experiment Setup

The parameters are similar to the ones in the experiment described in section 8.2 and can be found in Figure A.2. The most notable change is the steering value that is again fixed with a ± 0.2 value. Another, probably less relevant hyperparameter that was reverted is the discount value from 0.9 to 0.99, mainly to align it again with the first vehicle platooning experiment.

The image size is 64 by 64 pixels. Although, we did setup a second model training with 128 by 128 pixels, the simulation on that PC crashed too often for unknown reasons and did not produce any usable results. An overview of the most important parameters can be found in Table 8.1.

Table 8.1: Experiment setup for fixed steering with improvements

Steering	Throttle	Braking	Resolution	Reward weights[1]	Discount	Location
± 0.2	0.5	0.25	64x64	P/L: 0.7, S/P: 0.3, angle: 0.0	0.99	60 randomly selected

[1] P/L = point-line distance, S/P = scalar-projection distance.

8.1.2 Analysis

The best result from the long-list evaluation are models 880 and 910, hence they are used for this evaluation (see Figure A.6 for all evaluated runs). (A more detailed description of how an evaluation is performed can be found in the previous experiment in subsection 8.2.1 and subsection 8.2.2).

They are actually close to the performance of the first platooning experiment and can identify and follow the leading vehicle in a relatively stable manner. While the steering is wigglier since both models switch between left and right quite often, they both can control the vehicle much better. In comparison to previous experiments, where the steering was much smoother, although it was rarely done, these two models do steer much more often. The detailed metrics from each model can be found in Figures A.17 to A.19 and Figures A.20 to A.22, respectively.

The assumption defined in Hypothesis 1 seems to hold true in that the steering delta was too complex to be learned, at least in the time frame that was provided, and hence, a simplification proved to be a good decision.

It is noteworthy, that the other improvements, frame stacking and skipping, Gaussian reward function, smaller image input and the Atari-style CNN seem to work as expected.

8.2 Throttle-Braking Comparison

The question which should be investigated in this experiment originates from the observation that the way a car accelerates and brakes influences its behavior. If the acceleration is high and the car therefore gains speed quickly, it is more difficult to control. On the other hand, a slower acceleration could increase the control over the vehicle and reduce the stop-and-go behavior. But, following may be more challenging with a lower acceleration since catching up with the leader vehicle is delayed.

Hypothesis 2. *A higher value for acceleration (i.e., throttle) and slowing down (i.e., braking) leads to better following behavior at various speeds due to better control of the speed.*

8.2.1 Experiment Setup

For this experiment, the latest available improvements, like the Atari-style CNN (see: section 6.3) along with the frame stacking and skipping (see: subsubsection 8.3.2.3) is used. It is trained with the most recent version of the reward function (see: subsection 6.4.1) and the weights are set as per the experiment in subsection 8.3.7. It uses the Gymnasium wrapper (see: section 7.3) but without calling the Gymnasium library directly. Instead, the class is directly called in the script that combines the environment, the agent (i.e., the model), and the statistics collection.

Since the main focus of this experiment is on the behavior of accelerating and braking, we decided to use the same location for the training instead of selecting randomly many different spawn points. This has the advantage that time until first useful results appear is reduced, but also the issue with missing solid line markings in the maps is not as widespread as in other locations in this map (see subsection 7.4.1 with observations around lane markings from running tests in CARLA). To compensate for the reduced experience bias, the traffic manager was set in a way so that the leader demonstrates random behavior in terms of lane changes and making turns.

The most important parameters for this models are listed in Table 8.2. The other parameters were set as usual and can be found in Table A.2.

In a first step after the model was trained for close to 5,000 episodes, a long-list with the a few dozen of the most recent model snapshots was created. An example of the long-list along with commentary is shown in Table A.3. From that long-list, the criteria to pick a model for further evaluation are:

- if the follower car moves at all,
- if the follower does not crash early,

Table 8.2: Experiment setup for throttle-braking comparison

Config	Throttle	Braking	Resolution	Reward weights[1]	Discount	Location[2]
A	0.5	0.25	64x64	P/L: 0.7, S/P: 0.3, angle: 0.0	0.9	L: 7, F: 2
B	0.3	0.10	64x64	P/L: 0.7, S/P: 0.3, angle: 0.0	0.9	L: 7, F: 2

[1] P/L = point-line distance, S/P = scalar-projection distance.
[2] L = Leader, F = Follower.

- if it shows some signs of following,
- if the overall reward is in the top half.

In this case, experiments with different leader speeds, along with a comparison of the models and an attempt to look for generalization features are performed. Model 1040 was picked from configuration A (see Table 8.2 for the configurations) and model 870 from configuration B.

For the evaluation multiple runs are conducted. Each run continues as long as one of the following conditions is not met:

- Follower-leader distance greater than 100 meters,
- Follower collides with an object.

Neither of these conditions return a high negative reward since it would distort the charts and make the interpretation of smaller, but almost indistinguishable values difficult.

8.2.2 Different Leader Speeds

The leader speeds were varied for both configurations with values between 10 to 30 km/h. Table 8.3 presents an overview of the observations. It is interesting to note that the model 870 with configuration B (i.e., trained with throttle of 0.3 and braking of 0.1) was performing best in terms of rewards when run with the configuration for A (i.e., run with throttle of 0.5 and braking of 0.25). The model 870 when using its intended configuration B performed only well at low speeds, where it was also trained on. The model 1040 was doing best at higher speeds, since its throttle value is higher, meaning that it is used to fewer, but higher acceleration.

Table 8.3: Different leader speeds

ID	Config	Speed	LC[1]	Visual Observation	Reward
870	A[2]	10 km/h	No	Stops after a few meters driving straight.	0.5735
870	A[2]	20 km/h	No	Starts to follow for a few meters, then slows down and loses leader.	0.4331
870	A[2]	30 km/h	No	Follows quite closely, then suddenly turns slightly left, slows down, crosses solid line and crashes into an object.	0.3252
870	B	10 km/h	No	Stops after a few meters driving straight.	0.5550
870	B	20 km/h	No	Starts to follow for a few meters, then slows down and stops as it lost sight of the leader.	0.3257
870	B	30 km/h	No	Starts to follow, then slows down and comes to a halt as it lost the leader.	0.2991
1040	A	10 km/h	No	Follows leader closely, but too fast and crashes into it.	0.1878
1040	A	20 km/h	No	Follows leader closely, turns slightly right, stops before a solid line.	-0.2418
1040	A	30 km/h	No	Follows leader closely, turns slightly right, loses sight of leader and stops.	0.2479

[1] LC = Lane Change; it indicates the direction of the leader doing a lane change.
[2] Configuration was inadvertently set to A for evaluation of B.

8.2.2.1 Analysis

The model with ID 870 but with the wrong configuration A applied when evaluated has the most interesting observations. We can verify the visually observed behavior in the statistics overview. For this evaluation, the leader car is set to drive only straight in order to have the same comparison but for different leader speeds.

In the first chart of Figure 8.1 we notice that when the leader drives faster the follower does more steps where it accelerates. This can be seen in the *straight* values, which is the action where the acceleration is applied along with the previous steering value. Also, compared to lower speeds, the number of actions where the follower brakes is lower.

In addition, the situation at 30 km/h where the follower turns slightly left can be observed in the second chart between steps 150 and 200 where a few left turn actions appear. This can be confirmed in the third chart where the steering value is plotted. A negative value means that the wheels are turned to the left (and vice versa a positive value implies a right turn).

In Figure 8.2 the metrics that influence the rewards are shown. The grey area indicates where positive rewards are given with the dotted black line representing the highest re-

ward. The total reward is a weighted sum of the different features, for example, the scalar-projected distance contributes 30% to the total reward.

Various behaviors of the two cars can be derived from these charts. Using the 30 km/h run, we can see that the angle between the cars is increasing when the follower starts to turn at about step 150. Since the leader is driving straight in this experiment we can infer that the angle between the two cars is increasing due to the follower turning. However, this is not reflected in the total reward because the angle does not contribute to it.

Finally, in Figure 8.3 additional metrics are plotted. In the first chart we can observe the leader's speed set to the different fixed values at 10, 20 and 30 km/h, respectively. Also, the left turning action of the follower at step 150 and speed 30 km/h leads to a few broken (or dotted) line markings were crossed. Eventually, just before crashing into an object on the side of the road, two solid lines were crossed by the follower.

As a comparison, the results of the other two models, 870 with configuration B and 1040 with configuration A, can be found in Figures A.8 to A.10 and Figures A.11 to A.13.

8.2.3 Different Leader Directions

A second evaluation was conducted where the leader switched to the left or right lane shortly after the start, or stayed in the center lane (similar to the previous experiment). This evaluation should demonstrate whether the follower can adapt to the leader's behavior and follow correctly.

8.2.3.1 Analysis

From the visual observation overview in Table 8.4, both models have the relative highest reward when the leader drives straight. This could be due to the fact that they are perfectly aligned in the beginning, and as soon as the leader turns left or right, the point-line distance increases. Since it has a contribution of 70% to the overall reward, it reduces it significantly, but it should also provide the follower enough incentive to reduce the distance again.

When looking at the third chart in Figure 8.4 we notice the leader's steering when turning left or right. This is also reflected in the angle plot in Figure 8.5 where the initial spike is due to the leader turning left or right.

As observed with model 1040 when the leader turns left, the follower avoids a collision when it catches up with the leader. In the Euclidean distance chart this situation is reflected by the green line around steps 80 to 100 where the cars are too close. Also in the *Km/h* chart in Figure 8.6 the green line, representing the left-lane change scenario, indicates the follower increasing the speed to catch up.

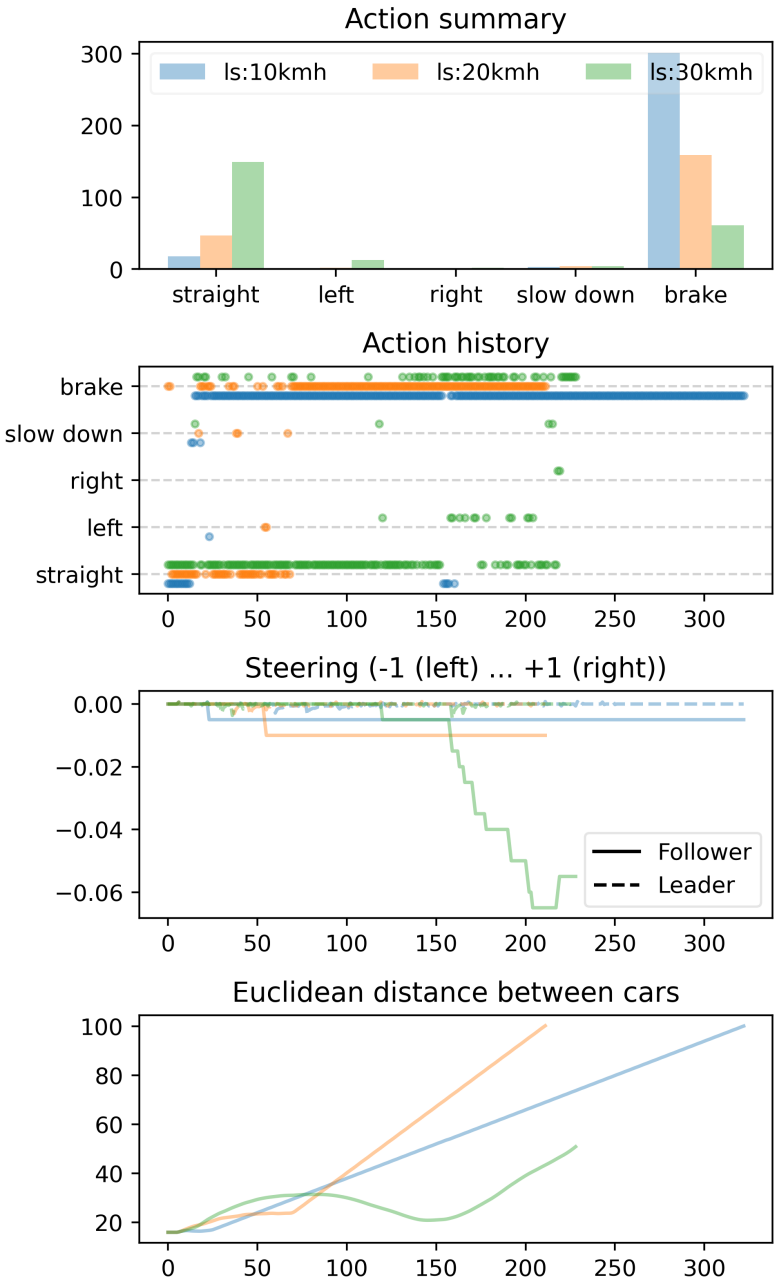

Figure 8.1: Model ID 870 config A - different leader speeds experiment (part 1 of 3).

For completion, the results for model 870 with configuration B can be found in Figures A.14 to A.16.

8.2.3.2 Direct Model Comparison

We now want to directly compare the two models, 870 with configuration B and 1040 with configuration A, respectively. The setup is at the same location as where the training took

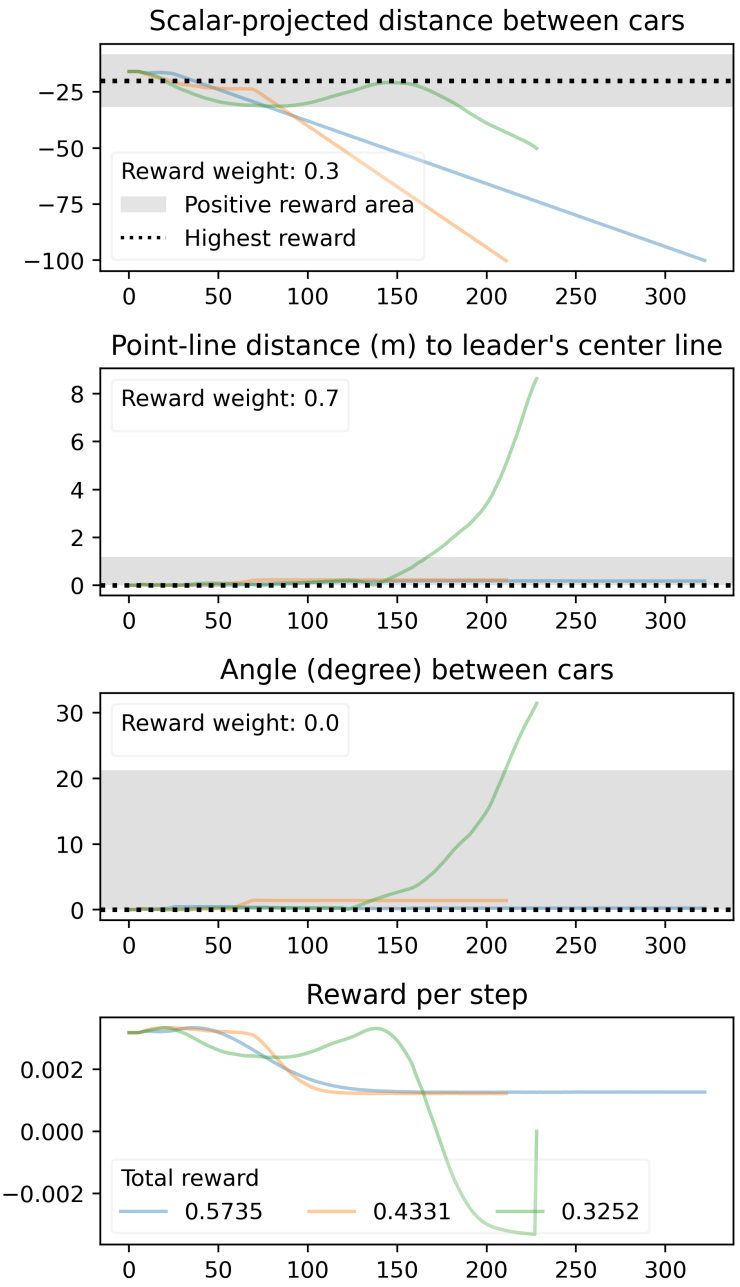

Figure 8.2: Model ID 870 config A - different leader speeds experiment (part 2 of 3).

place with the leader driving straight ahead at 20 km/h. Both models collected the highest reward at that setting.

Even though the model with ID 1040 and the higher acceleration can keep up with the leader for a longer period of time, it receives a lower total reward than the model that terminates early due to the leader-follower distance going above 100 meters. Up until about 100 steps, the model with ID 1040 performed better than the other with ID 870, but then the rewards per step became negative.

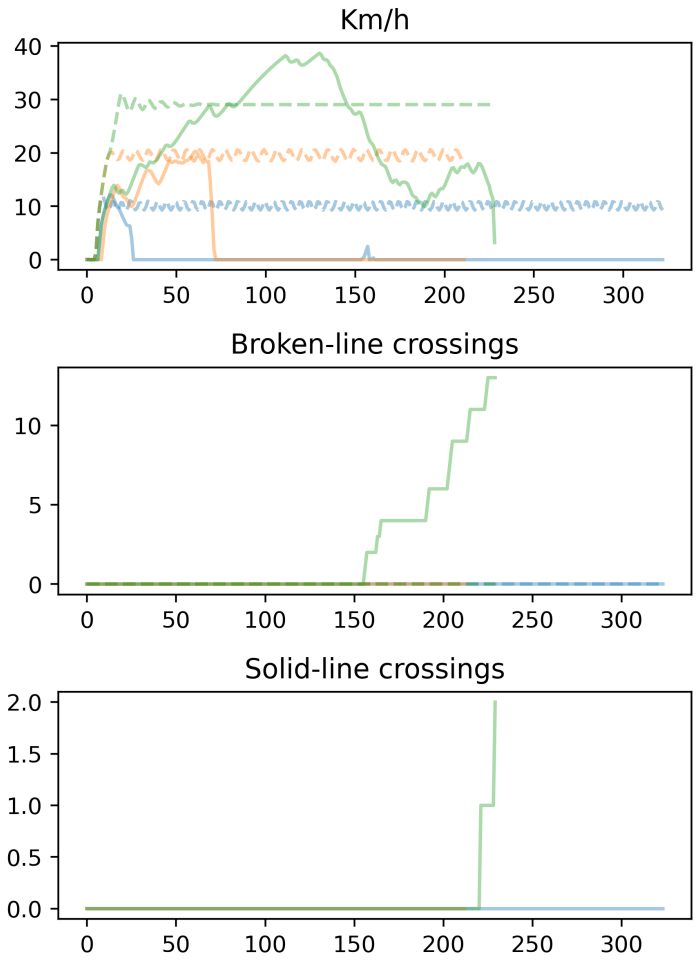

Figure 8.3: Model ID 870 config A - different leader speeds experiment (part 3 of 3).

One reason is that the model with ID 1040 steered to the right, which is indicated in the second and third charts of Figure 8.7. An additional cause for the drop in performance is can be observed in the second chart in Figure 8.8 where the point-line distance between these two cars is in the negative reward area. The follower turned slightly to the right and continued to drive close to the leader, but only at the end turned it back, but then it was too late to compensate for it.

The follower's speed around this phase was also above the leader's, which almost led to a collision. Again, this is evidenced in the first chart in Figure 8.9 where the speed tops leader's 20 km/h.

Table 8.4: Different leader locations

ID	Config	Speed	LC[1]	Visual Observation	Reward
870	B	20 km/h	No	Stops after a few meters driving straight.	0.3145
870	B	20 km/h	Right	Stops after a few meters driving straight.	-0.3872
870	B	20 km/h	Left	Starts to follow, but loses the leader after a while.	-0.0273
1040	A	20 km/h	No	Follows leader initially, turns slightly right, then corrects to the left, but stops above a solid line.	-0.2585
1040	A	20 km/h	Right	Follows leader closely, even catches up, but does not change to the right when the leader does, but to the left and stops.	-0.6138
1040	A	20 km/h	Left	Follows leader to the left, but then continues straight, avoids a crash and stops above a solid line	-0.5596

[1] LC = Lane Change; it indicates the direction of the leader doing a lane change.

8.2.4 Unseen Location - Generalization

The last step is to compare both models using the same settings from the direct comparison, but at a location that was never visited by any of the two models during their training. This should give some indication how well the trained models generalize.

There exist many locations in Town06 that were never visited by the two vehicles. One of these locations is at position 410 for the leader and 111 for the follower in Town06. A special feature of this location is the left curve which the leader is on and follows. For the follower there are some unseen markings on the road where the lane diverts. In addition, the starting position has the leader set slightly left to the follower, hence they are not perfectly aligned as in the training setting. An impression of the setting can be seen in Figure 8.10.

We set the throttle to 0.5, the brake to 0.25 (i.e., configuration A) and the leader speed to 20 km/h. Mainly because the model with ID 1040 was trained on this and also because there were surprisingly good results with model ID 870 that was trained on 0.3 and 0.1, respectively.

The resulting observations are noted in Table 8.5.

8.2.4.1 Analysis

It seems that again, the model with ID 870 but configuration A is performing better than the other model with ID 1040. Even though both models have previously indicated a behavior

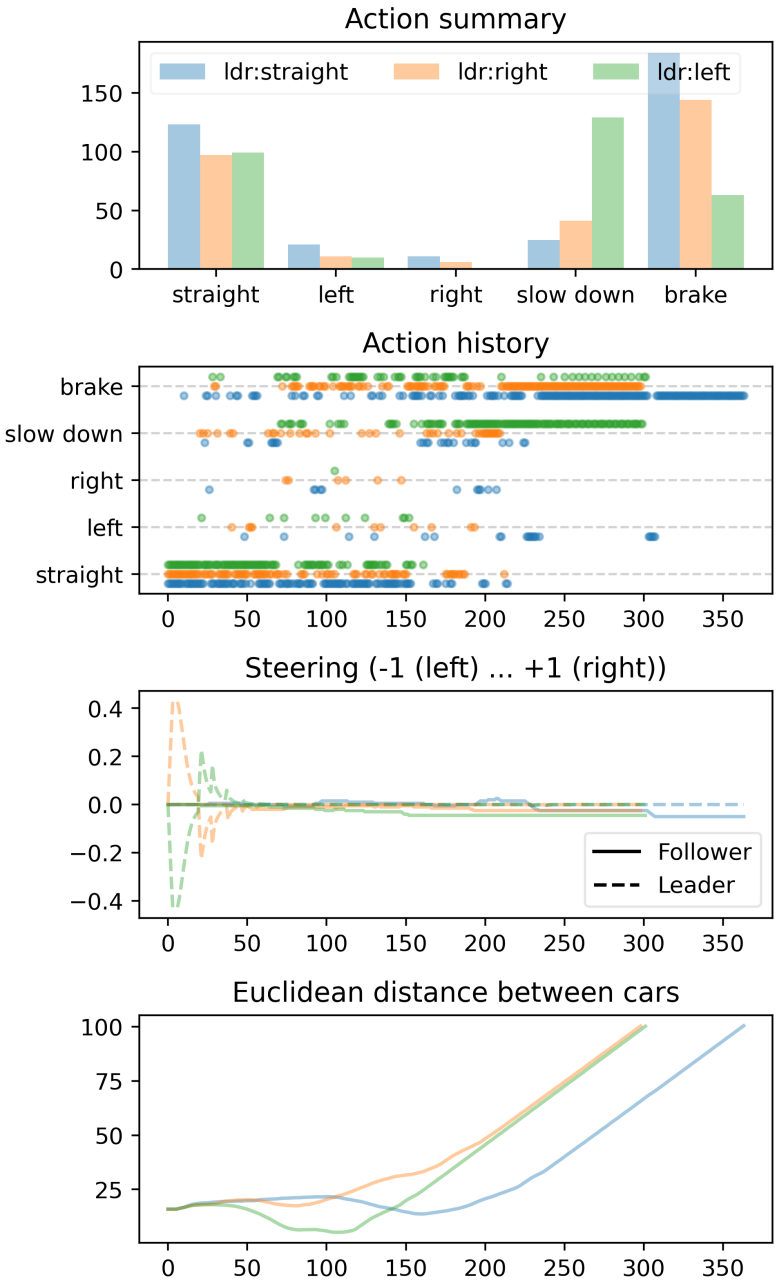

Figure 8.4: Model ID 1040 config A - different leader locations experiment (part 1 of 3).

that could follow the leader when it is slightly to the left, both models did not succeed in following the leading vehicle properly.

When we look at the first chart in Figure 8.11 it is obvious that no enough *left* actions were taken. The model with ID 870 took a few more left actions, which can be also be noticed in the second chart, especially early on which would be crucial to be able to follow the leader. However, both models eventually resorted to the braking action.

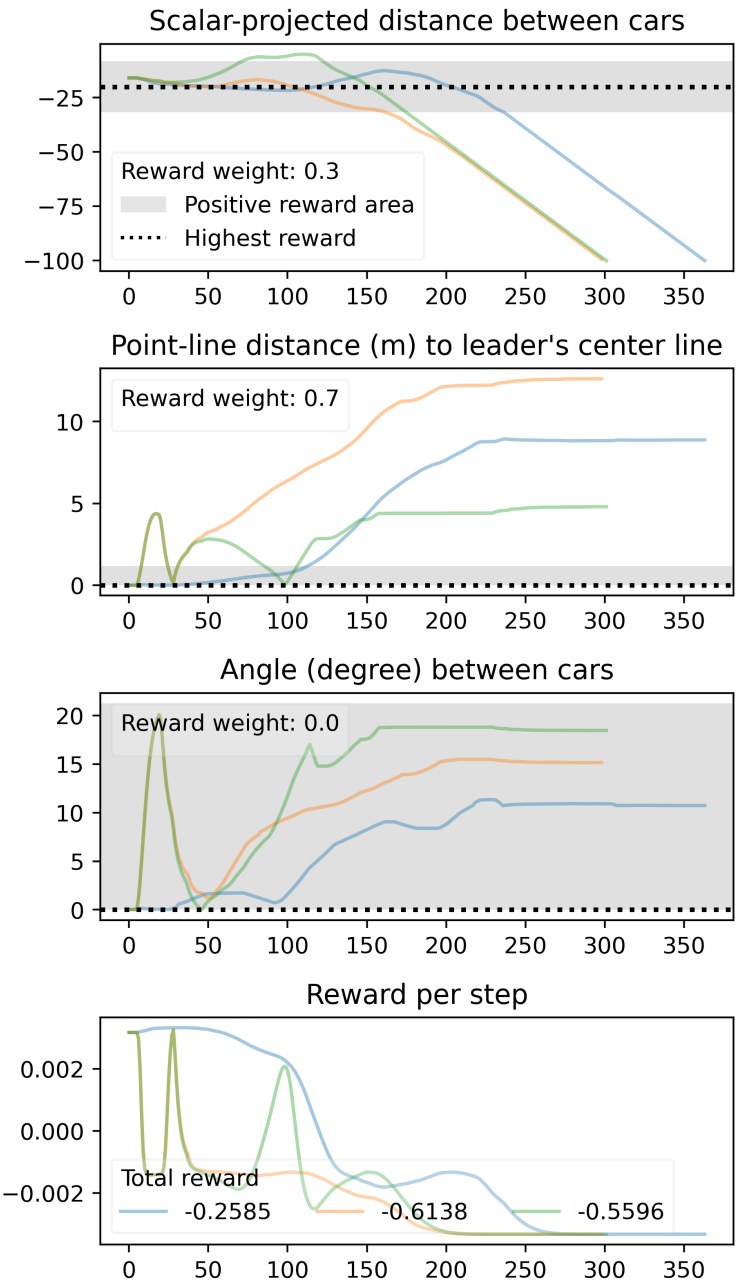

Figure 8.5: Model ID 1040 config A - different leader locations experiment (part 2 of 3).

The steps 100 to 150 in the first plot of Figure 8.12 may be surprising since the distance is zero and even positive. This would indicate that the follower overtook the leader. However, in this case the leader followed the left lane which makes a U-turn and at some point was driving in the opposite direction from the follower. The third chart that tracks the angle between the cars confirms this behavior.

Even though the follower crossed a solid line, it was not registered as there is no increase visible in the last chart in Figure 8.13. Some solid lines in the maps in CARLA, especially

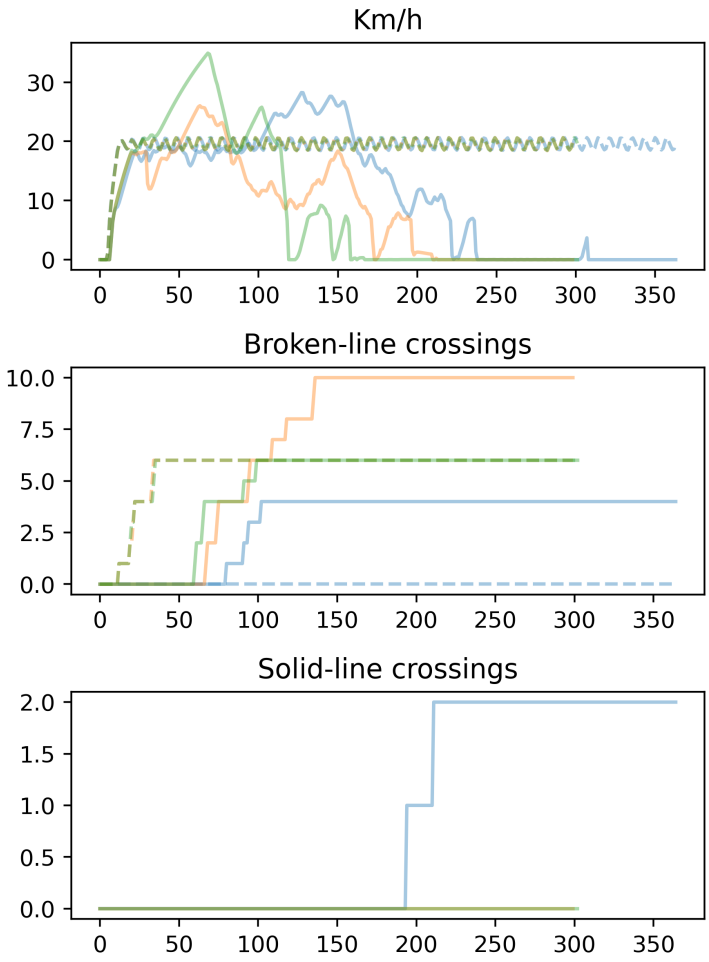

Figure 8.6: Model ID 1040 config A - different leader locations experiment (part 3 of 3).

Table 8.5: Unseen location at 20 km/h

ID	Config	Speed	LC[1]	Visual Observation	Reward
870	A	20 km/h	No	Starts to follow, even turns slightly left in the direction of the leader, but it does not turn enough and stops	-0.6583
1040	A	20 km/h	No	Starts driving straight ahead even though the leader is on the left side, then it loses leader and stops.	-0.7937

[1] LC = Lane Change; it indicates the direction of the leader doing a lane change.

around intersections and exits are not properly registered as such. We do not know if this is intentional or a flaw, however, this may affect the learning somewhat since at some

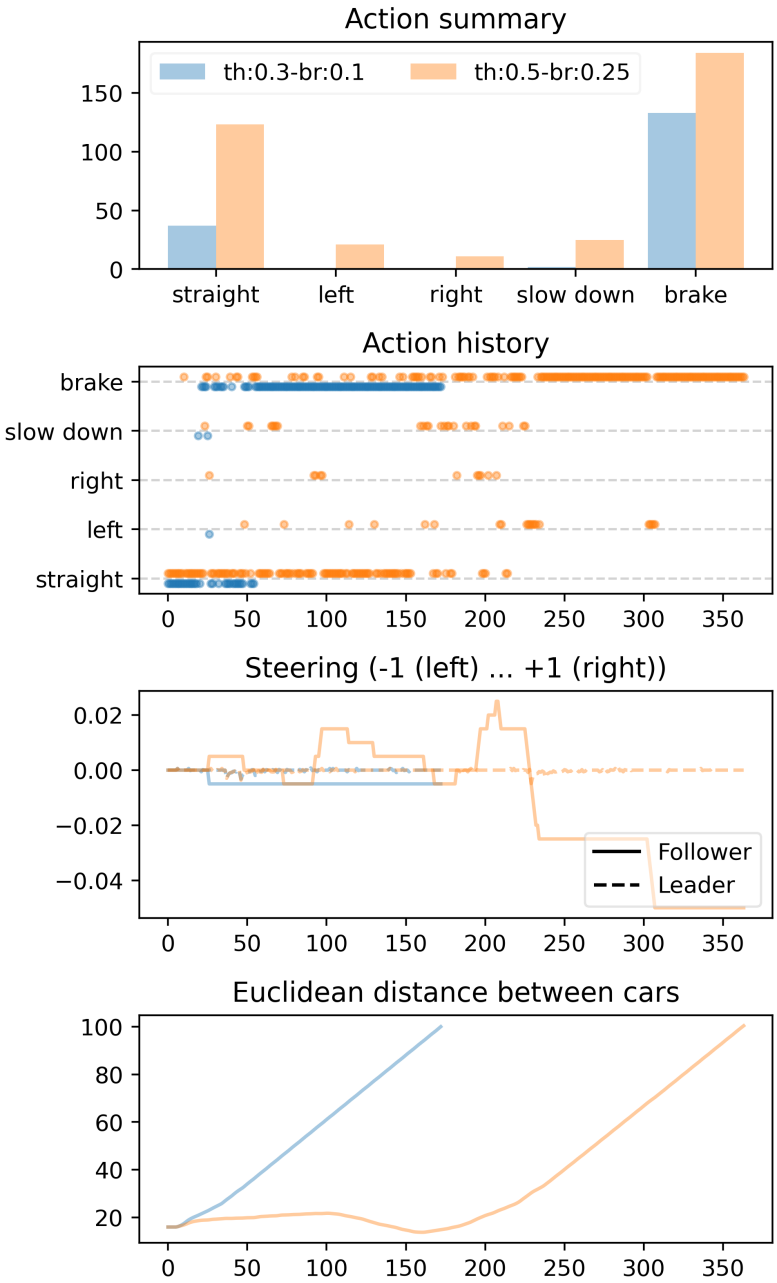

Figure 8.7: Model IDs 870 (blue - config B) and 1040 (orange - config A) at 20 km/h leader speed (part 1 of 3).

locations there is no negative reward when crossing a solid line. It seems that at this U-turn this may be the case.

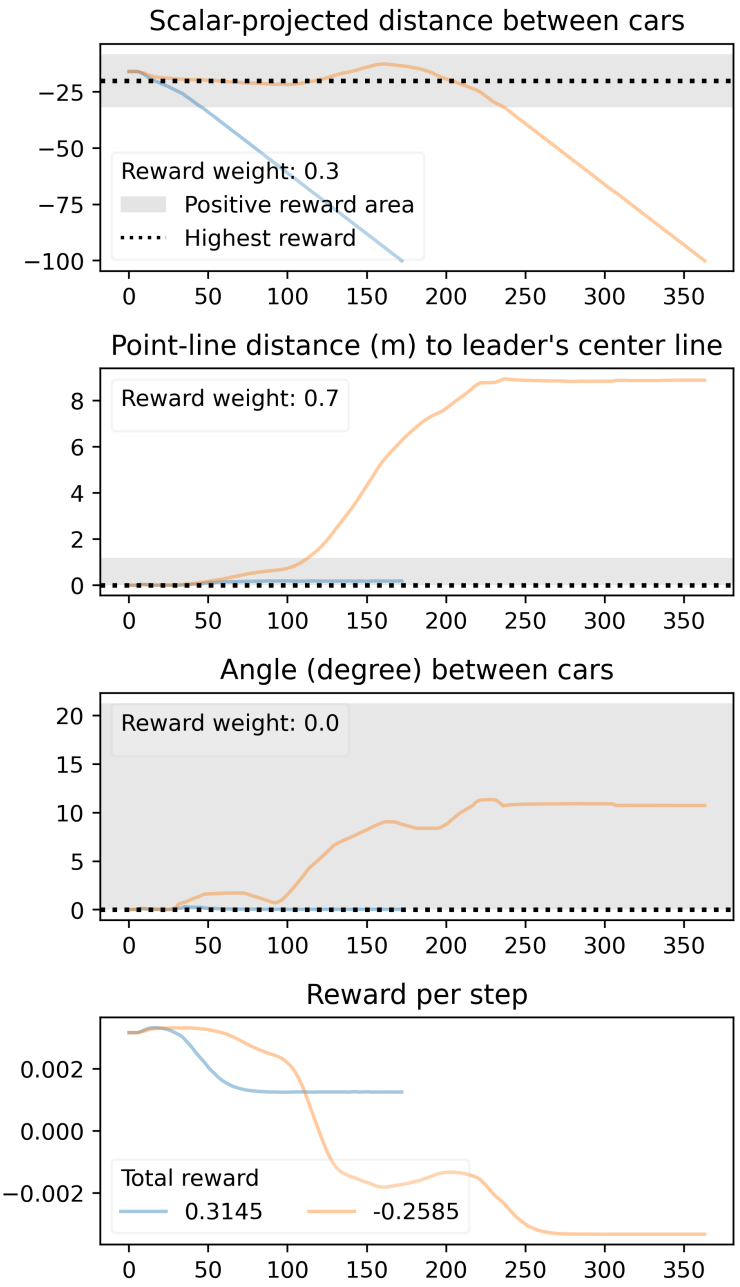

Figure 8.8: Model IDs 870 (blue - config B) and 1040 (orange - config A) at 20 km/h leader speed (part 2 of 3).

8.2.5 Assessment

The assessment of the evaluation for the throttle-braking comparison can be split into two parts.

Throttle-Braking The values for throttle and braking seem to perform optimally when set to the higher values of 0.5 and 0.25, respectively. However, a surprisingly good result from

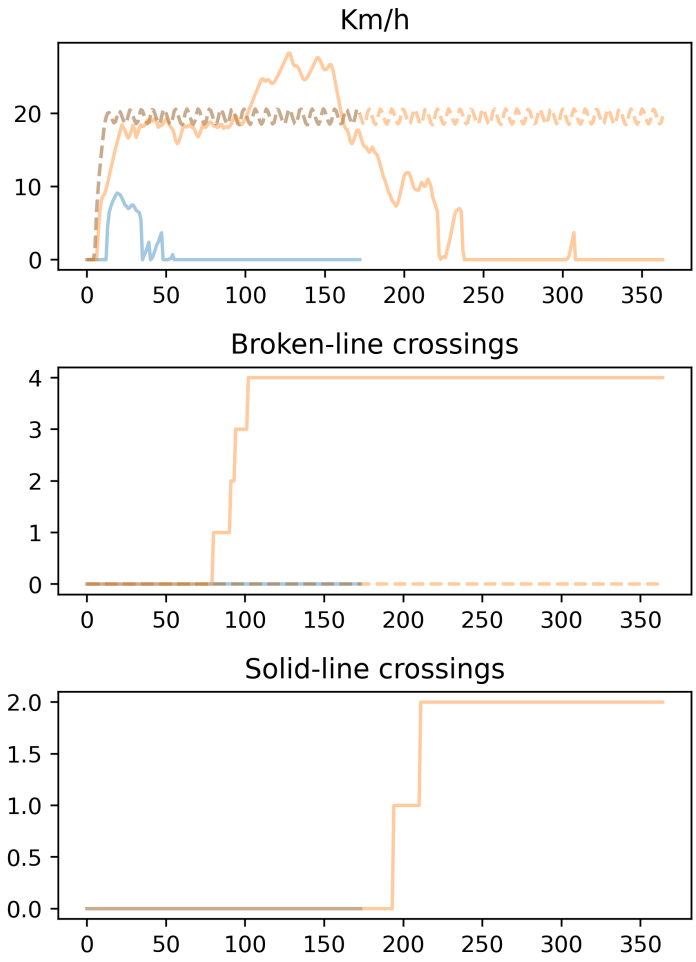

Figure 8.9: Model IDs 870 (blue - config B) and 1040 (orange - config A) at 20 km/h leader speed (part 3 of 3).

the model that was trained on the lower values of 0.3 and 0.1 but which was evaluated with the higher values indicated that it can adapt to different settings. The correct behavior of following and keeping the distance seems to be embedded in the model's parameters, independently of the throttle and braking values it is applied to. Nonetheless, the result should not be overestimated, since only a small subset of possible values was evaluated.

Model Performance Even though the models showed some signs of following behavior, all of the evaluations did not exceed the expectations of sustained vehicle platooning. While repeated evaluations with the same settings indicated that the models do behave very similarly, and hence act not just in a random fashion, there needs to be further tuning in the area of rewards and steering, potentially even adjusting the camera input resolution.

Figure 8.10: The starting position at the previously unseen location 410:111.

8.2.5.1 Analysis

Therefore, as defined in Hypothesis 2, the higher values for steering and braking do lead to better following behavior, but so does a model that was trained with lower values but then run with the higher values.

8.2.5.2 Enhancements

Steering It seems that the steering is quite difficult since the follower often choses to drive straight, brake or slow down. Steering left and right seems to divert the vehicle too quickly from the ideal line, or it may be difficult to get back to the center line.

In previous models, where the steering was just a fixed value rather than a delta, setting the wheels straight was just a matter of taking that action. Performing an abrupt steering maneuver, such as when changing lanes, is not possible with a fixed small steering delta. While it increases the driving stability, the maneuverability is limited.

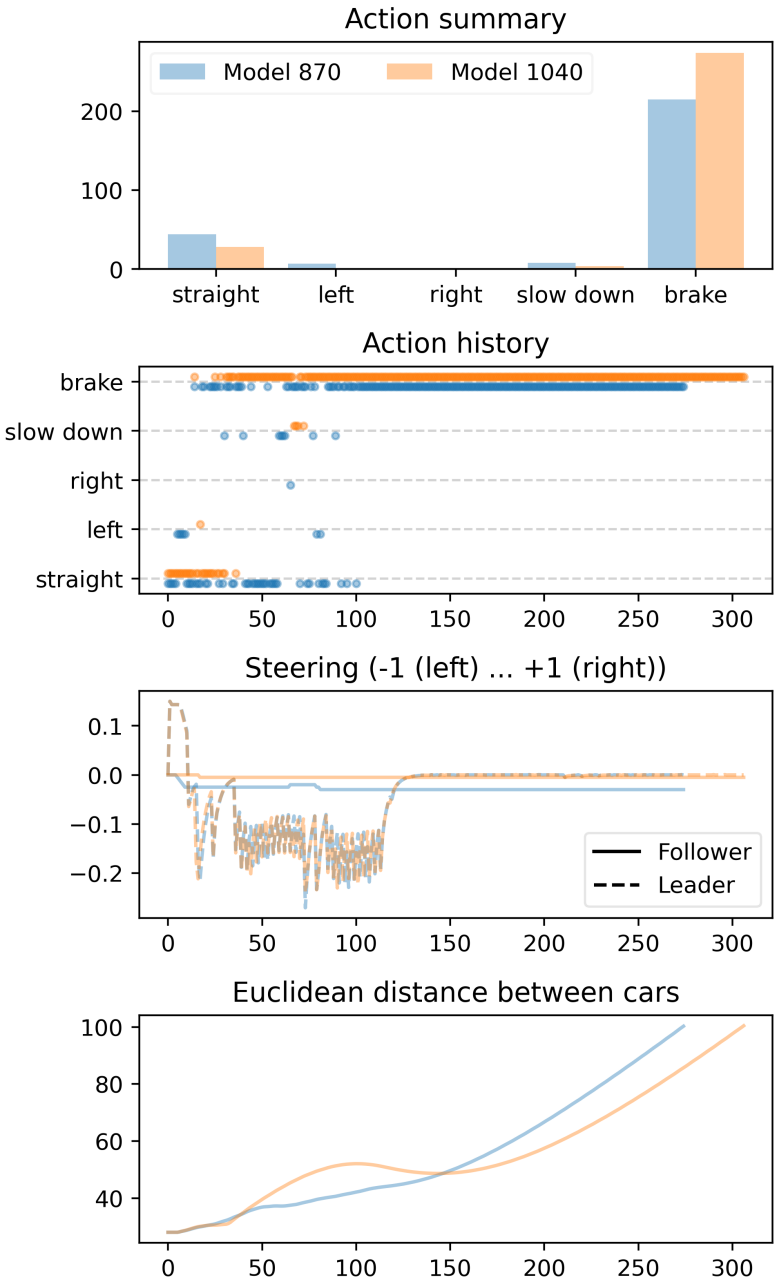

Figure 8.11: Unseen location at 20 km/h leader speed (part 1 of 3).

Ideally, a continuous steering value based on a distribution that takes the current steering angle into account would be implemented. However, this results in added complexity and needs to be tested extensively.

Rewards The reward function may need to be fine-tuned to provide better indications of where the rewards increase. Especially the point-line distance drops off quite early and

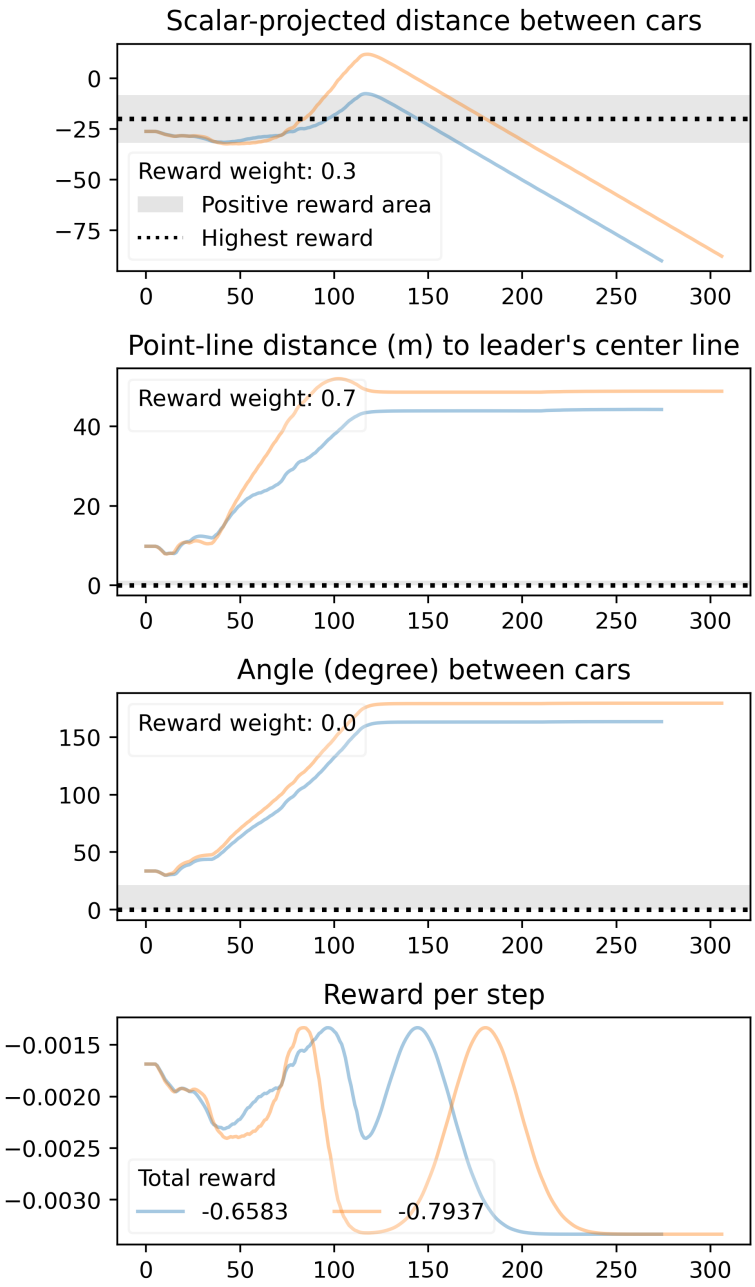

Figure 8.12: Unseen location at 20 km/h leader speed (part 2 of 3).

the positive reward area may need to be extended so that the signal of changing rewards could picked up even from further away.

Camera The camera's input resolution of 64 by 64 pixels does seem to work up to a certain distance. However, if the leader is further away then the follower's camera is not able to identify the leader's position. An increase in the resolution to 128x128 or higher allows for better tracking, however, the training time increases due to the tradeoff between

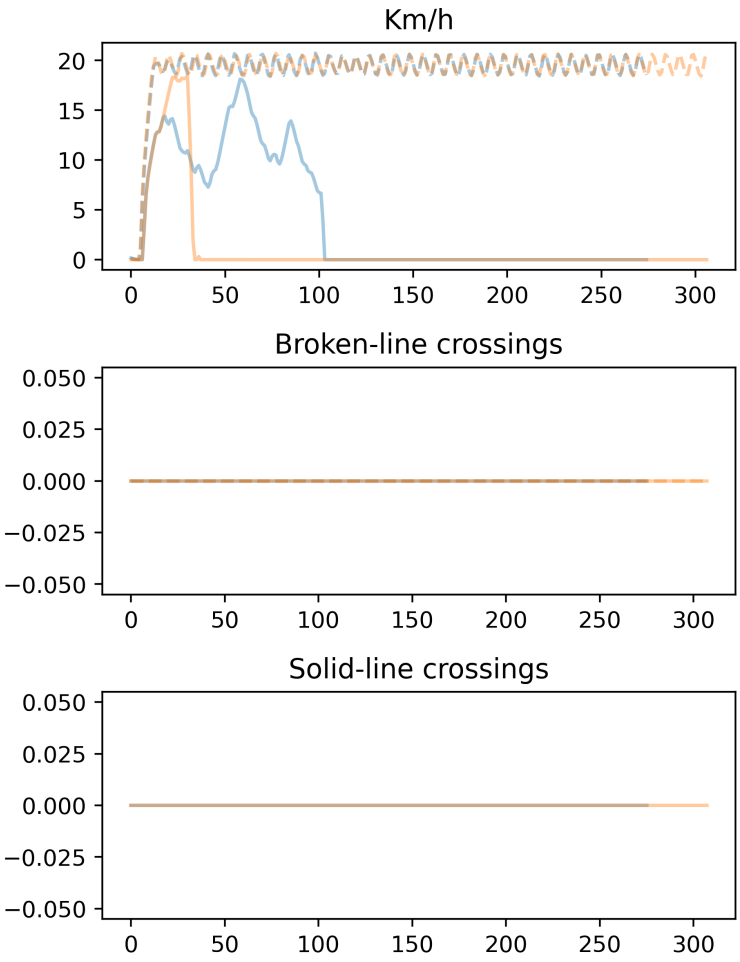

Figure 8.13: Unseen location at 20 km/h leader speed (part 3 of 3).

memory and batch size. A higher resolution requires more memory for a single batch, hence the GPU's memory limit is exhausted much faster. For example, a doubling of the resolution quadruples the size of the image. Using a better GPU with more memory and a taking a longer training time into account would offset such an enhancement.

8.3 Previous Experiments

The fixed steering and throttle-braking experiment were done using the most recent improvements in the reward function along with frame stacking to train their models. However, in previous iterations, earlier models were analyzed to assess the impact of hyperparameter or other changes. In this section, these experiments are described in ascending order so that later descriptions build on previous experiments.

Most notably is the first vehicle platooning experiment, described in subsection 8.3.2, which generated a model that is capable of identifying (or extracting the pixels of) the leader vehicle, imitate its actions and follow it for an extended period of time. It even generalizes in the sense that the platoon ventured into areas on that map which were not visited during training. Also, it crosses intersections successfully and made two left turns in left-bent curves.

8.3.1 CARLA Simulator Experiments

The milestone 2 goal is to train a single agent (or car) to follow a straight lane using reinforcement learning. The challenge is to implement an RL algorithm in a way that it can communicate with the CARLA server, collect samples from the environment and use these for the training.

8.3.1.1 Online Tutorials

There are a few publicly available implementations that use RL with CARLA, however, almost all of them were quite outdated. The problem with older implementations is less the logic, which remains roughly the same, but the outdated libraries that were used. Most of them use Tensorflow 1, which is now in its major version 2 that had many breaking changes. While an old version of Tensorflow can usually be installed, the problem is with the dependencies of other packages, which are sometimes no longer compatible or just throw strange errors. However, one tutorial that we found had very hands-on instructions that allowed to implement a simple DQN algorithm from scratch and connect it to the CARLA simulator [85].

The completion of this tutorial took a few hours and it resulted in a working implementation of a Double DQN algorithm that was trained using the CARLA simulator. The convolutional neural network (CNN) for the image processing is a Keras Xception model [88], however, it has almost a hundred millions of parameters and is probably too complex for this task. This CNN was later changed to the InceptionV3 model, which should be a bit smaller, however it still had around 22 million parameters to train. This is too much for this use case and the available computing power and therefore, the next milestone has the task of reducing the convolution part. Also, the next iteration should include frame stacking so that the model has a sense of motion and time.

8.3.1.2 Experiments

We ran about 20 experiments during the two week phase of milestone 2. The majority was to test the simulator environment and the learning algorithm.

At first, the experiment setup was quite trivial by providing the agent with a reward for moving above a certain low speed (in this case 5 km/h), and penalizing it for crossing the broken lines while terminating episodes after crossing solid lines or crashing.

The crossing of lines is registered using a lane invasion sensor in the CARLA simulator. This sensor registers all lane changes and the associated line crossing. CARLA uses line names as defined by the OpenDrive standard. For this thesis, the labels in Table 8.6 are relevant.

Table 8.6: OpenDrive Line Marking Labels.

Label	Description
Broken	Dotted, or broken, line marking.
Solid	A single solid line, usually at the side of the road before a pedestrian walk or grass begins.
SolidSolid	A double solid line, often separates two lanes in opposite direction.
NONE	Often, a single solid line, but could reference not categorized lane markings, too.

By using this information, a reward can be defined that informs the agent of a line crossing.

8.3.1.3 Challenges

What became noticeable was while watching the training for a few episodes it seemed that the observed behavior did not permeate through to the trained model. At first, a general unsuitability of the Double DQN model for such a complex task was assumed. After some debugging it was noticed that the training part was not called after sampling and predicting.

An indication for this behavior could have been that the level of reward decreased with the epsilon decay. This fact was noted, but the conclusion was not yet drawn at that point. The reason for the missing training call was mainly due to the fact that the DQN is self-implemented and heavily modified from the initial adaptation as described in subsubsection 8.3.1.1.

After fixing the training part, the model learned successfully and settled on the optimal policy of just running straight ahead to collect the most reward. But running straight was the only action it learned from the training setup. When letting it run for longer than 30 seconds (to which the training time was restricted), it just kept going straight, even if there was a curve ahead.

Another topic that will probably follow through to the end of this thesis is the reward engineering. Giving rewards has to be thought through properly, since the rewards are the

values that the model optimizes for. According to [20] the reward function needs to be properly designed and validated.

After all, the efforts spent in this milestone were worth it since the accumulated learnings, the setup of the infrastructure and the increasing familiarity with the CARLA simulator and the reinforcement learning concepts led to the successful training of a line-following car.

8.3.2 First Vehicle Platooning Scenario

In milestone 3, the goal is to train a model that can follow the leader in a simple scenario. Hence, a second car, the leader car, is added that simply drives straight ahead along a road. In Hypothesis 3 we define the parts that are of interest in this experiment.

Hypothesis 3. *A positive incentive to drive rather than to stand still, with negative incentives to cross broken and solid lines leads to a stable following behavior in a simple obstacle-free scenario.*

The model is trained with a simple reward function:

- small positive incentive to drive above zero kilometers per hour,
- positive incentive to stay within 20 to 50 meters Euclidean distance to the leader,
- small negative incentive of when crossing a broken (i.e, dotted) line,
- negative reward of -1 along with the termination of the episode when crossing a solid line or colliding with an object.

CNN We reduced the CNN from a massive InceptionV3 with about 22 million trainable parameters down to a much simpler CNN with around 82’000 trainable parameters when the input image’s resolution is at 128 by 128 pixels. This increased the training speed and reduced the model size significantly: from 245 MB for the InceptionV3 down to 360 KB for this simple CNN. The new CNN has three convolutional layers, each consisting of 64 filters with a stride of 3x3. The final layer is a Dense layer with output neurons equal to the number of actions (e.g. 3 for forward, left and right)

Segmentation Camera The segmentation camera is used instead of an RGB camera with the reason that a pre-processed segmentation should help the CNN to identify relevant features. CARLA provides a segmentation camera as part of its suite of sensors.

Training episodes At first, it seemed that the model learns quite quickly and not a lot of simulation time is required to arrive at an acceptable model performance. However, now

that some models were trained over more than 20'000 episodes, it turns out that they become way more stable and flexible, especially dealing with new situations.

Synchronous mode Due to other higher priorities for the current milestone (e.g. input segmentation, evaluation automation, location accuracy using vectors), the synchronous mode has not been implemented yet. As an approximation, the frames per second (FPS) were observed to be around 4. Which means that the training sequence can process around four input images per second and train the model with it.

Therefore, the time to take images using the camera was set to 4 per second. This is not as accurate as synchronous mode, but it seems to be good enough for now to train better models than previously. However, as soon as models of different complexity are trained, this estimate may not be sufficiently accurate and hence, the synchronous mode should be used.

Other configurations The rest of the configurations can be found in Table A.1. Most notably is the fixed steering value of ± 0.2, that was chosen for this experiment as per Table 6.1.

8.3.2.1 Evaluation

At different stages during the training, especially when an obvious increase in rewards is observed in the training monitoring charts in Tensorboard, a few of the more recently saved down versions of the model were evaluated. An example of the monitoring chart for this experiment can be found in Figure A.7.

The most interesting evaluations were two models, where the first with ID 3930 was used as the basis to continue the training that resulted in the second model with ID 9640. The IDs refer to the number of episodes that they were trained on. The second model is therefore trained on a total of $3,930 + 9,640 = 13,570$ episodes which lasted over two days of training time.

For the evaluation as described in Table 8.7, the leader speed was set to 25 km/h, the autopilot for the leader vehicle was activated and instructed to only drive straight ahead, however respecting the lanes including curves and intersections. The traffic lights were all set to green so that the leader does not have to perform a full stop since it was not part of the training as the follower has not yet the ability to brake (only to slow down).

All long-list evaluations can be found in Figure A.3.

Table 8.7: Selected model evaluation in first platooning experiment.

ID	Speed	LC[1]	Visual Observation
9340	25 km/h	No	• Follows quite well, even in new situations • In the beginning it stays in the lane • Overall, good, but not too stable
9640	25 km/h	No	• Very slow start, but catches up quickly • It follows the leader extremely well, even slowing down when the leader slows down at an intersection, but also navigating the left curve (twice!) • The interesting part is that in Town06 there is another vehicle placed besides the road. When it gets into the followers view the follower gets confused because it identifies two vehicles. After the leader vehicle is out of sight, it turns to the stationary vehicle (as expected) and tries to follow up, hence crashing next into it. • This is a proof that the model learned to identify the leader car and that it should follow it. Once it is out of sight, it does no longer know what to do.

[1] LC = Lane Change; it indicates the direction of the leader doing a lane change.

8.3.2.2 Analysis

As defined in Hypothesis 3 the positive incentive to drive had definitely an impact in that the follower car started to drive. Also, the negative incentives to cross solid and broken lines had some impact, although not in the desired stability. The follower crossed broken lines even though there was no need to. Also, but this was not specified in the hypothesis, the follower's steering seemed to be somewhat unstable in the sense that it seemed to zigzag at some points. This may be due to the fixed steering value.

8.3.2.3 Enhancements

Even thought the model with ID 9640 performs very well and it was good to see the progress, there are a few things that can be improved.

Camera The input image is currently 128 by 128 pixels. We could increase the input to higher resolutions which would allow to track some objects better and observe them from further away. Currently, with a distance of about 150 meters the leader car disappears from

the input image since it is smaller than 1 pixel in size. The follower car loses sight of it and gets disoriented.

Frame Stacking With only one frame from the camera image as the input, the model has no sense of direction or history. If more frames are stacked on top of each other and processed in the CNN, then the direction of the movement of the leader car can be extracted from the input. This is also done in [71] where four frames are stacked.

Frame Skipping Another feature from [71] is the frame skipping where the model is trained only every fourth frame. In the meantime, the previous action is simply repeated. This is to collect more experiences since the ones at the beginning appear quite often in the replay buffer, but later states are scarcer. Using frame stacking with the same value of four in combination with frame skipping ensures that actually all frames during an episode are in the replay buffer.

Steering The model with ID 9640 wiggles quite a lot due to the fixed steering value. To smoothen the movements, a steering delta, rather than only a fixed value can be applied.

Bias In this experiment, the leader and follower started from the same initial location with the leader always driving straight. To add more bias to the replay buffer, different locations should be used in addition to varying leader behavior.

8.3.3 Implementing Upgrades

In milestone 4 the focus was on implementing the changes from the previous experiment, such as frame stacking or adding more different experiences. This resulted in only a minimal amount of time left to perform experiments. What is noticeable from some early screenings, is, that the follower is more stable. The left/right jerks of the follower are greatly reduced and much smoother due to the delta value for steering rather a fixed left-/right number. However, while the follower can identify and follow the leader, it still crosses the dotted lines or bumps into the leader if it stops.

8.3.3.1 Observations

Model Fluctuations Models are saved to disk every ten episodes in order to reload those that look interesting or promising from the training monitoring charts. When evaluating the different models, it becomes apparent, that the behavior of the follower can drastically differ between ten episodes. It may be that the follower identifies and follows the leader in one model, but it fails to move at all in the next model that is only ten episodes apart. These

stark fluctuations make it challenging to assess which model is promising. It requires to run and watch many models.

Mapping Errors The lane invasion sensor is used to register when the follower crosses a broken or solid line. At some spawn locations we noticed that the sensor does not register a line crossing at all. Primarily around intersections, where many different lines are visible, they are not registered as such in the metadata which the lane invasion sensor could detect.

This is especially problematic when solid line crossings are not registered, since an episode is terminated in that event along with a high negative reward. It may appear to the follower that in certain occasions the crossing of a solid line has no impact. In that case, the model may learn the wrong behavior.

8.3.4 Spawn Points, Atari and Reward Function

Spawn Points One feedback from the interim presentation was to add more different experiences. This implies that the locations from which each episode starts should be randomly chosen. The challenge with finding such locations, is, that two such points are required that are aligned in a way that the follower has sight of the leader.

In Town06 there exist 435 different so-called spawn points. These are points defined by the map creators. They are callable from the script and they are guaranteed to lie on a road and in a location from which any car could start (e.g., not in the middle of an intersection). The challenge was to go through many or all of these points and identify those which satisfy the criteria of being aligned, i.e., ideally in-line on the same lane.

By using a script, this arduous task was greatly simplified, but it took a few hours to locate enough valid spawn points that could be used for the training. In total, 60 different spawn point locations were used for the training. Listing A.1 contains all spawn point locations that were used for the training. Figure A.1 provides an example of the mapping exercise that was conducted to identify suitable spawn points.

Frame Stacking In the last milestone we implemented the frame stacking feature. It allows to stack an arbitrary number of frames on top of each other. The additional dimensions are simply added to the color channel. For example, by stacking four images with three color channels, an image with 12 color channels results.

Replay Buffer Size Another hyperparameter that can be tuned is the size of the replay buffer. It needs to be as large as possible, so that the memory (either system or GPU) is utilized as much as possible.

With a frame stacking of four in place, the size of the input image is increased by four, but two times: for the current and the next observation. Since the available system memory was not fully utilized in prior experiments, this is an opportunity to optimize its usage.

The calculation is as following:

```
1    image height * image width * color channels * stacked frames * bytes per ↵
  pixel * 2 observations.
```

Since we use an unsigned integer of size 32 bits to represent values from 0 to 256, a pixel is 4 byte in size (this was later changed to a uint8 size, reducing the size by a another factor of four). Hence a 128 by 128 pixel RGB image results in a size of 1.5MB as per Equation 8.1 (the division by 1,024 is necessary to convert from bytes to megabytes).

$$\frac{128 \times 128 \times 3 \times 4 \times 4 \times 2}{1024 \times 1024} = 1.5\text{MB} \tag{8.1}$$

With 32GB of system memory available, minus a few gigabytes that CARLA and the operating system itself requires, we can safely use a replay buffer of 15,000 images as Equation 8.2 demonstrates.

$$\frac{15000 \times 1.5}{1024} = 21.97\text{GB} \tag{8.2}$$

Atari CNN In addition, the CNN is switched from the custom CNN with three hidden layers each with 64 filters to the Atari CNN as proposed by [71], with implementation details by [72]. It increased the trainable parameters from around 82,000 to over 4.8 million, however runs twice as fast as the previous custom CNN, probably due to not using an average pooling at each layer. With this change, the CNN is a factor less to consider that could influence the results. Nonetheless, further experiments with different CNNs could be conducted, of course.

Reward Function Finally, the reward function is improved from a simple Euclidean distance metric to multiple functions that calculate the point-line and the scalar-projection distance along with the angle as described in subsection 6.4.1.

The changes are listed in Table 8.8.

Table 8.8: Experiment setup for the replay buffer increase.

Parameter	Change	Rationale
Spawn points	From 1 to 60 locations	Add more spawn points to increase the variance in the replay buffer.
Replay buffer	From 10,000 to 15,000	Increase by 5,000 due to available system memory.
Frame stacking	From 1 to 4	As per the Atari paper [71], 4 frames are stacked.
Atari CNN	From custom to Atari CNN	CNN as per the Atari paper [71].
Reward function	From simple to complex	The simple reward function which was based on the Euclidean distance is improved with scalar-projection and point-line distances, and angle.

8.3.4.1 Observations

The resulting models were evaluated at three different locations, which were part of the training

After a total training time of over 72h for 10,000 episodes, the results as listed in Table 8.9 are mixed. It may be that due to the higher number of spawn locations, a longer training time is required to achieve the same number of experiences as in a single location setting. Nonetheless, the follower shows some signs of activity, just not yet the desired following behavior.

8.3.5 Reduced Input Size

In another experiment, after receiving access to the NVIDIA DGX-2 HPC (see subsubsection 7.1.2.2 for details), the image input size is reduced to allow for a higher batch size and larger replay buffer. It is in contrast to the previous observation that a large input image could help identifying objects from further away. While the GPU and system memory limits impose an upper limit on how many images can be stored, a faster GPU can process a larger batch size, hence train the model on more samples in one step.

Also, a smaller CNN should speed up the training and prediction phases. By reducing the image input size, the number of trainable parameters is reduced by almost a factor of 9 from 4,816,037 to 621,733. The caveat is the smaller number of distinct states that can be identified by the CNN.

Two models were trained, one with 128 by 128 and the other with 64 by 64 input size. The batch size for the 64 by 64 pixel model was increase from 256 to 1,024.

Table 8.9: Results of the spawn points, Atari CNN and new reward function experiment.

ID	Location 7:2, 30 km/h	Location 111:131, 30 km/h	Location 317:404, 30 km/h
9860	No action, braking only	Follows quite good	Slight forward, then stops
9870	No movement	No movement	No movement
9880	Slight forward	Follows, then starts turning rather than going straight	Slight forward, then stops
9900	Crashes early	Crashes early	Turns only right, then crashes
9960	No movement	Minimal action, braking mainly	Minimal action, braking mainly
9890	No movement	Follows a bit, then crashes	Just slight forward, then stops
9910	Crashes early	Follows straight, then collides with leader	Drives, but crashes
9920	Only braking	Follows until left exit, then stops	Only braking
9999	Follows a bit, but crashes early	Follows straight, then collides with leader	Drives, then crashes
9940	Crashes early	Waits, then crashes	Drives in a left circle
9930	Crashes early	Waits, then crashes	Drives in a left circle
9980	Crashes early	Waits, then crashes	Slight forward, then brakes
9950	No action	No action	Drives left, then crashes
9970	No action	Waits, then small advance, stops	Slight forward, then stops
9990	Crashes early	Follows straight, then turns right	Follows a bit, then veers to the left and crashes

Also, All changes are listed in Table 8.10.

Table 8.10: Experiment setup for the reduced input size experiment.

Parameter	Change	Rationale
Image size	From 128x128 to 64x64	Reducing the image size by 4x from 1.5MB to 0.375MB.
Batch size	From 64 to 1,024	Increasing the batch size to better utilize the GPU.
Normalization	Rewards are not normalized	Investigate the effect of higher cumulative returns.

8.3.5.1 Observations

After 5,000 episodes, the loss of the smaller model of 64 by 64 pixels seemed to have exploded. Most likely this was due to the rewards not being normalized. Also, it overfit on a specific action, which was confirmed in the evaluation. Every model that was evaluated only turned left, without any other action.

The 128 by 128 pixel models expressed more diverse actions, but it seemed not to recognize the leader vehicle. At some points, even when driving straight ahead, it suddenly turned left or right and crashed.

The results can be found in Table A.4.

In a next experiment, the normalization of the rewards needs to be re-enabled.

8.3.6 Finer Steering

In milestone 6 the experiment is based on the observation that the steering delta might be too coarse and it is difficult for the follower to stay aligned. Mainly, some experiments showed that almost no steering was performed, which indicated that it may be too complex and the optimal strategy is to avoid steering as much as possible.

In contrast to the first platooning experiment described in subsection 8.3.2 where the follower could simply reset the angle of wheels to zero, this is not possible with a steering delta. There, multiple steps are required to align the wheels as desired. Also, since the delta uses floating points, minor inaccuracies can emerge from adding and subtracting the delta.

The changes are listed in Table 8.11.

Table 8.11: Experiment setup for the finer steering experiment.

Parameter	Change	Rationale
Steering delta	From 0.01 to 0.005	Doubling the steps from 200 to 400 for a full left-to-right steering.

8.3.6.1 Observations

From visual observations it seems that some models work quite well in terms of following.

Reduced Distance What is common is that the follower stops as soon as it loses sight of the leader. This happens much earlier at a distance around 70 meters due to the smaller 64 by 64 pixel resolution of the camera input.

The distance of 70 meters is about half of the 150 meters with an 128 by 128 resolution. This distance can be reached quite quickly, and hence the follower is stopping rather early if the leader is too fast.

Reduced Steering Also, these models rarely turn left or right, which indicates that the steering is still too complicated. The resulting negative rewards are too high when they start turning so the best strategy is to turn only when absolutely needed or not at all.

Training Time The speed up from the reduced image size is about 1 to 7 frames per second on the CPU. In addition, the replay buffer from which the Double DQN algorithm samples is increased by four. Ultimately, this should lead to a more robust model since the experiences differ.

8.3.7 Reward Weights

Since the follower is not able to determine from the input image the angle at which it faces the leader vehicle, this reward was removed by setting its weight to the overall reward to zero.

The other two reward functions, the point-line and the scalar-projection distance (see subsection 6.4.1 for details), an experiment was designed to determine the optimal weights. To narrow down the possible weights, two different configurations were run. The first had a weight of 0.3 for the point-line distance, and a 0.7 weight for the scalar-projection distance. The second configuration switched the weights as can be seen in Table 8.12.

Table 8.12: Experiment setup for the reward weights comparison.

Config	P/L[1]	S/P[2]
A	0.3	0.7
B	0.7	0.3

[1] P/L = point-line distance.
[2] S/P = scalar-projection distance.

8.3.7.1 Observations

The evaluation results for config A can be found in Figure A.4 and for config b in Figure A.5.

The config B seems to outperform config A, since it shows some signs of following behavior. Also, the collected reward during training is higher. This can be seen in Figure 8.14.

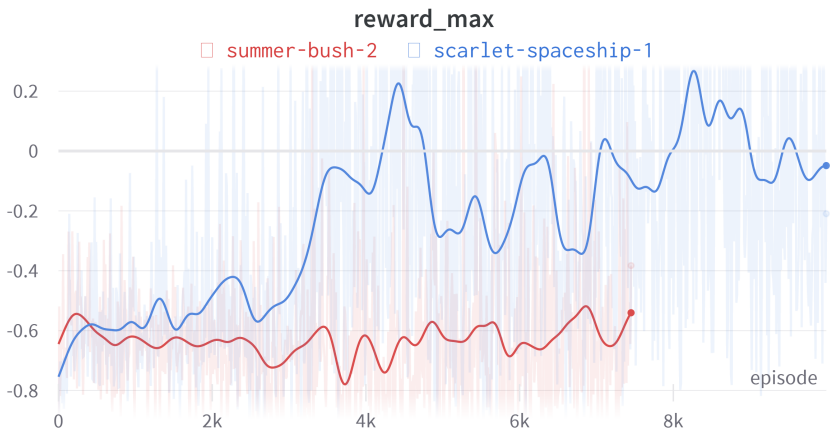

Figure 8.14: Reward comparison for config A (summer-bush-2) and config B (scarlet-spaceship-1)

It seems that config B is more likely to collect higher rewards, since previous runs with the same configuration also show a higher reward as seen in Figure 8.15 compared to other runs with config A (Figure 8.16).

Going forward, the config B will be used for experiments.

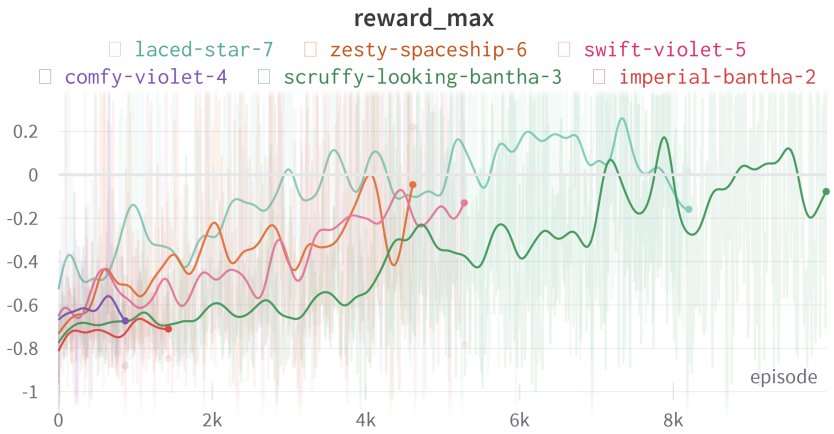

Figure 8.15: Reward comparison for previous runs with config B.

8.4 Performance Comparison

What we noticed when launching the training for the first time without any modifications to the code apart from the upgrade to Tensorflow 2, was, that the training did not run as fast as we expected. It did in some cases not even run as fast as on the local computer's CPU.

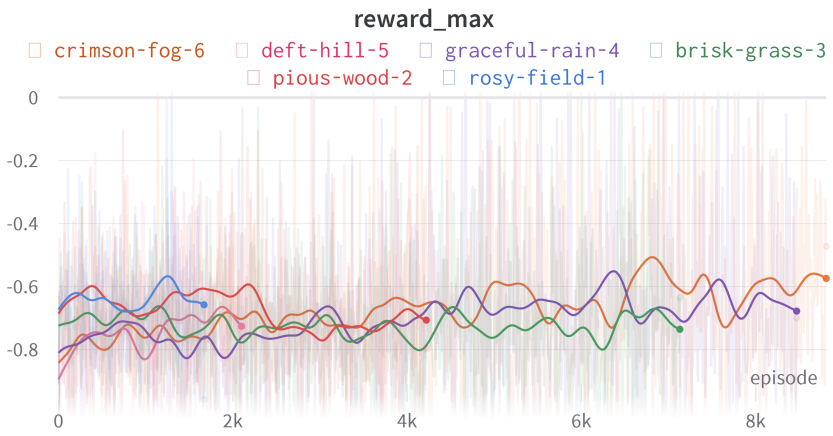

Figure 8.16: Reward comparison for previous runs with config A.

There are multiple reasons that could slow down such an execution.

Parallelization First, the majority of the code is not parallelizable. Much of it is setting up the CARLA environment, sending commands to the server, retrieving image and other sensor data, along with storing this data in variables. This part is not suitable for parallel processing, since it is a sequence of actions which the two cars finally require to execute.

Tensorflow Second, Tensorflow reserves by default the complete available shared memory of the GPU. In the case of DGX-2 which has NVIDIA Tesla V100 GPUs, it reserves 32 GB of memory. There is no space left for any other application data.

CPU Third, while the training and prediction parts are the only ones relevant for a GPU parallelization, the remaining part could be run on the CPU. Currently, everything is instructed to run on the same GPU, however, parts of it could be either run on a separate GPU or on the CPU itself.

Memory Fourth, it is unclear which data is stored on the GPU's shared memory and if the system memory is also used. Transferring data between different memory types or even between memory on different GPUs induces a latency that may accumulate if memory is accessed frequently. Therefore, it is important to control the data that is often required to reside close to the GPU (such as the replay buffer) whereas information on the training progress can reside in system memory.

These reasons need to be investigated further and may help to improve the training performance significantly. An experiment was run which compares the times for predictions

and trainings (i.e., fits) on the DGX-2 and the local PCs (CPU and GPU). The results are presented in Figure 8.17 and Figure 8.18.

The number prefixes *01* and *02* refer to the two local PCs, 128 refers to the batch size, and *diff_GPU* or *same_GPU* means that the CARLA simulator was running on a different or on the same GPU as the training. The last two entries with *2048/16_128* refer to a batch size of 2,048 that was split into 16 minibatches, which equals 128 individual batches when calling the fit function.

This split seems to reduce the time per fit of a 128 size minibatch significantly. However, running it on the DGX-2 seems slower overall, independently if CARLA and the script runs on the same GPU or not. What is definitely a massive speed up is the time for a single fit on the PC on a CPU versus the GPU. However, the inference seems faster on the CPU again.

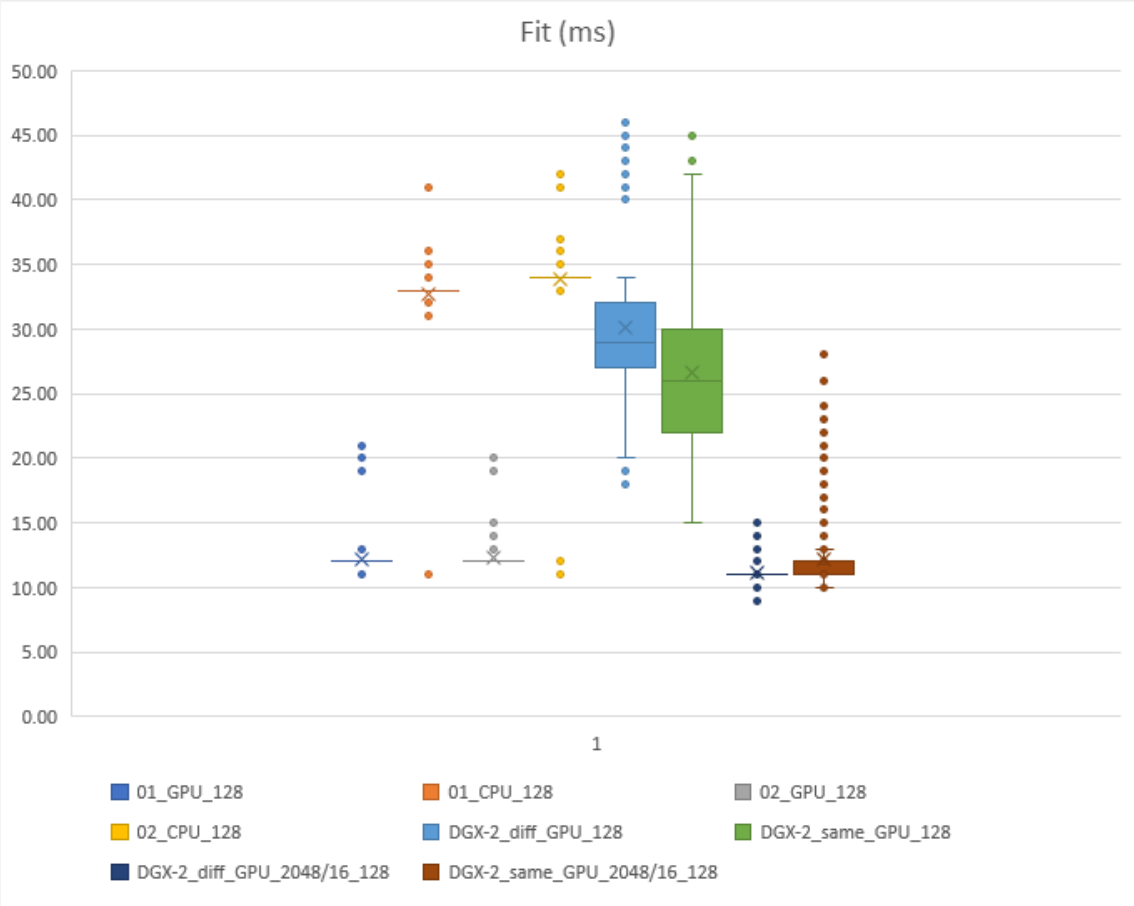

Figure 8.17: Comparison time for a training (fit) instruction to complete (in milliseconds).

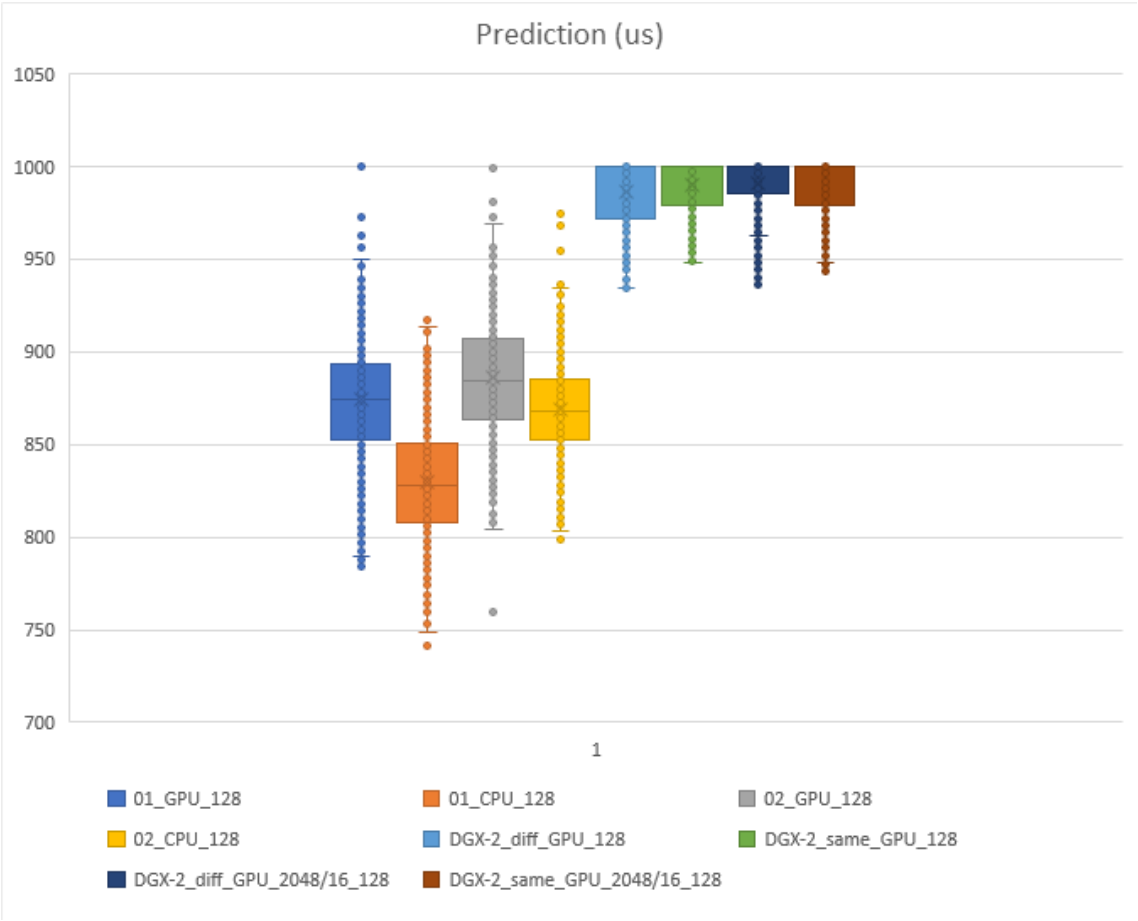

Figure 8.18: Comparison time for a prediction instruction to complete (in microseconds).

Chapter 9

Conclusion

Our main contribution is the successful training of a policy using reinforcement learning for a follower car in a platooning scenario. Specifically, the trained model allows the follower car to form and maintain a platoon by identifying and adapting its action to the leader vehicle.

9.1 Achievements and Contributions

The successful training of two different models that enable a follower car to form and maintain a platoon in a 3D multi-lane street scenario is a great achievement of this thesis. These models were trained with the Double Deep Q-Network reinforcement learning algorithm and by using the CARLA simulator for the environment.

9.1.1 Main Contribution

The initial research on existing work revealed that while reinforcement learning approaches are used in the field of platooning, they are rarely, if at all, used to enable participating vehicles to join and to maintain a platoon. Hence, a model which is trained specifically for platooning has, to our knowledge, not been released previously.

9.1.2 Other Achievements

Based on the task description for this thesis, multi-lane scenarios, crossing of intersections, dynamic obstacles in the form of traffic lights and random behavior of the leader vehicle are all requirements that were achieved as part of this thesis.

9.1.3 Code Base

Even though the code base was only rarely highlighted in this report, a significant amount of time was spent on its development. It is another important part that contributed to the achievements of this thesis. Crucially, it enables the seamless interaction of the Python-based model training with the CARLA simulator. Additionally, the evaluation, performance metric extraction, chart generation and video recording generation of each model is highly automated. With only a few parameters to set, a great number of different models can be processed, hence reducing the time from designing an experiment to the result generation.

9.2 Learnings

Many more models were trained, some with more others with fewer resulting capabilities of identifying and following a leader car. However, these intermediate steps were necessary to improve the algorithm based on the learnings from each experiment.

An important learning was how the design of actions greatly influences the agent's ability to learn a task, but also how increased complexity prolongs the training time. This was especially true for the delta steering, where the complexity was probably too high. Due to this, not all desired milestones could be achieved, such as the platoon navigating in dense traffic or re-forming after a split. Nonetheless, these goals seem to be feasible, especially with more training time or computing power available.

Also, the lane keeping turned out to be more difficult to achieve than initially thought. It seems that simply providing rewards for staying aligned with the leading vehicle is not sufficient, since the follower frequently deviates from the optimal path. Actually, it is not that surprising when considering that it never learned the concept of a lane. Something that could be achieved with a dedicated training on lane keeping, potentially combined with a hierarchical reinforcement learning approach.

Another element is the CNN which seemed to work well in identifying the leader vehicle from the input image. However, the identification of lane markings could have played a role in the difficulty of lane keeping. While the main focus of the model should be to follow the leader, the various lane markings could potentially have added complexity to the training of the CNN. An idea is to remove the lane markings from the input image and only train it with the leader vehicle.

9.3 Future Work

While the action space of our Double DQN algorithm is discrete and somewhat limiting the practical applicability, an extension to a continuous action space is simply a matter of time. With that, a switch to a suitable RL algorithm is required, which is, again, only constrained by the available resources. Therefore, based on this research, a number of further experiments and extensions can be built.

9.3.1 RL Libraries

One obvious enhancement is the use of RL algorithm libraries, such as Stable-Baselines3 or RLlib, that offer tried and tested algorithms. It would offer access to other algorithms, but it would also reduce the risk of errors in a custom implementation (there are still many that could be made).

9.3.2 GPU HPC

Another improvement with a significant impact is the usage of a GPU cluster by adapting the code base to better utilize its resources. Key challenges are the separation of sequential instructions when accessing memory or the CARLA simulator, and designing a parallelizable batch processing.

9.3.3 Hierarchical RL

As briefly mentioned, hierarchical RL could be a suitable approach to tackle the task of platooning. It may be composed of different conditions, such as when the vehicle is just driving and needs to respect the traffic regulations, or when it joined a platoon and its main task is to imitate the leading car's actions. Alternatively, multiple models can be trained, each with a different capability, like one that learned to imitate a leading car's actions. They are eventually triggered based on the condition the vehicle finds itself in.

9.3.4 String Stability

The number of follower cars in the experiments of this thesis was always one. When adding more follower vehicles, then the problem of string stability arises. There exists a number of research on connected vehicles in a platoon, some even using reinforcement learning, that could be leveraged to tackle this issue. However, since in our case the vehicles do not have means to communicate with each other, experiments should be performed that address string stability.

9.3.5 Obstacles and Traffic

Increasing the complexity of the environment is a further area of potential research. This can be achieved by adding obstacles to the road, either in the form of static objects, like parked cars or roadworks, or dynamic objects, like other road users. Also, traffic could be added that interferes with the platoon by cutting into an existing platoon or even splitting it. Such an enhancement implies the extraction of relevant features from an image through segmented information, or even other sensor types like lidar or depth cameras.

9.3.6 Sim-to-Real Transfer

The ultimate challenge is an attempt to transfer a model to a real car. Even if only considering a toy car could be a significant achievement, since the control of actuators, the on-device image processing, and the adaptation from the simulated to the real world are non-trivial problems. However, the learnings and discoveries from such an endeavor are likely to be worth the effort and they could even advance the field of autonomous driving in general.

Appendix A

Experiments

A.1 Configuration Files

A.1.1 First Vehicle Platooning Scenario

Table A.1: Configurations for the first vehicle platooning experiment.

Parameter	Value
AGGREGATE_STATS_EVERY	10
DISCOUNT	0.99
EPISODE_LENGTH	FPS * SECONDS_PER_EPISODE
EPISODES	5_000
EPSILON_DECAY	0.999
EPSILON_START	1
FOV	90
FPS	4
IMG_HEIGHT	128
IMG_WIDTH	128
MEMORY_FRACTION	0.8
MIN_EPSILON	0.1
MIN_REPLAY_MEMORY_SIZE	1_000
MIN_REWARD	-3
MINIBATCH_SIZE	16
MODEL_NAME	" ""20230329_02"""
NUM_ACTIONS	4
PREDICTION_BATCH_SIZE	1
RELOAD_TOWN	True

Parameter	Value
REPLAY_MEMORY_SIZE	5_000
SAVE_IMGS	False
SECONDS_PER_EPISODE	30
SET_SPECTATOR	True
SHOW_PREVIEW	False
SPAWN_EGO	2
SPAWN_LEADER	7
STEER_AMT	0.2
THROTTLE_EGO	0.5
THROTTLE_LEADER	THROTTLE_EGO * 0.8
TICK	0.1
TIMEOUT	20
TOWN	'Town06'
TRAINING	True
TRAINING_BATCH_SIZE	MINIBATCH_SIZE // 4
UPDATE_TARGET_EVERY	10

A.1.2 Throttle-Brake Comparison

Table A.2: Configurations for the throttle-brake comparison experiment.

Parameter	Value
brake	0.1
throttle_ego	0.3
aggregate_stats_every_ep	10
cam_fov	90
car_model	model3
carla_port	2000
carla_timeout	30
cnn	Atari
connect_carla_server	TRUE
device_type	gpu
discount	0.9
env_name	carla_for_rl/CarlaSimTown06-v0.6
episodes	15000
epsilon_decay	0.99965
epsilon_min	0.05

Parameter	Value
epsilon_start	0.4
fixed_delta_seconds	0.1
frame_skip	4
frame_stack	4
frames_per_ep_est	300
image_height	64
image_width	64
leader_lane_chg_left_pct	1
leader_lane_chg_right_pct	1
leader_speed_diff	60
learning_rate	0.00025
load_model	FALSE
loss	Huber
max_episode_duration	30
max_euclidean_dist	100
max_substep_delta_time	0.01
max_substeps	10
min_replay_mem_size	10000
min_reward	-1
minibatch_size	64
model_folder	models
model_name	fix_spawn
model_path	/
num_actions	5
optimizer	Adam
prediction_batch_size	1
random_spawn	FALSE
random_traffic_manager	TRUE
reload_town	TRUE
replay_memory_size	45000
reward_strike	-1
reward_weights.angle	0
reward_weights.point-line	0.7
reward_weights.scalar-proj	0.3
rgb_channels	3
save_images	FALSE
save_model	TRUE

Parameter	Value
seed	42
sensor_tick	0
set_lights_green	FALSE
set_spectator	TRUE
spawn_ego_loc	2
spawn_leader_loc	7
steer_delta	0.005
town	Town06
traffic_manager_port	8000
training_active	TRUE
training_batch_size	16
update_target_every_step	1000
use_rgb_cam	FALSE
verbosity	0
wandb_logging	TRUE

A.1.3 Spawn Points

```
    spawn_pair_loc = {
        5: 0, 6: 1, 7: 2,
        71: 75, 72: 84, 73: 85, 74: 86,
        87: 91, 88: 92, 89: 93, 90: 94,
        95: 99, 96: 100, 97: 101, 98: 102,
        111: 131, 110: 130, 109: 129, 108: 128, 107: 127,
        179: 423, 180: 424,
        185: 181, 186: 182, 177: 421,
        223: 218, 222: 217, 221: 216, 220: 215, 219: 214,
        236: 403,
        252: 247, 253: 248, 254: 249, 255: 250, 256: 251,
        258: 311,
        295: 226, 294: 225, 293: 224,
        312: 287,
        317: 404,
        332: 309, 328: 288,
        338: 317, 342: 116, 341: 115,
        350: 87, 351: 88,
        375: 390,
        405: 107, 406: 108, 407: 109, 408: 110, 409: 111, 410: 111,
        266: 363, 415: 266, 296: 343, 387: 296,
    }
```

Listing A.1: Spawn-pair locations used for training

A.1.4 Fixed Steering

A.2 Long-List Evaluations

A.2.1 First Vehicle Platooning Scenario

A.2.2 Throttle-Brake Comparison

Table A.3: Evaluations of the long-list for the throttle brake comparison experiment..

Model	30 kmh (seed 42)	Total reward
780	No movement	
790	No movement	
800	Starts to drive; then stops	
810	Starts to drive; then stops	
820	Waits; then drives left; stops before solid line	
830	Starts to drive straight; then a bit left; then stops before the solid line	
840	Starts to drive slowly; then stops	
850	Starts to drive right; then stops before solid line	
860	Starts to drive right; then left; but too slow; loses leader	
870	Same as 860	
880	No movement	
890	No movement	
900	Starts slowly straight; then loses leader	
910	Same as 900	
920	Turns right; then waits just before solid line	
930	Same as 920	
940	Same as 920	
950	No movement	
960	No movement	
970	Starts to follow; but slows down; waits; loses leader;	-0.2908
980	Starts to follow; but turns left and waits before solid line	-0.116
990	Starts straight; but then waits; loses leader;	-0.3162
1000	Only turns left; then crashes;	0.0677
1010	Starts straight; then left; crosses green;	-0.2927
1020	Starts straight; then waits; loses leader;	-0.2394
1030	Same as 1020;	-0.1825
1040	Initial attempt to follow; but does not turn as leader does; waits;	-0.1563
1050	No movement	
1060	No movement;	-0.2196

Model	30 kmh (seed 42)	Total reward
1070	Starts to follow; loses leader; slows down;	-0.3834
1080	Same as 1070;	-0.2195
1090	Same as 1070	-0.3469
1100	No movement	-0.2198
1110	No movement	-0.2195
1120	No movement	-0.2196
1130	Starts to follow; then turns right; stops before solid line	-0.2677
1140	Same as 1130	-0.2646
1150	Same as 1130	-0.2665

A.2.3 Smaller Input Size

Table A.4: Evaluations of the long-list for the smaller input comparison experiment..

Model	Input Size	Visual Observation
4990	128x128	Drives; but randomly
4991	128x128	Drives; mostly right
4993	128x128	Drives straight; stays in the lane but then turns right and crashes
4992	128x128	Drives; straight; then turns left
4998	128x128	Drives; but randomly
4999	128x128	Only turns left
4994	128x128	Drives; then stops
4995	128x128	Drives; mostly right; then crashes
4996	128x128	Drives; right; then left and crashes
4997	128x128	Drives right; then crashes
4990	64x64	Turns left only
4991	64x64	Turns left only
4992	64x64	Turns left only
4993	64x64	Turns left only
4994	64x64	Turns left only
4995	64x64	Turns left only
4996	64x64	Turns left only
4997	64x64	Turns left only
4998	64x64	Turns left only
4999	64x64	Turns left only

A.2.3.1 Reward Weights

A.2.3.2 Fixed Steering

A.3 Training Monitoring

A.3.1 First Vehicle Platooning Scenario

A.4 Experiment Statistics

A.4.1 Throttle-Brake Comparison

A.4.1.1 Different Leader Speeds - Model 870

A.4.1.2 Different Leader Speeds - Model 1040

A.4.1.3 Different Leader Locations - Model 870

A.4.2 Fixed Steering

A.4.2.1 Unseen Location - Model 880

A.4.2.2 Unseen Location - Model 910

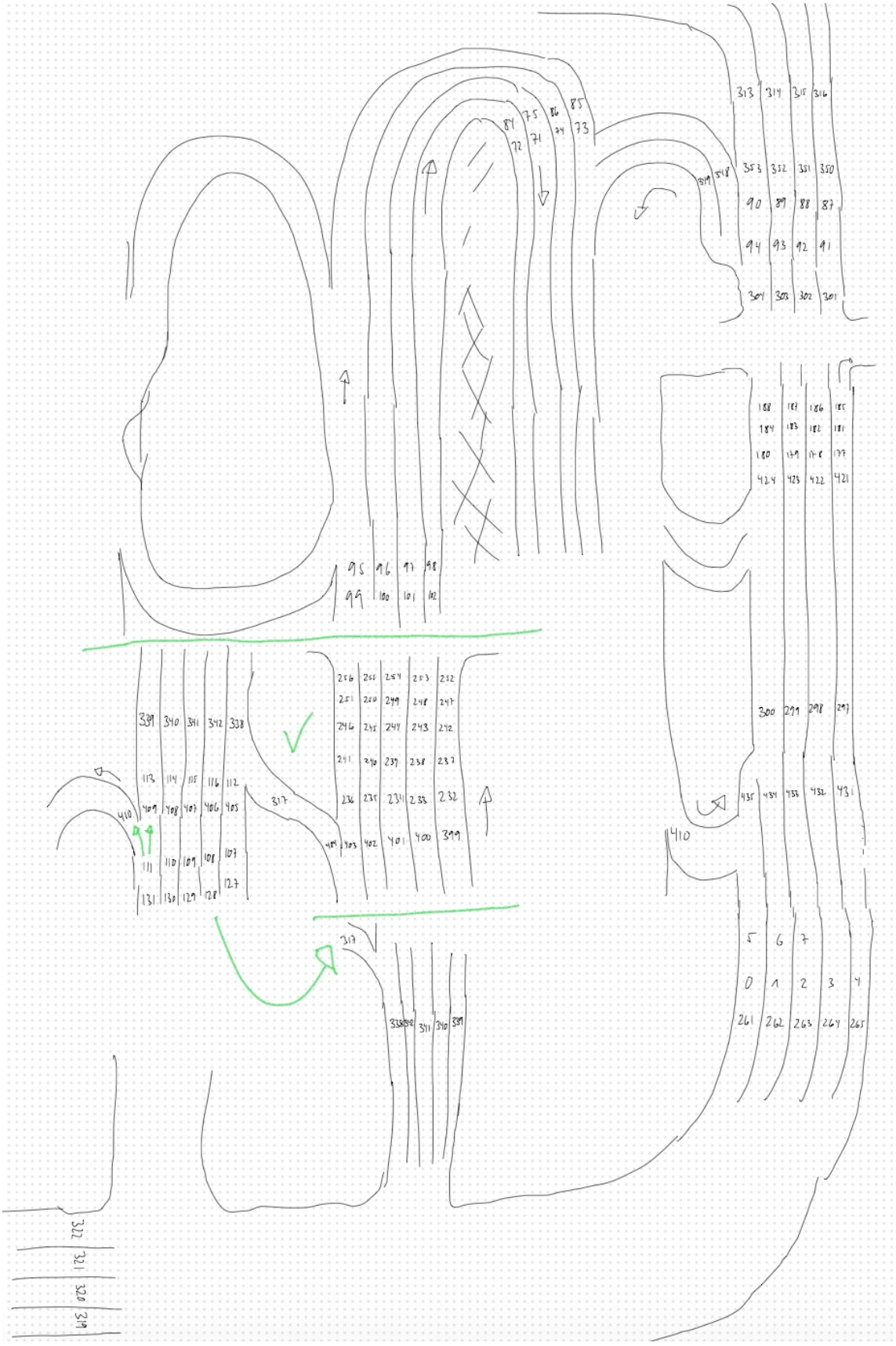

Figure A.1: Mapping of a subset of the 435 spawn points in Town06

Key	Value
env_name	...
device_name	1
device_type	cpu
wandb_logging	True
model_name	fixed_steering
load_model	True
save_model	True
model_folder	models
model_path	models/...
training_active	False
connect_carla_server	True
num_actions	5
image_height	64
image_width	64
rgb_channels	3
carla_port	2000
traffic_manager_port	8000
carla_timeout	30
reload_town	True
town	Town06
set_lights_green	False
car_model	model3
use_rgb_cam	False
cam_fov	90
sensor_tick	0
save_images	False
seed	42
random_spawn	True
spawn_ego_loc	2
spawn_leader_loc	7
set_spectator	True
random_traffic_manager	False
leader_lane_chg_left_pct	1
leader_lane_chg_right_pct	1
leader_speed_diff	60
throttle_ego	0.5
brake	0.25
steer_delta	0.005
steer_fixed	0.2
reward_weights	{
angle	0
point-line	0.7
scalar-proj	0.3
}	
reward_strike	-1
max_episode_duration	60
max_euclidean_dist	100
frames_per_ep_est	10*30
replay_memory_size	240_000
min_replay_mem_size	218_400
minibatch_size	128
training_batch_size	128 // 4
prediction_batch_size	1
update_target_every_step	1_000
min_reward	-1
episodes	15_000
discount	0.99
epsilon_start	0.09
epsilon_decay	0.99965
epsilon_min	0.05
cnn	Atari
learning_rate	0.00025
optimizer	Adam
loss	Huber
frame_skip	4
frame_stack	4
fixed_delta_seconds	1.0/10.0
max_substep_delta_time	0.01
max_substeps	10
aggregate_stats_every_ep	10
verbosity	0

Figure A.2: Configurations for the fixed steering experiment.

Model	Max Reward	Min Reward	Avg Reward	Avg Distance	Rating	Comment
3710	-0.78	-0.97	-0.92	24.31	0	Crashes
4520	1.37	-0.93	-0.6171		5	Follows quite ok; crashes after some time
4540	1.41	-0.96	-0.63			
4650	1.29	-0.97	-0.68			
4700	-0.78	-0.96	-0.91	25.59		
4730	1.26	-0.97	-0.7	27.76		Follows well at the beginning; then slows down and crashes
4970	-0.74	-0.97	-0.89	27.65	9	Very good following; very stable; slows even down to keep distance; at the end it approaches leader too closely and then gets confused; but so far so good. Second run even better; even more stable
5000	-0.72	-0.95	-0.86			Ok following in the beginning; then starts to wiggle; crashes eventually
9340						Follows quite well even in new situations; Still sometimes a bit unstable; but in the beginnings it stays in the lane; However; very sensitive to the speed of the leading car; Speed of 40 works quite well; However; 45 and above is too fast and it loses sight of the leader; eventually getting confused; Too unstable
9640						Very slow start but catches up quite well; Follows lane quite well in the beginning; If leader speed is 25.0 then it follows extremely well; even slowing down when the leader slows down at an intersection; but also making the curve (twice); The interesting part is that in Town06 there is another vehicle placed besides the road. When it gets into the followers view the follower gets confused because it identifies two vehicles. After the leader vehicle is out of sight; it turns to the stationary vehicle (as expected) and tries to follow up; hence crashing next into it. This is a proof that the model learned to identify the leader car and that it should follow it. Once it is out of sight; it does no longer know what to do.

Figure A.3: Evaluations of the long-list for the first vehicle platooning scenario.

Model	30 kmh	20 kmh (seed 42)	25 kmh	20 kmh (seed 21)	20 kmh (seed 21)2
9900	Drives straight; right next to the leader for a while; slows down; but then turns right and crashes				
9910	Turns left and stops				
9920	Turns a bit right; then stops				
9930	Turns left and crashes				
9940	A bit straight; then left and crashes				
9950	Drives straight in the lane next to the leader for a while; then suddenly stops	Follows leader in lane next to it; even slows down; but as soon as leader is too small to detect; it stops	Similar to 20kmh	Follows similarly but stops when it loses sight	
9960	Drives straight in the lane next to the leader for a while; then suddenly stops	Follows leader much better; always on the right side not straight behind; does not follow when leader changes lane; but slows down; waits for it to merge again; but when it loses sight it stops	Similar to 20 kmh	Follows similarly; very close to the leader but does not crash; however it turns right to avoid a crash but then stops as it loses sight	Follows closely; even across two intersections; speeds up to catch up with the leader and even turns a bit to better adjust; but at some point is probably confused and stops
9970	Drives straight in the lane next to the leader for a while; then suddenly stops; probably when it loses sight of the leader	Follows very closely; waits at the intersection; but again; when it loses sight it stops; also the different lines at the intersection seem confusing	It stops right next to the leader at the intersection; continues again when green; but crossing the second intersection is too confusing due to the many lines	Crashes into the leader at high speed	
9980	Turns left and crashes				
9990	Turns left and crashes				
9999	Turns right then stops				

Figure A.4: Evaluations of config A for the reward weights comparison experiment.

Model	20 kmh (seed 21)	25 kmh (seed 21)	30 kmh (seed 21)
9900	Follows quite closely; then after 30s turns slightly right and stops	Follows quite closely; then after 30s turns slightly right and stops before the solid line	Loses leader quite early and stops
9910	Turns right and stops on the road but just before the solid line	Turns right and stops on the road but just before the solid line	Same as 25kmh
9920	Turns right and stops on the road but just before the solid line	Turns right and stops on the road but just before the solid line	Same as 25kmh
9930	Turns right and stops on the road but just before the solid line	Turns right and stops on the road but just before the solid line	Same as 25kmh
9940	Follows closely; adapts speed of the leader; then gets very close and turns right to avoid collision; loses leader and stops	Follows closely; loses leader; but keeps driving for some time; turns slightly left and stops before the solid line	Follows closely; loses leader; but keeps driving for some time; turns slightly left and stops; eventually crashes
9950	Follows closely; adapts speed of the leader; then gets very close and turns right to avoid collision; loses leader and stops	Follows closely; loses leader; but keeps driving for some time; turns slightly left and stops before the solid line	Follows closely; loses leader; but keeps driving for some time; turns slightly left and stops; eventually crashes
9960	No movement; only slow down		
9970	No movement; only slow down		
9980	No movement; only slow down		
9990	No movement; only slow down		
9999	No movement; only slow down		

Figure A.5: Evaluations of config B for the reward weights comparison experiment.

Location 7:2	30 km/h - leader straight ahead	40 km/h	50 km/h	Total reward
640	Follows well, stops at some point and loses leader			-1.0686
690	No movement			0.3367
820	Follows well, slows down at some point, loses leader			-1.096
830	Follows very well until the intersection			-2.0685
840	Follows the leader, always a bit to the right, stays in the lane mostly, almost loses the leader, then catches up, then crashes at the intersection			-2.3171
850	Follows a few meters, then loses leader and stops			-0.308
860	Follows a few meters, then loses leader and stops			0.2077
870	Follows quite ok			0.3557
880	Follows well for a longer time, overtakes leader at when it was waiting for green light, loses it, then catches it again, gets confused at some point	Also follows, however, farther behind, loses leader	Loses leader early on	-0.9385
880		25 kmh: Follows well, loses leader at second intersection	35 kmh: Leader is too fast	
890	Starts driving, but stops after a few meters			-0.5188
900	Drives for a few meters, then loses leader and stops			-0.481
910	Follows the leader very well, stops at the first intersection, crashes into leader at second intersection	25kmh: overtakes leader	35 kmh: follows well, but crashes into leader when it stops at intersection	0.2604
920	No movement			0.3367
930	No action			0.3367
940	No action			0.3367
950	No movement			0.3543
960	Follows agressively at first, circles around leader, stops			0.1055

Figure A.6: Evaluations of the long-list for the fixed steering experiment.

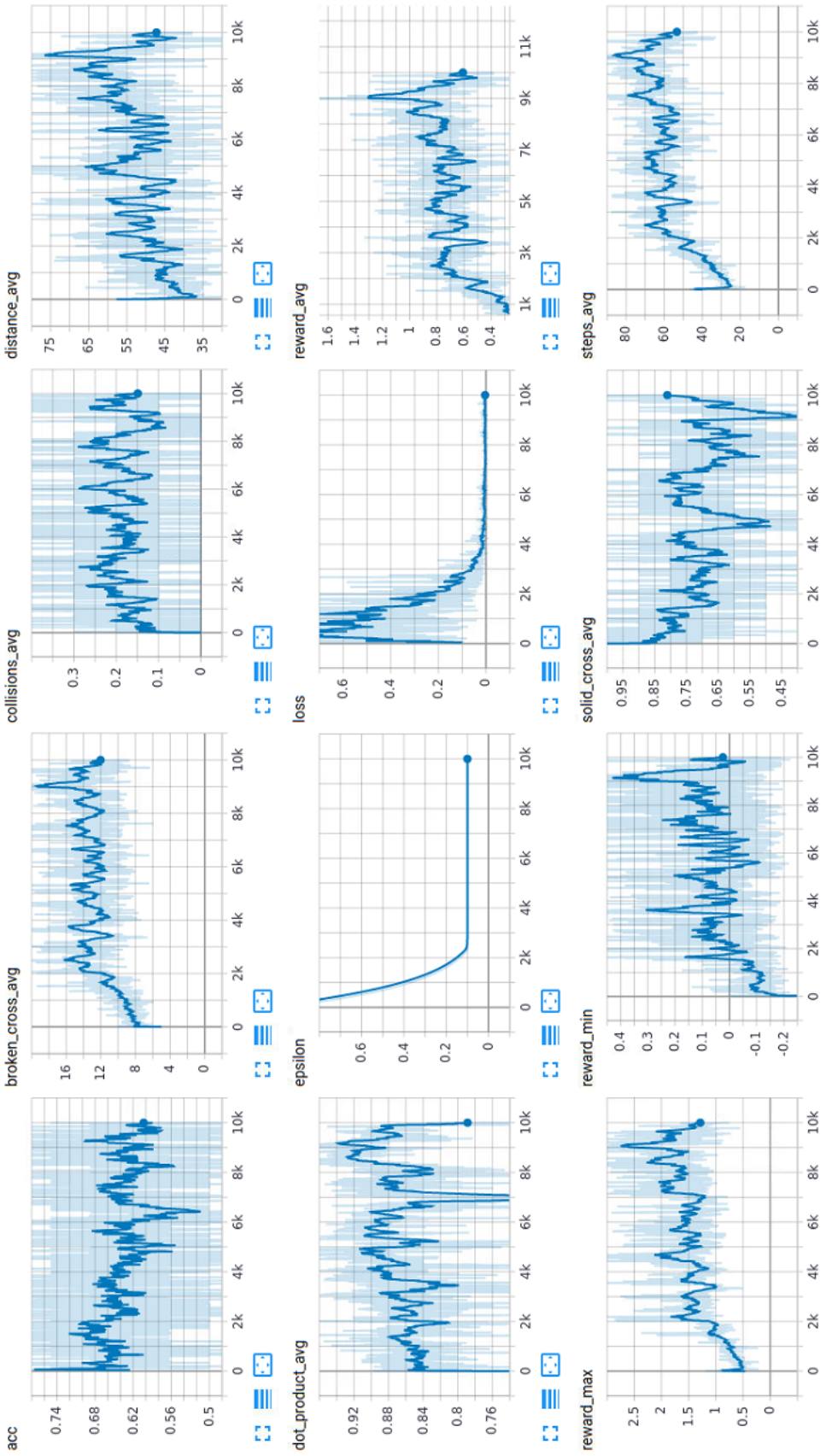

Figure A.7: Training monitoring in Tensorboard for the first platooning experiment.

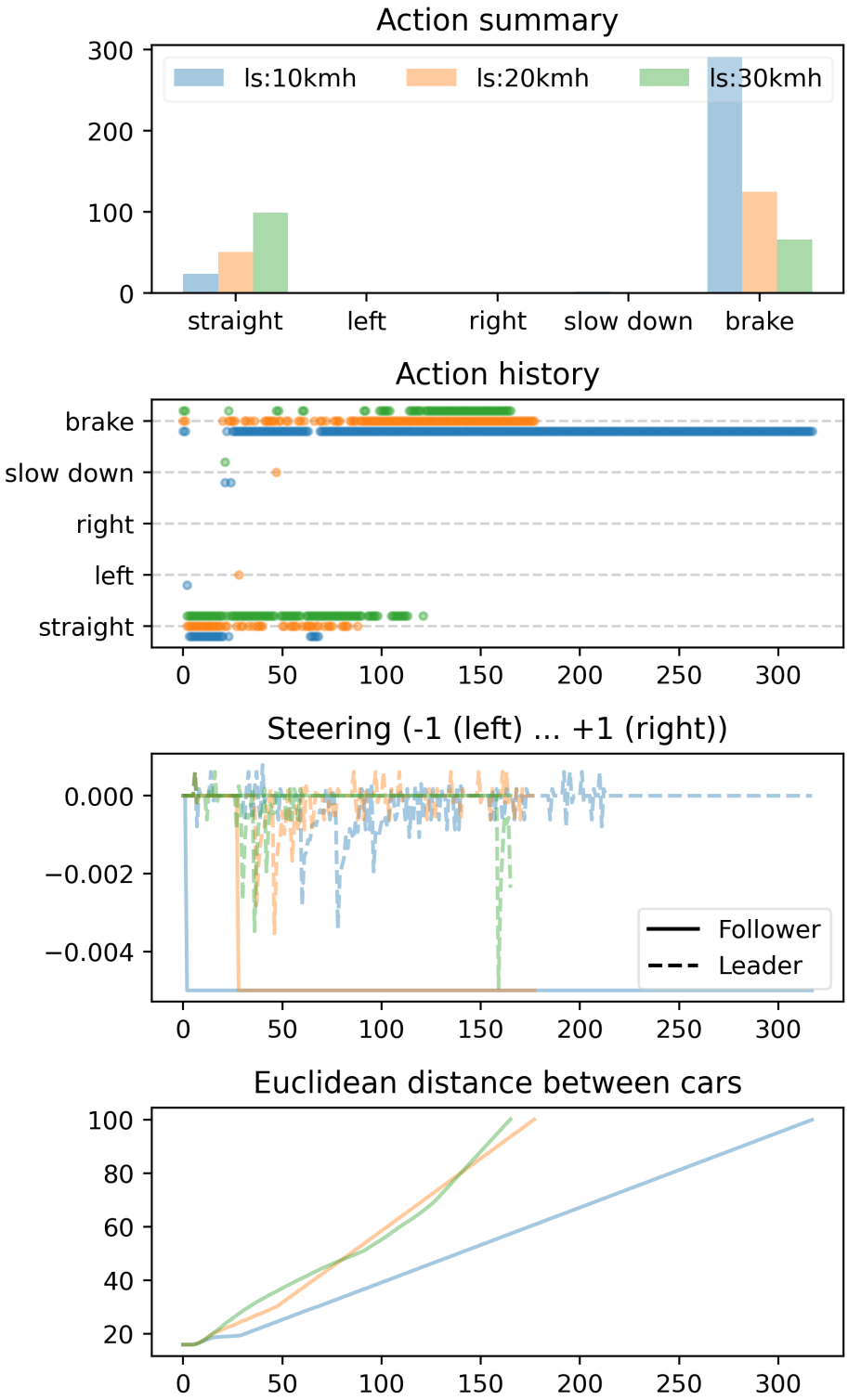

Figure A.8: Model ID 870 config B - different leader speeds experiment (part 1 of 3).

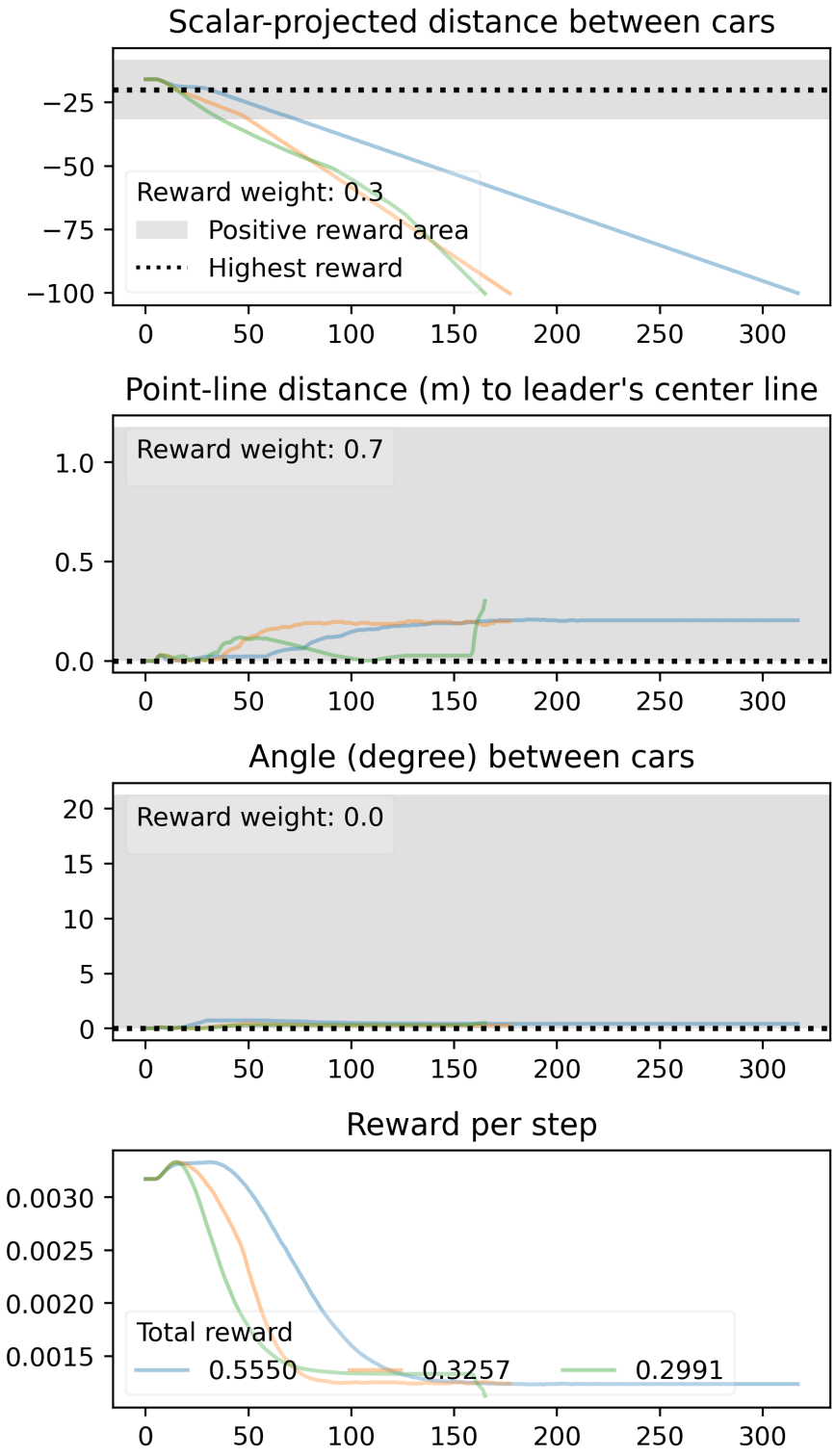

Figure A.9: Model ID 870 config B - different leader speeds experiment (part 2 of 3).

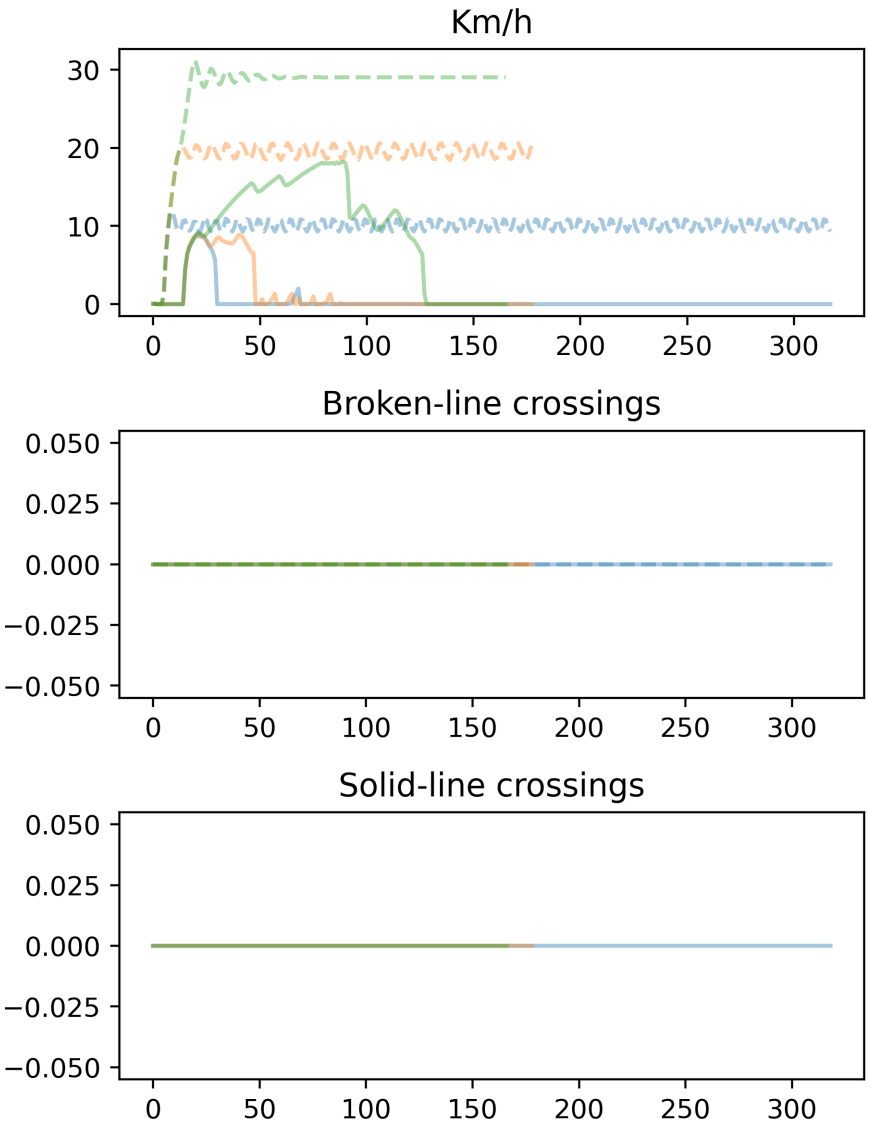

Figure A.10: Model ID 870 config B - different leader speeds experiment (part 3 of 3).

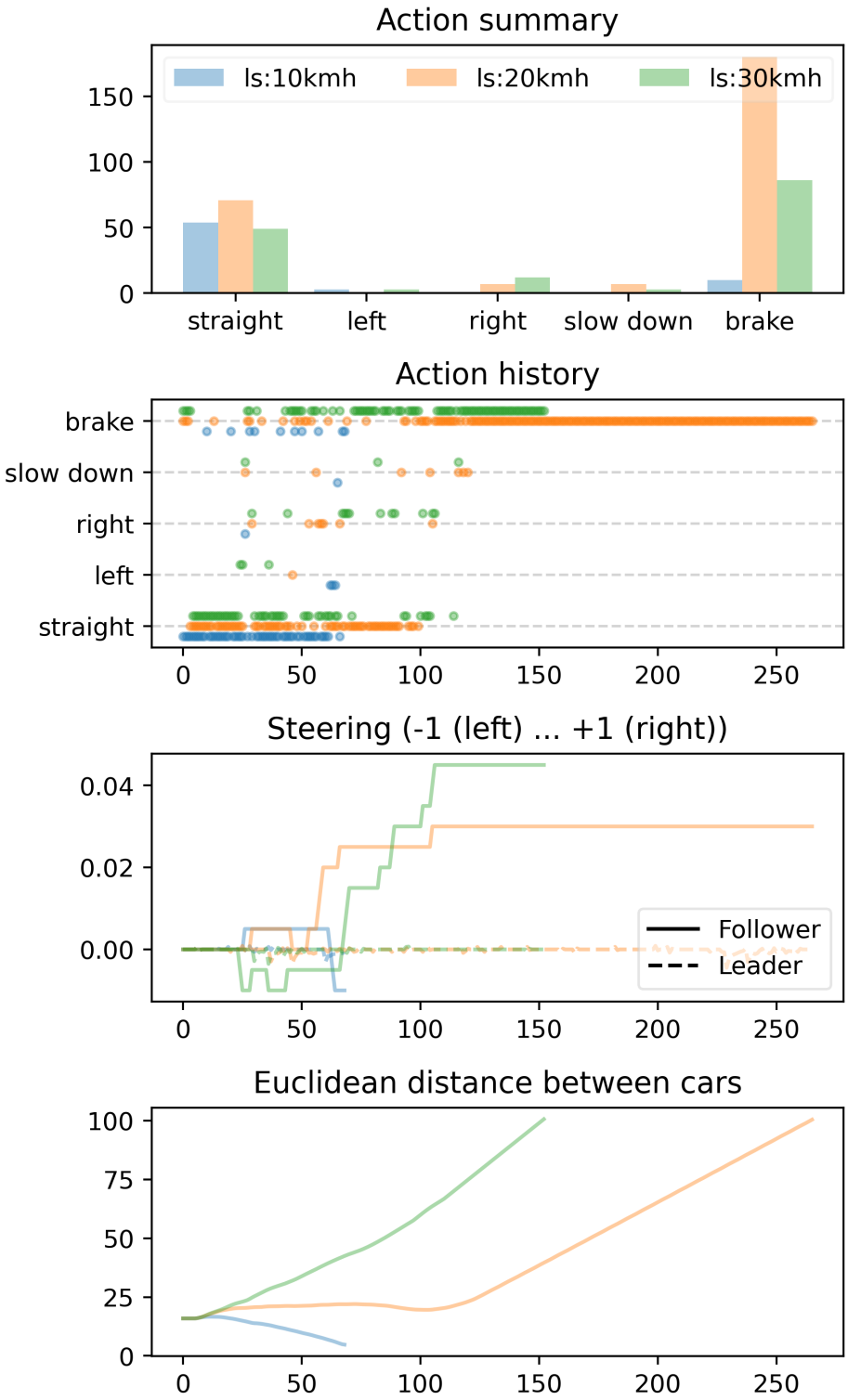

Figure A.11: Model ID 1040 config A - different leader speeds experiment (part 1 of 3).

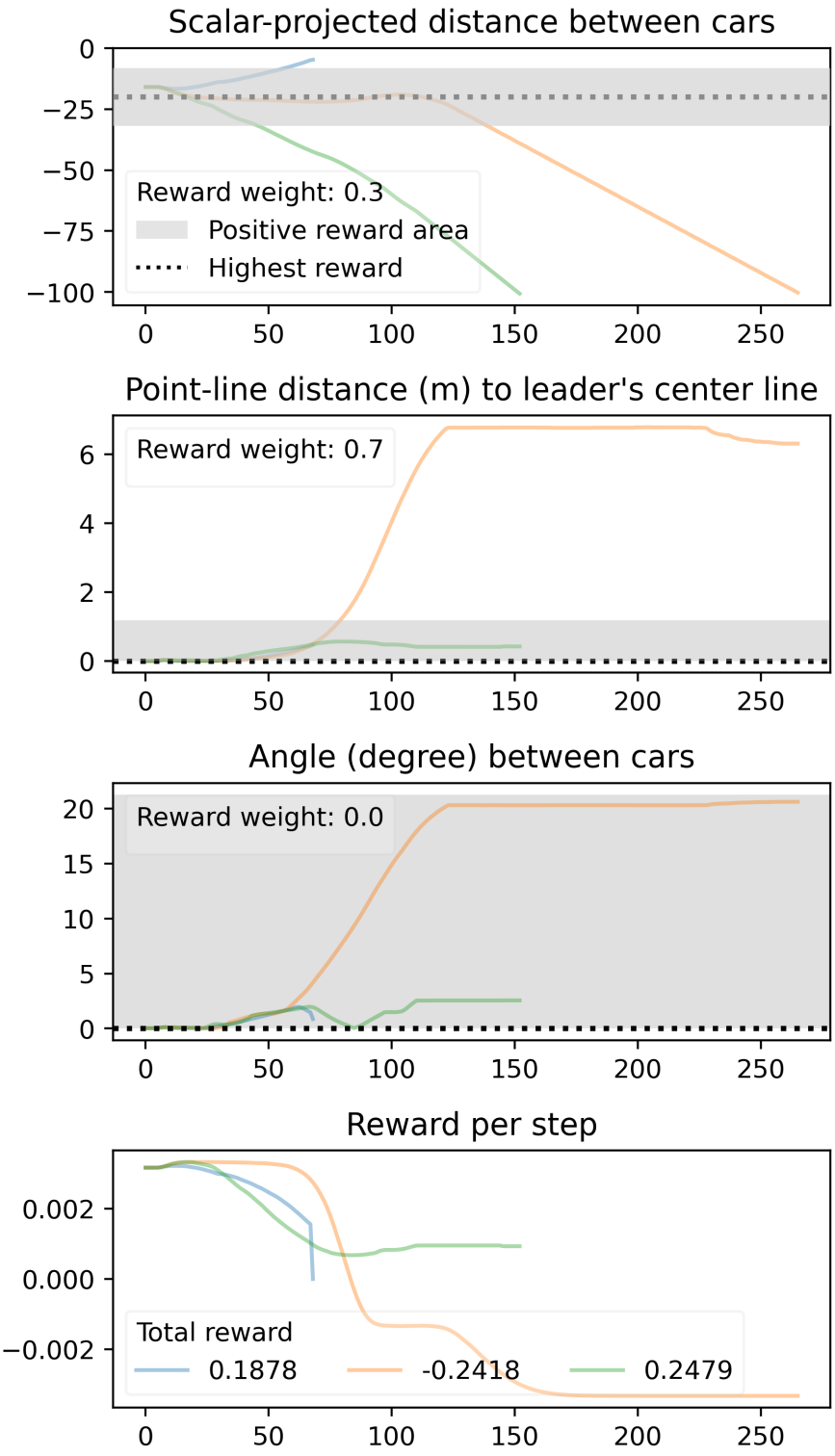

Figure A.12: Model ID 1040 config A - different leader speeds experiment (part 2 of 3).

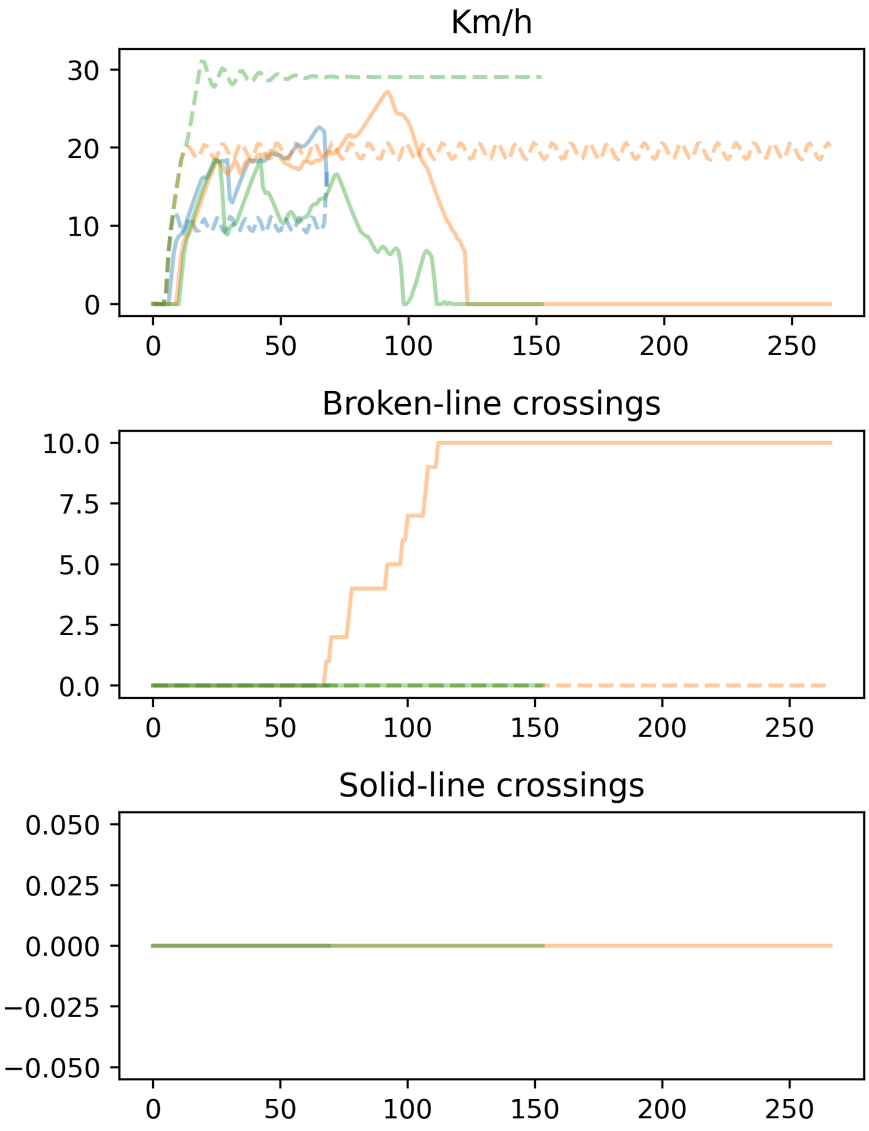

Figure A.13: Model ID 1040 config A - different leader speeds experiment (part 3 of 3).

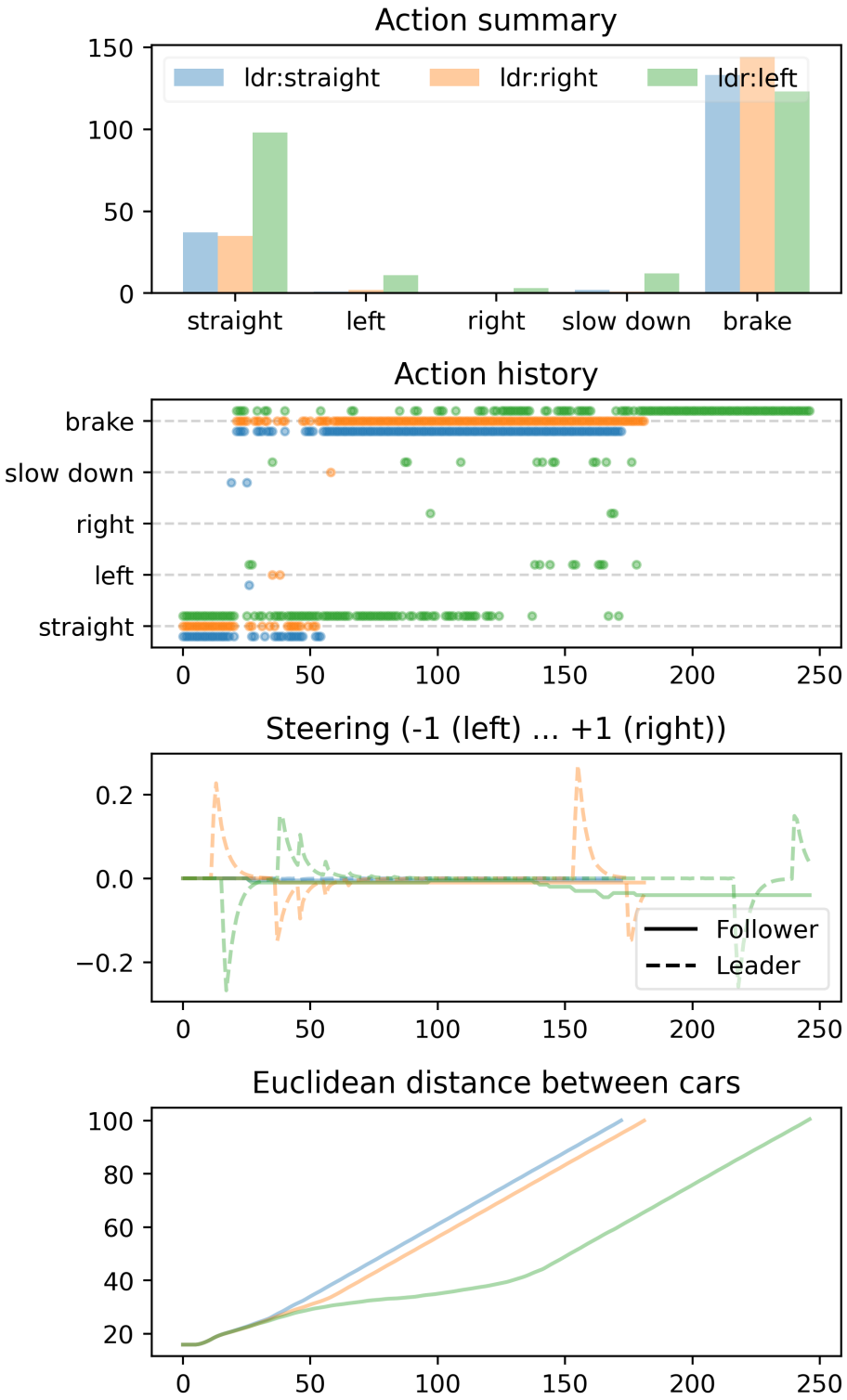

Figure A.14: Model ID 870 config B - different leader locations experiment (part 1 of 3).

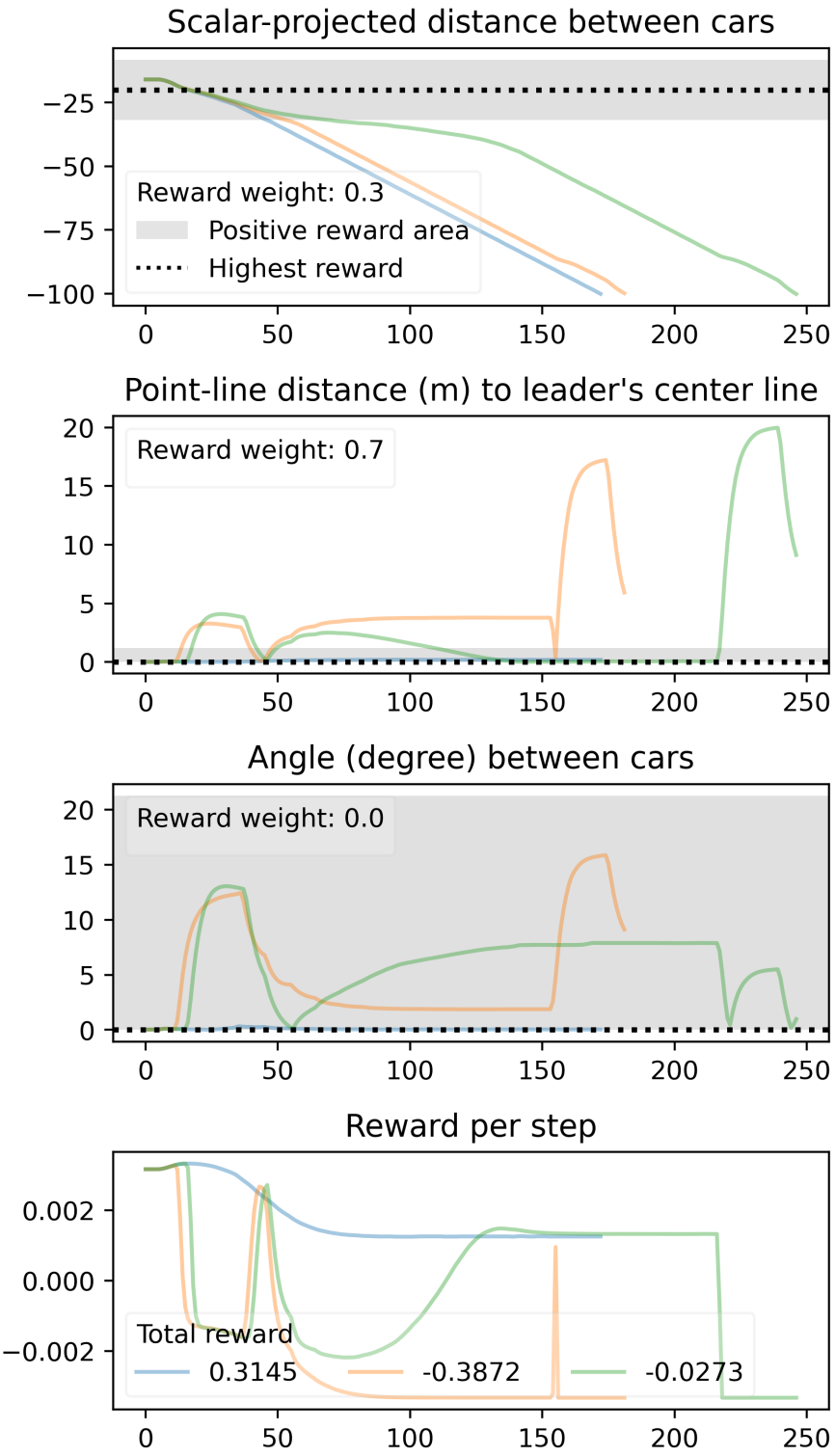

Figure A.15: Model ID 870 config B - different leader locations experiment (part 2 of 3).

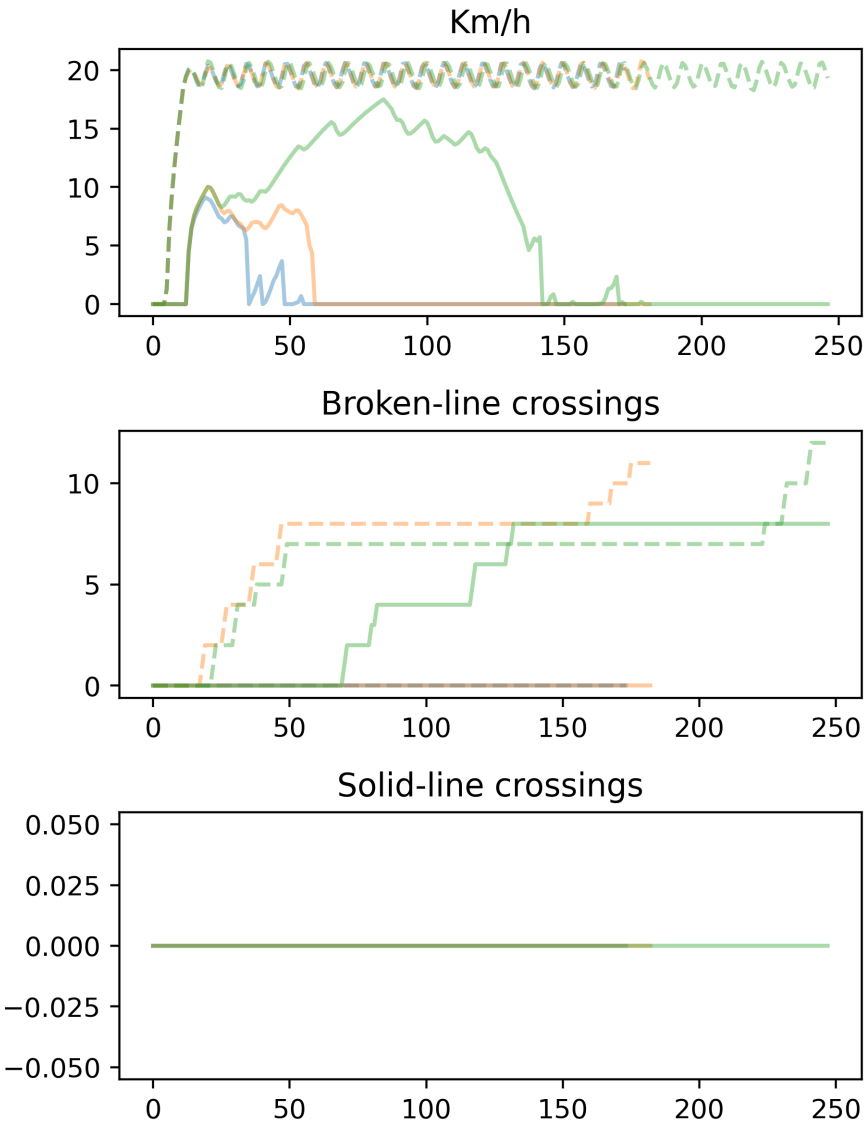

Figure A.16: Model ID 870 config B - different leader locations experiment (part 3 of 3).

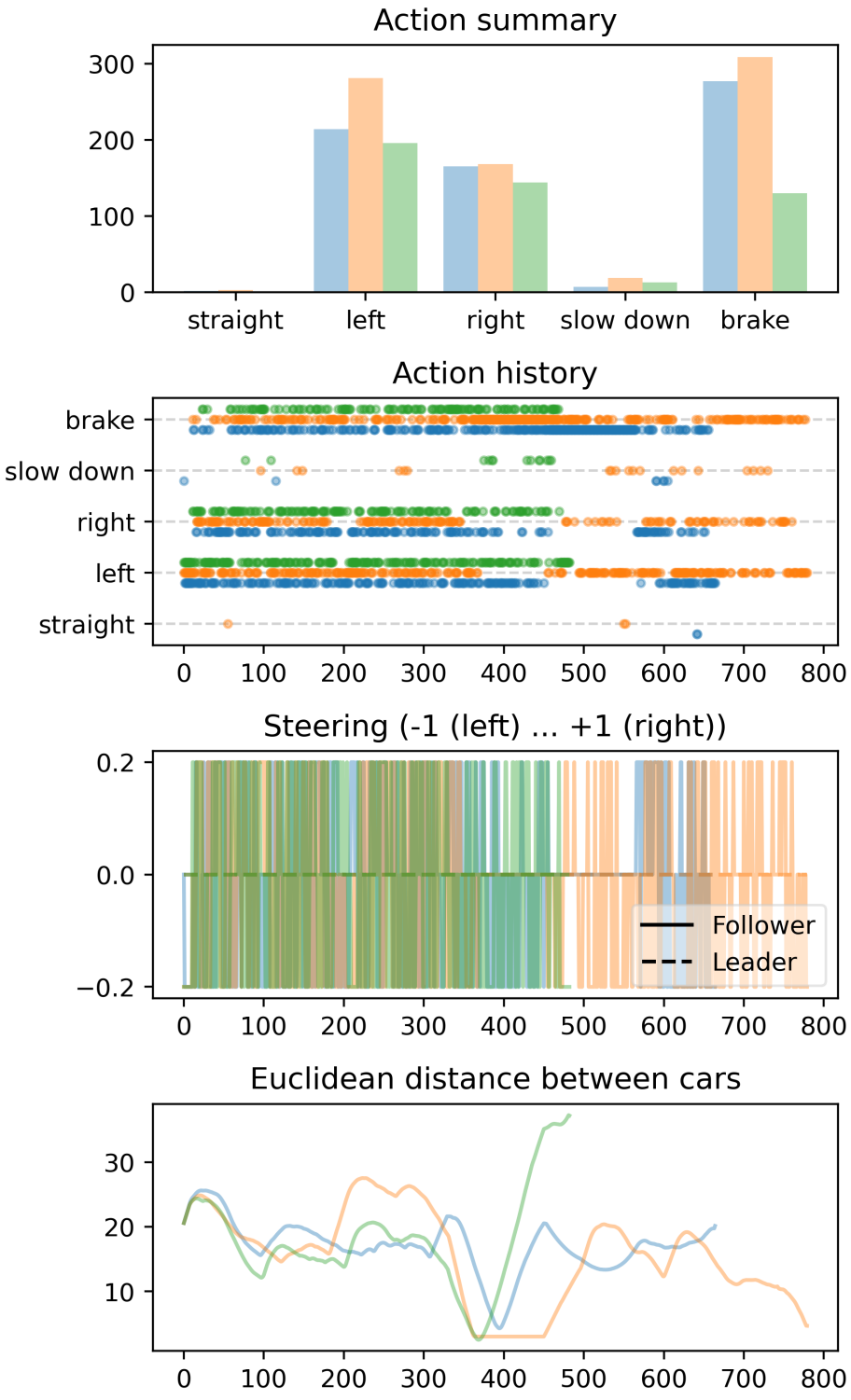

Figure A.17: Model ID 880 - unseen location (part 1 of 3).

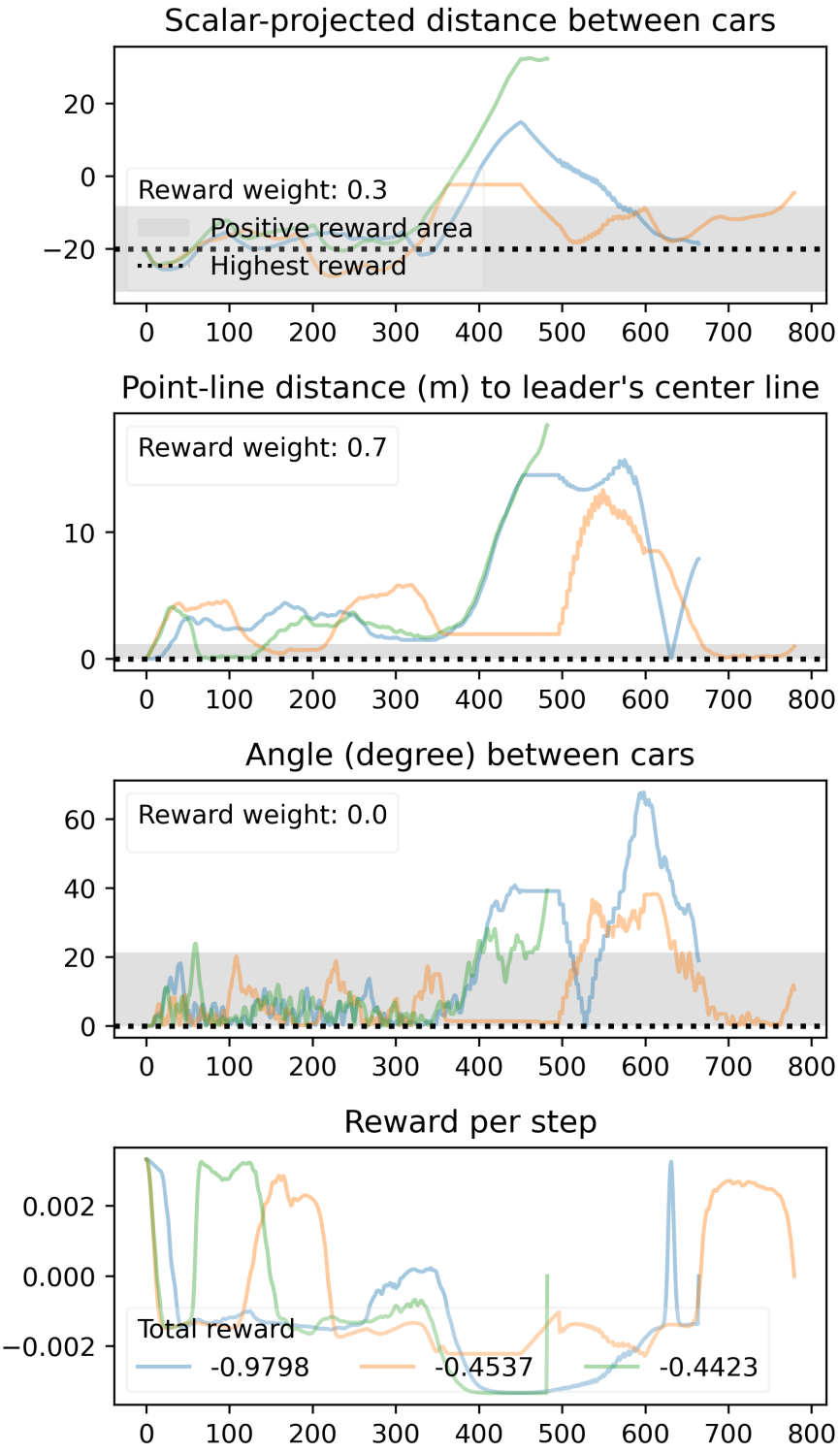

Figure A.18: Model ID 880 - unseen location (part 2 of 3).

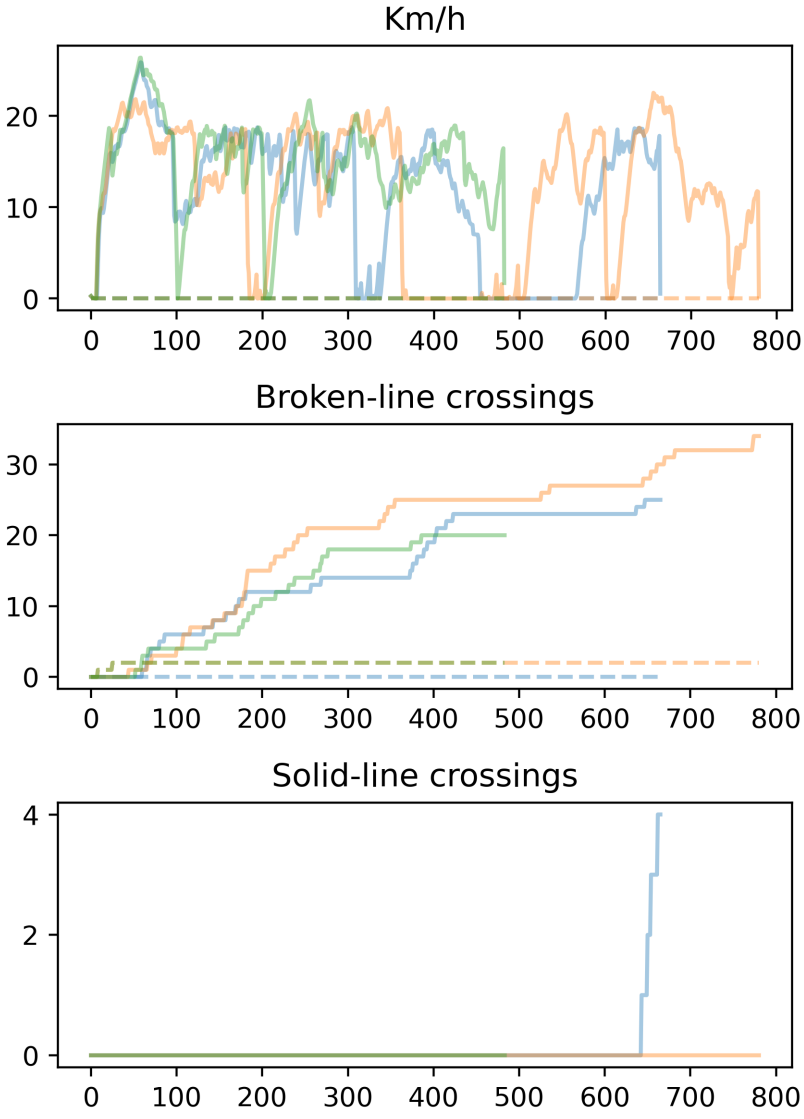

Figure A.19: Model ID 880 - unseen location (part 3 of 3).

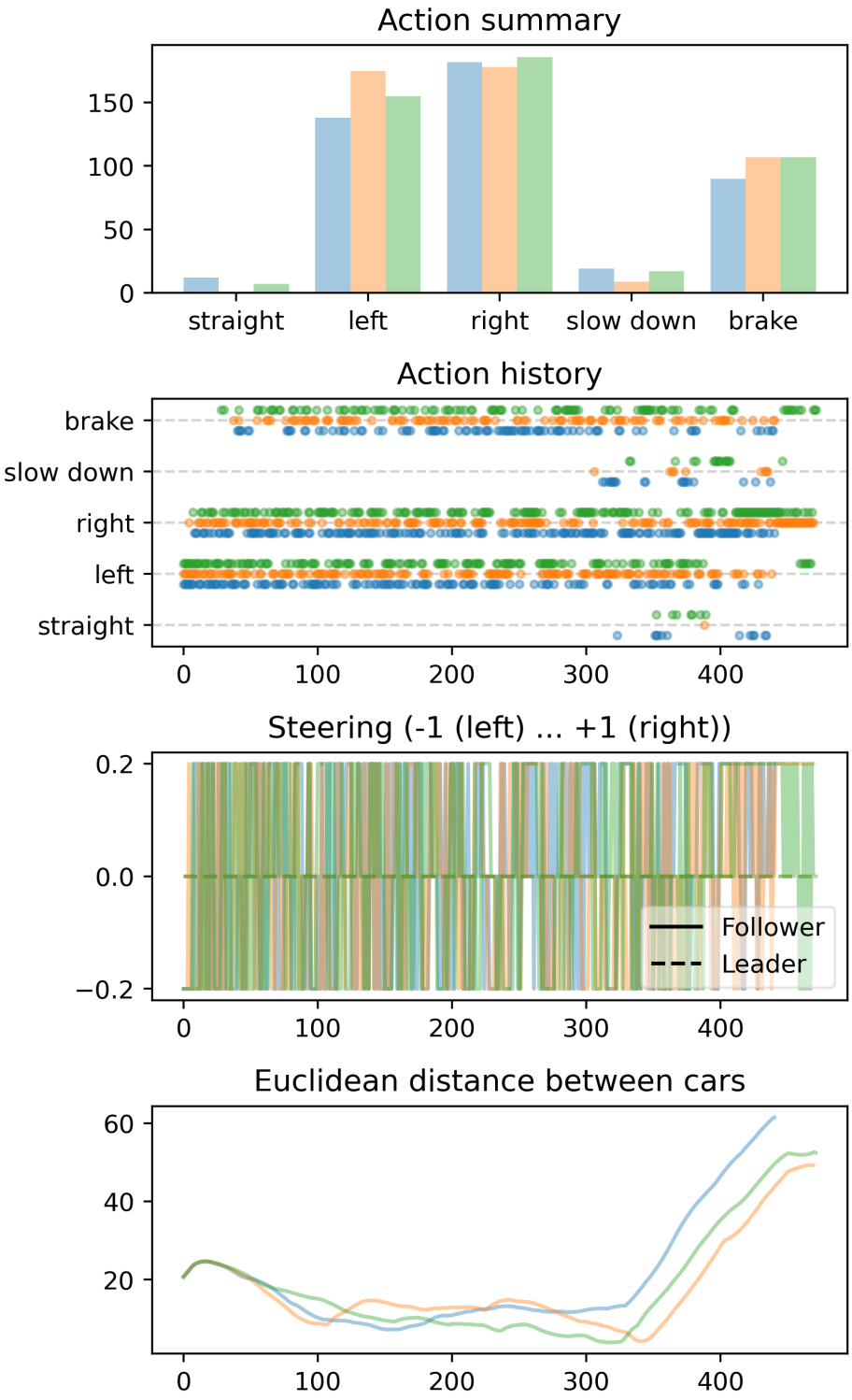

Figure A.20: Model ID 910 - unseen location (part 1 of 3).

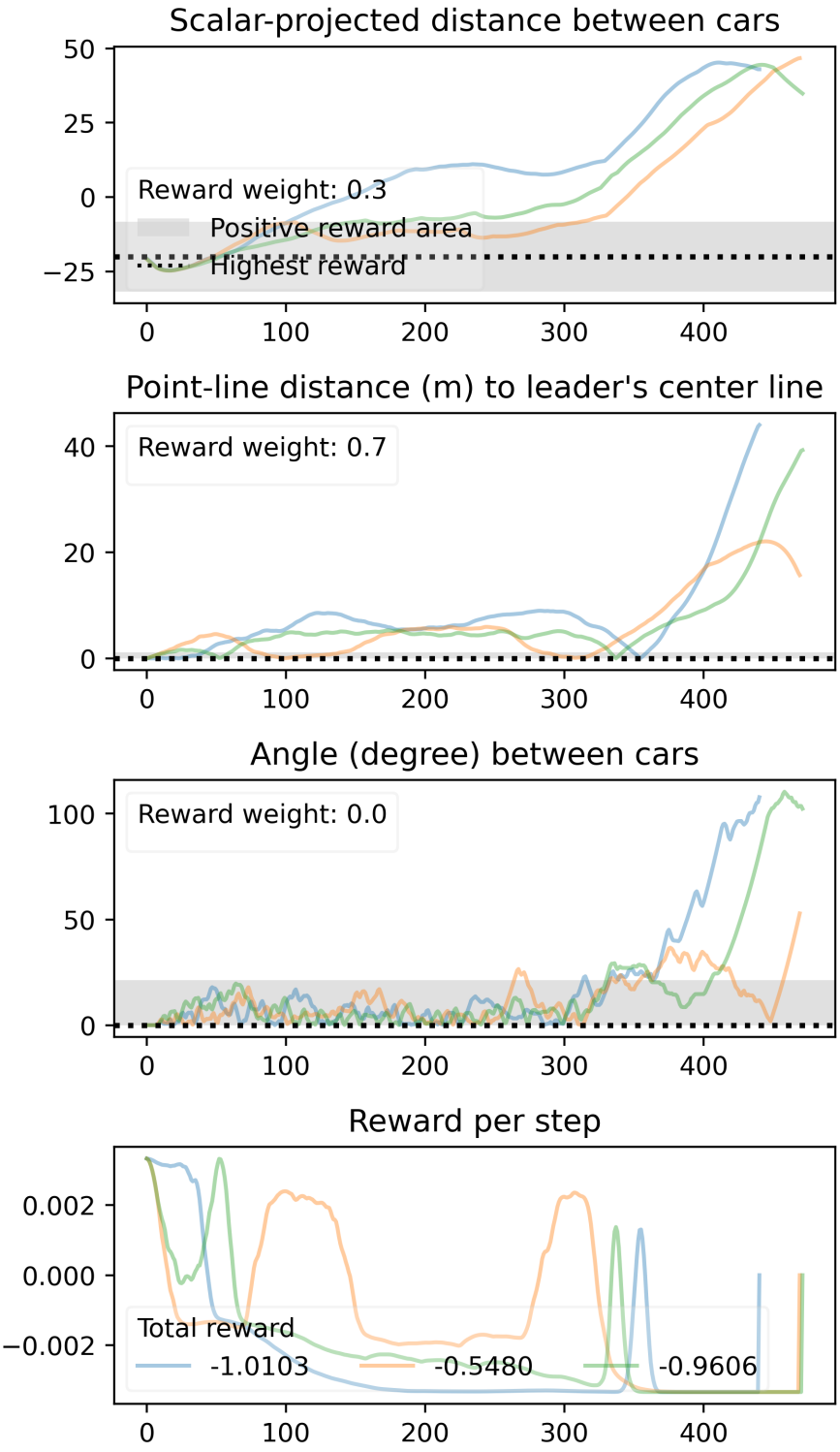

Figure A.21: Model ID 910 - unseen location (part 2 of 3).

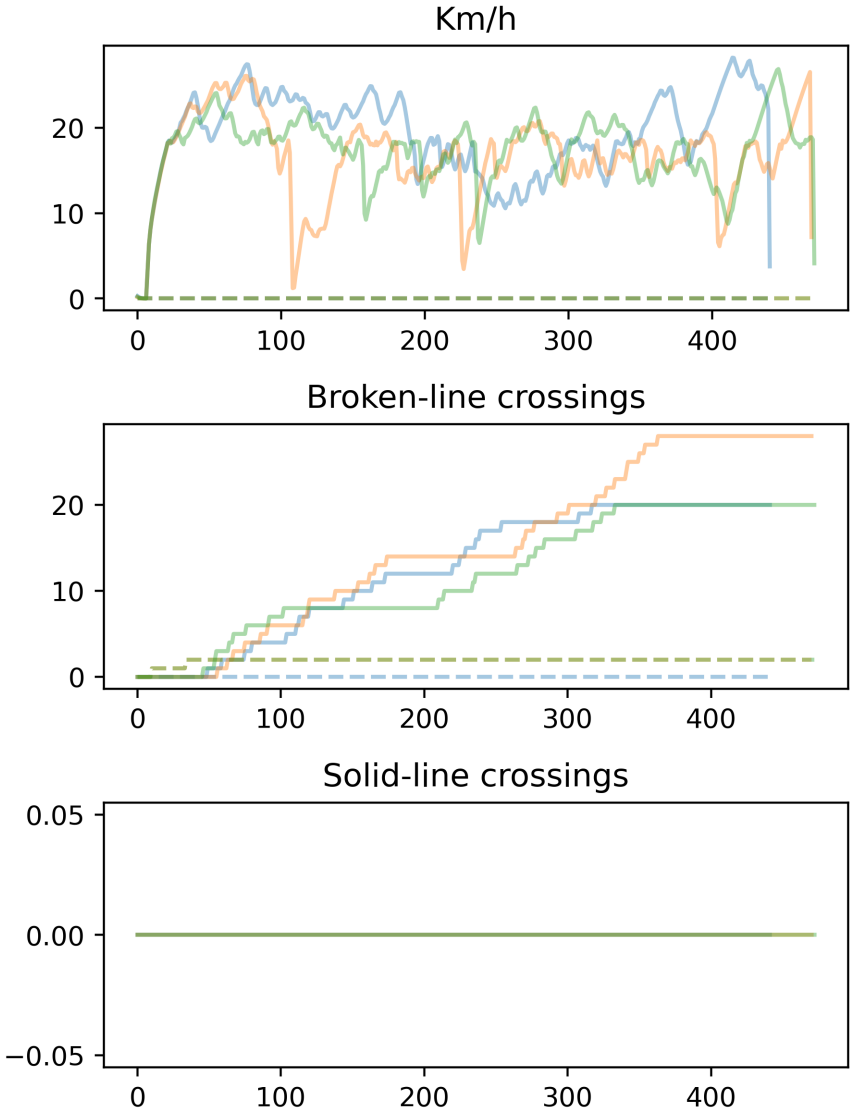

Figure A.22: Model ID 910 - unseen location (part 3 of 3).

Glossary

Lyapunov Lyapunov stability describes a state where the solutions that start out near an equilibrium point stay near it forever. [Wikipedia] 15

Markov Decision Process An MDP provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. [Wikipedia] 7, 15

Reinforcement Learning A machine learning paradigm where an agent acts in an environment according to a policy and receives a reward. Through this interaction, the agent learns the optimal policy. 4, 108

Abbreviations

CACC Cooperative Adaptive Cruise Control 1, 14, 16

RL Reinforcement Learning (see Reinforcement Learning) 4, 5

List of Figures

List of Tables

List of Listings

Bibliography

[1] “History of self-driving cars,” *Wikipedia*, Jun. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=History_of_self-driving_cars&oldid=1159337367 [Accessed: 2023-06-13]

[2] “Cruise Self Driving Cars | Autonomous Vehicles | Driverless Rides | Cruise.” [Online]. Available: https://getcruise.com/ [Accessed: 2023-06-13]

[3] PostAuto, “Autonomous driving.” [Online]. Available: https://www.postauto.ch/en/about-us-and-news/innovation/autonomous-driving [Accessed: 2023-06-13]

[4] “Der erste selbstfahrende Lieferdienst der Schweiz beginnt seine Testphase.” [Online]. Available: https://corporate.migros.ch/de/medien/mitteilungen/show/news/medienmitteilungen/2023/migronomous~id=3762259a-46fc-4939-b975-8195de7c3354~.html [Accessed: 2023-06-13]

[5] “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International.” [Online]. Available: https://www.sae.org/standards/content/j3016_202104 [Accessed: 2023-06-13]

[6] “Vehicular automation,” *Wikipedia*, Jun. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Vehicular_automation&oldid=1159626634#Challenges [Accessed: 2023-06-13]

[7] “World Automobile Production and Fleet, 1965-2021 | The Geography of Transport Systems,” Nov. 2017. [Online]. Available: https://transportgeography.org/contents/chapter5/road-transportation/automobile-production-fleet-world/ [Accessed: 2023-06-13]

[8] INRIX, “Scorecard.” [Online]. Available: https://inrix.com/scorecard/ [Accessed: 2023-06-13]

[9] "Platoon (automobile)," *Wikipedia*, Jun. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Platoon_(automobile)&oldid=1159214158 [Accessed: 2023-06-13]

[10] Z. Wang, G. Wu, and M. J. Barth, "A Review on Cooperative Adaptive Cruise Control (CACC) Systems: Architectures, Controls, and Applications," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 2884–2891.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, Oct. 2018.

[12] "The Reinforcement Learning Framework - Hugging Face Deep RL Course." [Online]. Available: https://huggingface.co/learn/deep-rl-course/unit1/rl-framework [Accessed: 2023-05-26]

[13] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to Drive in a Day," Sep. 2018. [Online]. Available: http://arxiv.org/abs/1807.00412 [Accessed: 2023-02-28]

[14] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Duerr, "Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, Jul. 2021. [Online]. Available: http://arxiv.org/abs/2008.07971 [Accessed: 2023-03-01]

[15] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," Jun. 2017. [Online]. Available: http://arxiv.org/abs/1706.03741 [Accessed: 2023-05-30]

[16] R. E. Wang, J. C. Kew, D. Lee, T.-W. E. Lee, T. Zhang, B. Ichter, J. Tan, and A. Faust, "Model-based Reinforcement Learning for Decentralized Multiagent Rendezvous," Nov. 2020. [Online]. Available: http://arxiv.org/abs/2003.06906 [Accessed: 2023-02-09]

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Dec. 2013. [Online]. Available: http://arxiv.org/abs/1312.5602 [Accessed: 2023-05-28]

[18] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017. [Online]. Available: https://www.nature.com/articles/nature24270/ [Accessed: 2023-05-28]

[19] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," Dec. 2019. [Online]. Available: http://arxiv.org/abs/1912.06680 [Accessed: 2023-05-28]

[20] S. Booth, W. B. Knox, J. Shah, S. Niekum, P. Stone, and A. Allievi, "The Perils of Trial-and-Error Reward Design: Misdesign through Overfitting and Invalid Task Specifications," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, Washington, D.C., Feb. 2023.

[21] CARLA. Team, "CARLA Simulator." [Online]. Available: http://carla.org// [Accessed: 2023-05-26]

[22] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.03938 [Accessed: 2023-02-28]

[23] "Donkey Simulator. - Donkey Car." [Online]. Available: https://docs.donkeycar.com/guide/deep_learning/simulator/ [Accessed: 2023-06-11]

[24] "Duckietown: Learning Autonomy | Teach, learn robotics and AI." [Online]. Available: https://www.duckietown.org [Accessed: 2023-06-11]

[25] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S.-Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H.-C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1497–1504.

[26] "FLOW." [Online]. Available: https://flow-project.github.io/ [Accessed: 2023-06-11]

[27] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: A Modular Learning Framework for Mixed Autonomy Traffic," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1270–1286, Apr. 2022. [Online]. Available: http://arxiv.org/abs/1710.05465 [Accessed: 2023-03-02]

[28] P. Palanisamy, "MACAD-Gym: Multi-Agent Reinforcement Learning for Connected Autonomous Driving," IEEE, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9207663 [Accessed: 2023-06-11]

[29] ——, “Multi-Agent Connected Autonomous Driving using Deep Reinforcement Learning,” Nov. 2019. [Online]. Available: http://arxiv.org/abs/1911.04175 [Accessed: 2023-03-25]

[30] “MetaDrive.” [Online]. Available: https://metadriverse.github.io/metadrive/ [Accessed: 2023-06-11]

[31] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou, “MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning,” Jul. 2022. [Online]. Available: http://arxiv.org/abs/2109.12674 [Accessed: 2023-06-11]

[32] “AirSim.” [Online]. Available: https://microsoft.github.io/AirSim/ [Accessed: 2023-06-11]

[33] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” Jul. 2017. [Online]. Available: http://arxiv.org/abs/1705.05065 [Accessed: 2023-06-11]

[34] “Facebookresearch/nocturne: A data-driven, fast driving simulator for multi-agent coordination under partial observability.” [Online]. Available: https://github.com/facebookresearch/nocturne [Accessed: 2023-06-11]

[35] E. Vinitsky, N. Lichtlé, X. Yang, B. Amos, and J. Foerster, “Nocturne: A scalable driving benchmark for bringing multi-agent learning one step closer to the real world,” Feb. 2023. [Online]. Available: http://arxiv.org/abs/2206.09889 [Accessed: 2023-06-11]

[36] “NVIDIA DRIVE Sim,” Apr. 2021. [Online]. Available: https://developer.nvidia.com/drive/simulation [Accessed: 2023-06-11]

[37] “Plexe.” [Online]. Available: http://plexe.car2x.org/features/ [Accessed: 2023-06-12]

[38] M. Segata, R. Lo Cigno, T. Hardes, J. Heinovski, M. Schettler, B. Bloessl, C. Sommer, and F. Dressler, “Multi-Technology Cooperative Driving: An Analysis Based on PLEXE,” *IEEE Transactions on Mobile Computing*, pp. 1–1, 2022.

[39] “SUMMIT.” [Online]. Available: https://adacompnus.github.io/summit-docs/ [Accessed: 2023-06-11]

[40] P. Cai, Y. Lee, Y. Luo, and D. Hsu, “SUMMIT: A Simulator for Urban Driving in Massive Mixed Traffic,” Mar. 2020. [Online]. Available: http://arxiv.org/abs/1911.04074 [Accessed: 2023-06-11]

[41] “Eclipse SUMO - Simulation of Urban MObility.” [Online]. Available: https://www.eclipse.org/sumo/ [Accessed: 2023-06-11]

[42] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic Traffic Simulation using SUMO," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 2575–2582.

[43] "Veins." [Online]. Available: http://veins.car2x.org/ [Accessed: 2023-06-12]

[44] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011.

[45] "Welcome to VISTA's documentation! — VISTA Simulator documentation." [Online]. Available: https://vista.csail.mit.edu/ [Accessed: 2023-06-11]

[46] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, Apr. 2020.

[47] "Aerial Autonomy: Project AirSim." [Online]. Available: https://www.microsoft.com/en-us/AI/autonomous-systems-project-airsim [Accessed: 2023-06-11]

[48] "CARLA Autonomous Driving Leaderboard," Feb. 2020. [Online]. Available: http://leaderboard.carla.org/ [Accessed: 2023-06-12]

[49] H. Shao, L. Wang, R. Chen, S. L. Waslander, H. Li, and Y. Liu, "ReasonNet: End-to-End Driving With Temporal and Global Reasoning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 723–13 733. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2023/html/Shao_ReasonNet_End-to-End_Driving_With_Temporal_and_Global_Reasoning_CVPR_2023_paper.html [Accessed: 2023-06-12]

[50] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-Enhanced Autonomous Driving Using Interpretable Sensor Fusion Transformer," Dec. 2022. [Online]. Available: http://arxiv.org/abs/2207.14024 [Accessed: 2023-06-12]

[51] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao, "Trajectory-guided Control Prediction for End-to-end Autonomous Driving: A Simple yet Strong Baseline," Oct. 2022. [Online]. Available: http://arxiv.org/abs/2206.08129 [Accessed: 2023-06-12]

[52] Ó. Pérez-Gil, R. Barea, E. López-Guillén, L. M. Bergasa, C. Gómez-Huélamo, R. Gutiérrez, and A. Díaz-Díaz, "Deep reinforcement learning based control for

Autonomous Vehicles in CARLA," *Multimedia Tools and Applications*, vol. 81, no. 3, pp. 3553–3576, Jan. 2022. [Online]. Available: https://doi.org/10.1007/s11042-021-11437-3 [Accessed: 2023-02-28]

[53] K. C. Dey, L. Yan, X. Wang, Y. Wang, H. Shen, M. Chowdhury, L. Yu, C. Qiu, and V. Soundararaj, "A Review of Communication, Driver Characteristics, and Controls Aspects of Cooperative Adaptive Cruise Control (CACC)," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 491–509, Feb. 2016.

[54] Y. Zheng, S. E. Li, J. Wang, L. Y. Wang, and K. Li, "Influence of information flow topology on closed-loop stability of vehicle platoon with rigid formation," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2014, pp. 2094–2100.

[55] G. J. L. Naus, R. P. A. Vugts, J. Ploeg, M. J. G. van de Molengraft, and M. Steinbuch, "String-Stable CACC Design and Experimental Validation: A Frequency-Domain Approach," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 9, pp. 4268–4279, Nov. 2010.

[56] B. Besselink and K. H. Johansson, "String stability and a delay-based spacing policy for vehicle platoons subject to disturbances," Feb. 2017. [Online]. Available: http://arxiv.org/abs/1702.01031 [Accessed: 2023-02-28]

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

[58] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, "Safe Model-based Reinforcement Learning with Stability Guarantees," *ArXiv*, May 2017. [Online]. Available: https://www.semanticscholar.org/paper/Safe-Model-based-Reinforcement-Learning-with-Berkenkamp-Turchetta/88880d88073a99107bbc009c9f4a4197562e1e44 [Accessed: 2023-02-16]

[59] Y. Chow, O. Nachum, E. A. Duéñez-Guzmán, and M. Ghavamzadeh, "A Lyapunov-based Approach to Safe Reinforcement Learning," in *Neural Information Processing Systems*, May 2018. [Online]. Available: https://www.semanticscholar.org/paper/A-Lyapunov-based-Approach-to-Safe-Reinforcement-Chow-Nachum/65fb1b37c41902793ac65db3532a6e51631a9aff [Accessed: 2023-02-16]

[60] T. J. Perkins and A. G. Barto, "Lyapunov Design for Safe Reinforcement Learning," 2002.

[61] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, "End-to-End Urban Driving by Imitating a Reinforcement Learning Coach," Oct. 2021. [Online]. Available: http://arxiv.org/abs/2108.08265 [Accessed: 2023-03-09]

[62] E. Vinitsky, A. Kreidieh, L. L. Flem, N. Kheterpal, K. Jang, F. Wu, R. Liaw, E. Liang, and A. Bayen, "Benchmarks for reinforcement learning in mixed-autonomy traffic," in *Conference on Robot Learning*, Oct. 2018. [Online]. Available: https://www.semanticscholar.org/paper/Benchmarks-for-reinforcement-learning-in-traffic-Vinitsky-Kreidieh/6865bc5ff0d28da7d2857f5821afc645c1e80807 [Accessed: 2023-02-28]

[63] E. Vinitsky, K. Parvate, A. Kreidieh, C. Wu, and A. Bayen, "Lagrangian Control through Deep-RL: Applications to Bottleneck Decongestion," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 759–765.

[64] C.-C. Yen, H. Gao, and M. Zhang, "Deep Reinforcement Learning Based Platooning Control for Travel Delay and Fuel Optimization," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. Macau, China: IEEE, Oct. 2022, pp. 737–742. [Online]. Available: https://ieeexplore.ieee.org/document/9922550/ [Accessed: 2023-06-11]

[65] S. B. Prathiba, G. Raja, K. Dev, N. Kumar, and M. Guizani, "A Hybrid Deep Reinforcement Learning For Autonomous Vehicles Smart-Platooning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13 340–13 350, Dec. 2021.

[66] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep Reinforcement Learning for Autonomous Driving: A Survey," Jan. 2021. [Online]. Available: http://arxiv.org/abs/2002.00444 [Accessed: 2023-06-11]

[67] S. Ergün, "A study on multi-agent reinforcement learning for autonomous distribution vehicles," *Iran Journal of Computer Science*, Mar. 2023. [Online]. Available: https://doi.org/10.1007/s42044-023-00140-1 [Accessed: 2023-06-11]

[68] K. Jang, E. Vinitsky, B. Chalaki, B. Remer, L. Beaver, A. A. Malikopoulos, and A. Bayen, "Simulation to scaled city: Zero-shot policy transfer for traffic control via autonomous vehicles," *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pp. 291–300, Apr. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3302509.3313784 [Accessed: 2023-02-28]

[69] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," Dec. 2015. [Online]. Available: http://arxiv.org/abs/1509.06461 [Accessed: 2023-06-13]

[70] H. Shengyi, D. Julien, Rousslan Fernand, R. Antonin, K. Anssi, and W. Weixun, "The 37 Implementation Details of Proximal Policy Optimization," in *ICLR Blog Track*, 2022. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[71] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *ArXiv*, Dec. 2013. [Online]. Available: https://www.semanticscholar.org/paper/Playing-Atari-with-Deep-Reinforcement-Learning-Mnih-Kavukcuoglu/2319a491378867c7049b3da055c5df60e1671158 [Accessed: 2023-02-16]

[72] J. TORRES.AI, "Deep Q-Network (DQN)," May 2021. [Online]. Available: https://towardsdatascience.com/deep-q-network-dqn-i-bce08bdf2af [Accessed: 2023-06-13]

[73] A. Bäuerle, C. van Onzenoodt, and T. Ropinski, "Net2Vis – A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 6, pp. 2980–2991, Jun. 2021.

[74] "Distance from a point to a line," *Wikipedia*, Jun. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Distance_from_a_point_to_a_line&oldid=1158874196#Another_vector_formulation [Accessed: 2023-06-13]

[75] "Vector projection," *Wikipedia*, Apr. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Vector_projection&oldid=1152418040 [Accessed: 2023-06-13]

[76] "Apodization Function – from Wolfram MathWorld." [Online]. Available: https://mathworld.wolfram.com/ApodizationFunction.html [Accessed: 2023-06-13]

[77] "Feature scaling," *Wikipedia*, May 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Feature_scaling&oldid=1157241504#Rescaling_(min-max_normalization) [Accessed: 2023-06-13]

[78] "Unreal Engine | The most powerful real-time 3D creation tool." [Online]. Available: https://www.unrealengine.com/en-US [Accessed: 2023-06-12]

[79] "ASAM OpenDRIVE." [Online]. Available: https://www.asam.net/standards/detail/opendrive/ [Accessed: 2023-06-12]

[80] "RLlib - Scalable, state of the art reinforcement learning in Python." [Online]. Available: https://www.ray.io/rllib [Accessed: 2023-06-12]

[81] "NVIDIA DGX -2 for Complex AI Challenges." [Online]. Available: https://www.nvidia.com/en-gb/data-center/dgx-2/ [Accessed: 2023-06-13]

[82] "Apptainer." [Online]. Available: https://apptainer.org/ [Accessed: 2023-06-09]

[83] "TensorFlow." [Online]. Available: https://www.tensorflow.org/ [Accessed: 2023-06-10]

[84] "TensorFlow," *Wikipedia*, May 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=TensorFlow&oldid=1154948861 [Accessed: 2023-06-10]

[85] "Python Programming Tutorials." [Online]. Available: https://pythonprogramming.net/introduction-self-driving-autonomous-cars-carla-python/ [Accessed: 2023-06-04]

[86] "Gymnasium Cart Pole." [Online]. Available: https://gymnasium.farama.org/environments/classic_control/cart_pole.html [Accessed: 2023-06-16]

[87] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html [Accessed: 2023-05-26]

[88] K. Team, "Keras documentation: Xception." [Online]. Available: https://keras.io/api/applications/xception/ [Accessed: 2023-06-04]