Aalto-yliopisto
Aalto-universitetet
Aalto University

Master's Programme in ICT Innovation

School of Electrical Engineering

Autonomous System

Cooperative Driving Behavior Model for Intelligent Traffic Generation

Theocharis Triantafyllidis

Master's Thesis

2021

Aalto-yliopisto
Aalto-universitetet
Aalto University

Author Theocharis Triantafyllidis

Title of thesis Cooperative Driving Behavior Model for Intelligent Traffic Generation

Programme Master's Programme in ICT Innovation

Major Autonomous Systems

Thesis supervisor Istenes Zoltán, Phd

Thesis advisor(s) Quan Zhou, Associate Professor

Collaborative partner Lindenmaier Autonomous Technologies

Date 16. 07. 2021 **Number of pages** 51 **Language** English

Abstract

Intelligent Test Vehicles are following pre-defined trajectory based on the actual scenario. Just the slightest offset from the ideal scenario and trajectory plan might lead to the situation where the desired outcome is unreachable. In order to be able to compensate such delays and offsets a cooperative behavior required between the test vehicles to adapt to new situations and compensate. ITVs are not able to make decisions that would lead to another scenario – their maneuver space is limited. But ITVs should be able to react on the decisions of the Vehicle Under Test (VUT) and they might be able to compensate in order to avoid accidents.

Keywords Autonomous Vehicles, Intelligent Test Vehicles (ITVs), Cooperative Driving Model (CDM), OpenDrive, OpenScenario

Topic Declaration

Intelligent Test Vehicles (ITVs) are following a predefined trajectory based on the actual scenario. Just the slightest offset from the ideal scenario and trajectory plan might lead to the situation where the desired outcome is unreachable. To compensate for such delays and offsets, cooperative behavior is required between the test vehicles to adapt to new conditions and compensate. ITVs cannot make decisions that would lead to another scenario – their maneuver space is limited. But ITVs should be able to react on the Vehicle Under Test (VUT) decisions, and they might be able to compensate to avoid accidents.

Intelligent Test Vehicles are used to generate real traffic situations described by a scenario template (phrased by OpenScenario). Ideally, on each ITVs we should have the same algorithm running. Trade-offs exist between the different technological approaches(Static decentralized evaluation, Dynamic centralized evaluation, Cooperative evaluation). The primary goal is to keep the infrastructure complexity and cost low because system adaptation to every proving ground facility would introduce large expenses. The facility might not necessarily be a partner on that.

I wish to build up the environment for ITVs, define control parameters, timing requirements, and required lateral and longitudinal precision. Moreover, I would like to investigate meaningful and realistic results about reinforcement learning methods and multiple agent techniques, and finally deliver a rough definition of a Cooperative Driving Model (CDM).

Generally, we have a simple analytical method for vehicle control, Proportional Integral Derivative (PID) controller, or Linear Quadratic Regulator (LQR) controller, widely implemented and used universally in the past years. Whereas Deep Reinforcement Learning (DRL) techniques are emerging nowadays, I will look into different physical models, engine parameters, properties that can be put into the training and provide us with stable vehicle control. I don’t know how tightly coupled the DRL Agents are; I will attempt to train these agents without depending on the physical properties and reach the physical parameters with the highest precision possible, eliminating the physical aspect from the trained model, and gaining advantages from the previous controlling techniques.

Research Questions:

How far can we rely on Deep Reinforcement Learning (DRL) techniques for stable vehicle control?

What is the dependency of DRL Agents on the given physical model?

How can multiple Agents interact within the simulation environment?

In order to reach the goal of CDM definition, a software stack has to be provided for each milestone. The software environment for the CDM framework is built by layers upon layers.

The following environmental models and standards are needed to be integrated:

1. ASAM OpenDrive

(a) the road surface where the ITVs are moving

(b) XML model classes to building up the road surface

2. ASAM OpenSenario

(a) declarative language to describe the scenario

(b) provide the speed profile and the desired outcome

3. Vehicle control parameters and a simple bicycle model

4. CDM at the top

This layered software architecture integrates the different functionalities and test environments. I will utilize a Cooperative Traffic Simulator (CTS), which is a simple Java Desktop Application to support Agent trainings for the Cooperative Driver Model. This desktop application is being built up using the following libraries:

- **Cooperative Traffic Simulator**
  - Contains the graphical user interface
  - *Dependencies*
    - JavaFX
    - CTSAgentLibrary
    - CTSModelLibrary
- **CTSAgentLibrary**
  - Contains the training environment for the longitudinal and lateral control agents
  - *Dependencies*
    - Deep Learning for Java (DL4J)
- **CTSModelLibrary**
  - Contains the vehicle models and related constants
- **CTSOpenDriveLibrary**
  - Contains the road elements as provided by the OpenDrive Standard
  - *Dependencies*
    - JAXB

Keywords: Autonomous Vehicles, Intelligent Test Vehicles (ITVs), Cooperative Driving Model (CDM), OpenDrive, OpenScenario

Table of Contents

1. Introduction 7
   A. Motivation 7
   B. Autonomous Vehicle 8
      I. SAE Levels of Automation 8
   C.Testing concepts for autonomous vehicles 9
      I. Software-in-the-loop (SIL) Environments 9
      II. Hardware-in-the-loop (HIL) Environments 9
      III. Intelligent Test Vehicles (ITV) Concept 10
2. Existing Solutions 12
   A. Real-Time Train Traffic Management System 12
      I. MiLP and constraints 14
   B. Control Algorithms 15
      I. Bicycle Model 15
   C. Industry standards 18
      I. OpenDrive 18
      II. OpenScenario 19
3. Reinforcement Learning 20
   A. Markov Decision Processes 22
   B. Q-Value function 22
   C. Deep Q-Learning 23
4.Control Algorithms 24
   A. Longitudinal Control 24
      I. Conclusions of Longitudinal Control 31
   B. Lateral Control 32
      I. Conclusions of Lateral Control 42
   C. Physical Vehicle Models 43
5. Multi-agent systems 44
   A. Collaborative Scenarios 44
   B. Competitive Scenarios 44
6. Cooperative Driving Model 45

A. In-depth Description of the developed concept 45
B. Results 45
I. Trained Scenario 45
II. Trained Agents 46
C. Answers 47
I. How far can we rely on Deep Reinforcement Learning (DRL) techniques for stable vehicle control? 47
II. What is the dependency of DRL Agents on the given physical model? 48
III. How can multiple Agents interact within the simulation environment? 48
D. Lessons Learnt 48
E. Summary 50
7. References 51

1. Introduction

A. Motivation

In contrast to existing methods our primary goal here is to train smooth input signals for driver robot technology where the vehicle is under full control by a set of electromechanical actuators. Existing methods are focusing on the vehicle parameters and not on the characteristics of the pedal set and steering wheel. With using driver robot technology, the actuator set being selected might have an impact on the overall driving behavior due to additional noise and delay being introduced. Trained reinforcement learning models are able to provide noise tolerant, calibration free actuator set for pedal- and steering wheel control as being proven during my thesis.

Smooth, vehicle independent and reliable collaborative driving behavior might not be achieved using traditional control methods. To guarantee any coherence between coupled driving robots might lead to computationally overcomplicated equations. In order to find a solution to this challenge we have decided to use the latest reinforcement learning techniques rather than traditional control methods like Proportional Integral Derivative (PID) or Linear Quadratic Regulator (LQR).

Intelligent Test Vehicles (ITVs) are following a predefined trajectory based on the actual scenario. Just the slightest offset from the ideal scenario and trajectory plan might lead to the situation where the desired outcome is unreachable. To compensate for such delays and offsets, cooperative behavior is required between the test vehicles to adapt to new conditions and compensate. ITVs cannot make decisions that would lead to another scenario – their maneuver space is limited. But ITVs should be able to react on the Vehicle Under Test (VUT) decisions, and they might be able to compensate to avoid accidents.

Collaborative vehicle behavior is essential to automated vehicle testing but also for autonomous driving. In our understanding the future of fully automated driving depends on developing collaborative behavior between multiple agents instead of sorting out complex scenarios alone. Traffic rules today are based on common social acceptance of different driving behaviors. The interactions between human drivers cannot be neglected by automakers and taken out of the traffic system. This might lead to deadly accidents and traffic breakdowns while having both human- and automated-drivers on the roads.

In our Markov Decision Processes (MDPs) we have been selected actions and system parameters abstracting the I/O needs of a linear actuator in case of pedal control (brake and acceleration), what we have called the longitudinal control, and we have modelled the I/O needs of a stepper motor with a simple gearbox that represents the steering wheel turn as required by a lateral control.

B. Autonomous Vehicle

An autonomous vehicle, also known as a robot car, a self-driving car or a driverless car [1], can monitor and sense its environment and move with limited or no human input [2].

Vehicles are reportedly to become so intelligent, human intervention would not be required in the future. Nevertheless, until we reach the level of fully autonomous vehicles, there are phases along the way in technology.

I. SAE Levels of Automation

There are six levels of autonomy: [3]

0. No automation: The driver is ultimately responsible for controlling the vehicle, performing actions like steering, braking, slowing down, or accelerating.

1. Driver assistance: This is the lowest level of automation. The vehicles with features that stop them from leaving a highway lane unexpectedly (lane departure assist), or enable them to follow traffic at a safe distance (adaptive cruise control) belong to this category.

2. Partial automation: The automated system can take full control of the acceleration, braking, maintaining speed, and steering the vehicle together. Although, the driver has to be prepared to intervene immediately if the system fails to respond accurately.

3. Conditional automation: At this level, the vehicle can perform more complex functions that pair steering (lateral control) with acceleration and braking (longitudinal control), owing to a greater awareness of its surroundings.

4. High automation: The vehicles can steer, accelerate, and brake on their own. Moreover, they can monitor road conditions and respond to obstacles, determining when to change lanes and when to turn.

5. Full automation: For a level 5 vehicle, no human intervention is needed. It would be entirely robotized and would work on all surfaces, weather without any conditions being required to be met.

C.Testing concepts for autonomous vehicles

Advanced driver assistance systems (ADAS) are the opening move toward a fully automated future. ADAS are electronic systems (i,e camera-based sensors) intended to help the driver and the vehicle to have greater awareness of the driving environment. Software-in-the-loop (SIL) Environments, Hardware-in-the-loop (HIL) Environments and Intelligent Traffic Vehicle Concept (ITV) are some testing concepts for autonomous vehicles [4].

I. Software-in-the-loop (SIL) Environments

SIL environment involves linking the algorithms that correspond to a specific vehicle's hardware to the simulation. By utilizing software-in-the-loop, developers can examine code performance in a simulated environment without actual hardware parts.

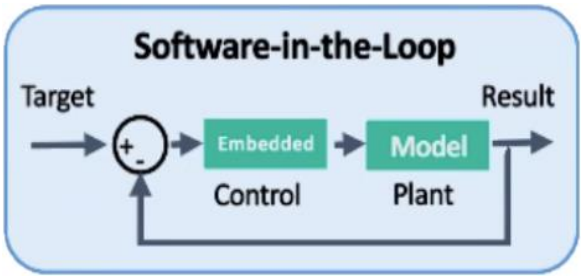


Figure 1. Software in the loop. Extracted from an internal presentation of Lindenmaier Autonomous Technologies.

II. Hardware-in-the-loop (HIL) Environments

HIL environment was previously a tool for developing a car's engine and vehicle dynamic controllers. At the moment, it's an approved method for ADAS and autonomous car testing. The hardware-in-the-loop approach means making use of a real-time simulation for inspecting a vehicle's hardware. The HIL method is flexible and significant for prototyping.

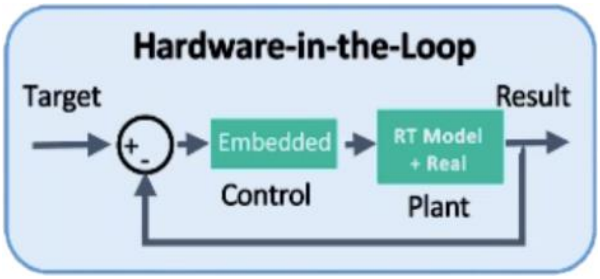


Figure 2. Hardware in the loop. Extracted from an internal presentation of Lindenmaier Autonomous Technologies.

III. Intelligent Test Vehicles (ITV) Concept

Intelligent Test Vehicles are used to generate real traffic situations as described by a scenario template (phrased by OpenScenario) [5]. Ideally on each ITVs we should have the same algorithm running. There are multiple options to evaluate a given scenario:

1. Static decentralized evaluation:

The control strictly follows the given trajectory with the given speed.
All vehicle control properties are known in advance.
No divergence allowed from the given goal.

2. Dynamic centralized evaluation:

There is a centralized supervision of the scenario outcome.
The vehicle control might get a new goal to adapt to the changes in the scenario.

3. Cooperative evaluation:

Each ITV has its role in the given scenario and has a clear vision about the goal.
In order to reach the scenario goal each ITV is adapting its control according to environmental changes.

	Advantages	**Disadvantages**
Static decentralized evaluation	Low complexity on ITV No control center needed Predictable properties Simple vehicle control - Given trajectory - Given speed profile	Environmental changes might make the scenario fail Scenario failure might lead to accident
Dynamic centralized evaluation	Low complexity on ITV Predictable properties Simple vehicle control - Given trajectory - Given speed profile Control center can react to avoid accident	Central supervision needed Reliable and high speed communication link to each ITVs Complex infrastructure needed

Cooperative evaluation	Scenario goal can be guaranteed - Adaptation to environment changes No central supervision needed - Lower infrastructure costs	Higher algorithmic complexity on ITVs - Increased computational power needed Unpredictable properties - ITV ISO26262 coverage More complex vehicle control - Trajectory calculation - Speed profile adaptation

Table 1. Advantages and disadvantages of static decentralized, dynamic centralized and cooperative evaluation.

Each of these models are having advantages and disadvantages as described by the table above.

The following figure shows the trade-offs between the different technological approaches. The primary goal is to keep the infrastructure complexity and cost low because system adaptation to every proving ground facility would introduce large expenses and the facility might not necessarily be a partner on that.

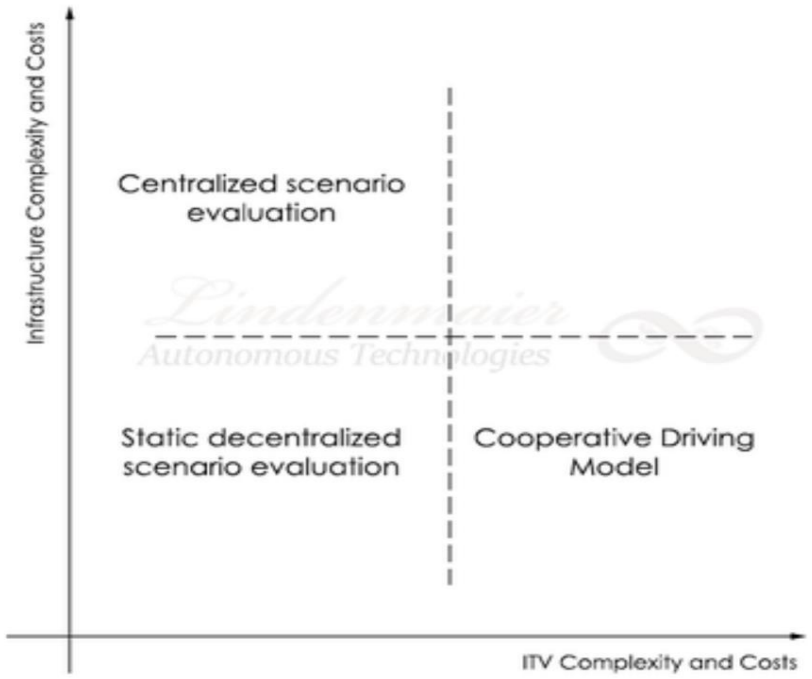


Figure 3. Trade-offs between the different technological approaches. Extracted from an internal presentation of Lindenmaier Autonomous Technologies.

2. Existing Solutions

A. Real-Time Train Traffic Management System

The real time railway traffic management seeks for the train routing and scheduling that minimize delays after an unexpected event unsettles the operations [6]. This problem has been resolved by introducing an algorithm based on a mixed integer linear programming model (MILP) [7], modeling the infrastructure in terms of track-circuits, which are the basic components for train detection. MILP is only a problem of formulation that takes into account all possible alternatives for train rerouting in the infrastructure and all rescheduling alternatives for trains along these routes. There are heuristic solutions but neither the solution of heuristic is always heuristic.

In the MILP-based railway traffic management solution [6], routes are divided into unique nominal track-circuits. Based on a proper graph representation of the control area this infrastructure model provides a fine realistic representation. Furthermore, it selects among all the possible routes that can be practically exploited and it considers train reorderings.

The real time railway traffic management problem is formed by selecting proper train routes and scheduling that decrease the propagation of delay in case of traffic perturbation.
This approach could be applied for the Cooperative Driving Model; we wish to create a test scenario where the vehicle under test avoids collision with intelligent test vehicles (ITV) , similarly minimizing the delays of the ITVs.

We investigate

- Scenario 1:

The vehicle 1 enters the bottom track, with initial speed 50 and entry time 1s,
the vehicle 2 enters the top track, with initial speed 100 and entry time 2s.
The speed of vehicle 2 significantly reduces in order to allow vehicle 1 to surpass it and avoid collision.
We can observe the scenario 1 below.

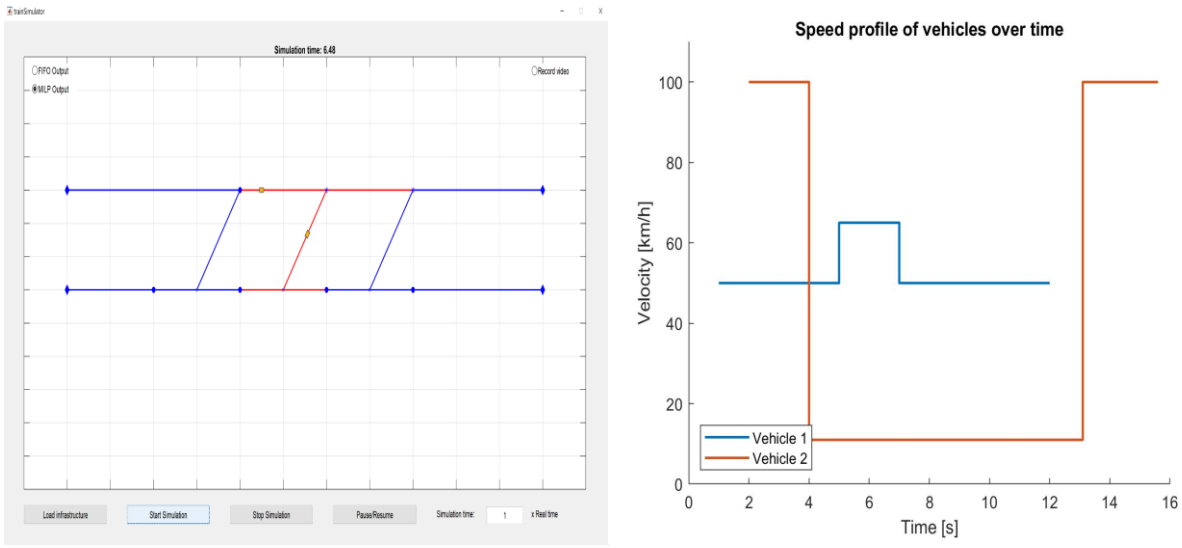


Figure 4. Execution of Scenario 1 and Speed profile of vehicles over time. Extracted from Matlab simulation.

- Scenario 2:

The vehicle 1 enters on the bottom track, with initial speed 50 and entry time 1s, the vehicle 2 enters on the top track with initial speed 30 and entry time 2s.
The vehicle 1 is faster than the vehicle 2 on the top track so the change of tracks is completed as fast as possible.

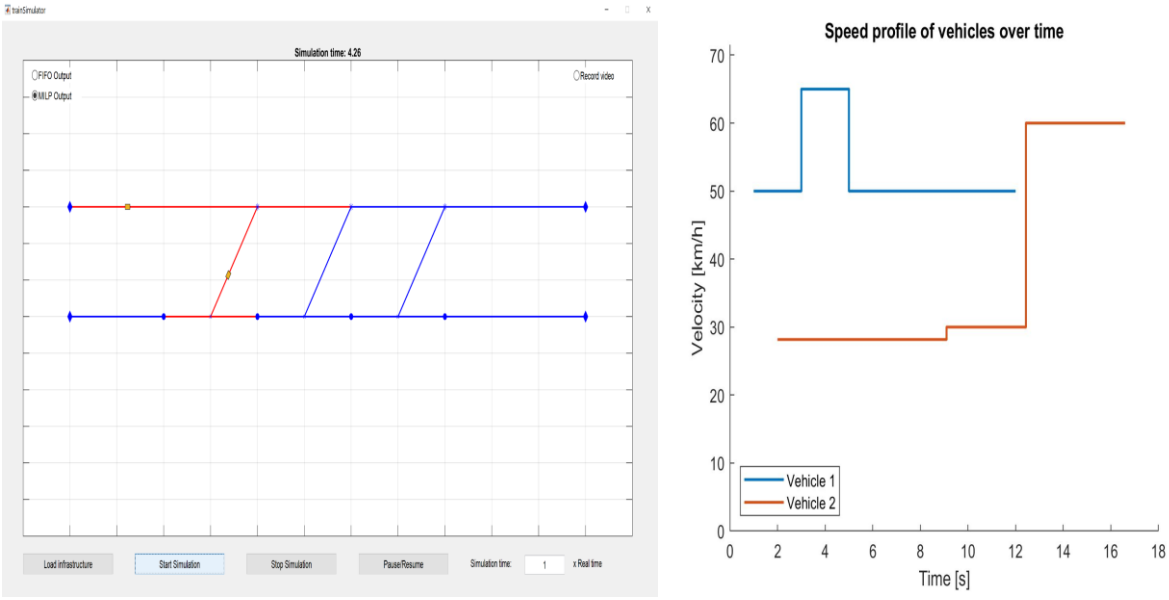


Figure 5. Execution of Scenario 2 and Speed profile of vehicles over time. Extracted from Matlab simulation.

In consequence, it can be mathematically proven that MILP always finds the global optimum of a problem in terms of objective function. We receive satisfactory performance since the collision between the two cars is avoided. Nonetheless, a few difficulties still emerge, regarding the discrete decisions (macro description of the scenario) and the centralized approach. Moreover, an extremely large number of routes can be introduced, a fact that could greatly increase exponentially the runtime complexity of our problem over the number of states. In railway traffic real-time this could mean 1 to 2 minutes but in vehicle tests it would be fatal. Another major issue that occurs is the scalability of the problem; we cannot add, for instance, two hundred cars, because it will computationally explode pretty fast. Thus we will look for alternative ways to solve our cooperative traffic behaviour problem, namely Deep Reinforcement Learning.

I. MiLP and constraints

A set of constraints are imposed for our vehicle scenarios that characterize the problem based on the existing ones from [6]. We will adapt the constraints of the real time railway traffic management problem and based on the changes that are needed, we will keep, remove, update the constraints.

First of all, time concerning constraints, a car cannot be scheduled earlier than its entry time. Before a car enters a block section, some time must be allowed for route formation and for taking into account the signal visibility distance, this time is called formation time. The time in which the car occupies two consecutive road-circuits is named clearing time: its rear is still on the current road-circuit and its front has already entered the following one. We can also introduce constraints for managing delays. We have the following options; no constraints, delay allowed at any signal and delay allowed only out of the control area. The signal controls the access to the sequence of road-circuits. Finally as far as the car model is concerned, cars can travel with constant velocity, during the signal positions they can accelerate /decelerate and have a run time. Constraints related to rolling stock configuration that are made for the train associated problems are ignored. In our vehicle test scenario joint and splitting trains are not interpreted. If two trains are in connection with each other, some new constraints would be introduced to manage their scheduling, but this is totally train related.

B. Control Algorithms

Research on autonomous vehicles has mostly focused on three main layers: perception, planning and control. With the help of control algorithms, we emphasize on the link between trajectory planning and low-level control.

Control system plays a very decisive role in the entire architecture of an autonomous vehicle and being the final member of the pipeline, it is in charge of actually "driving" the vehicle. If we are to look from the perspective of autonomous vehicles, the control system is dedicated to generate appropriate commands for brake and steering so that the vehicle tracks a prescribed trajectory by executing controlled motion. It is this sub-system, which eventually decides how the autonomous vehicle will behave and interact with the environment.

I. Bicycle Model

We will introduce the bicycle kinematic model to achieve lateral control for our agent. A 2D bicycle model can be expressed as a simplified car model [8]. It is a classic model that performs effectively at capturing vehicle motion in standard driving conditions.
The kinematic bicycle model is a simplification of the kinematic four wheel model, where the two rear wheels and the two front wheels are lumped into one imaginary rear wheel and front wheel respectively.
To analyze the kinematics of the bicycle model, three points may be used a) the centre of the front axle, b) the centre of the rear axle and c) the centre of gravity of the autonomous vehicle.

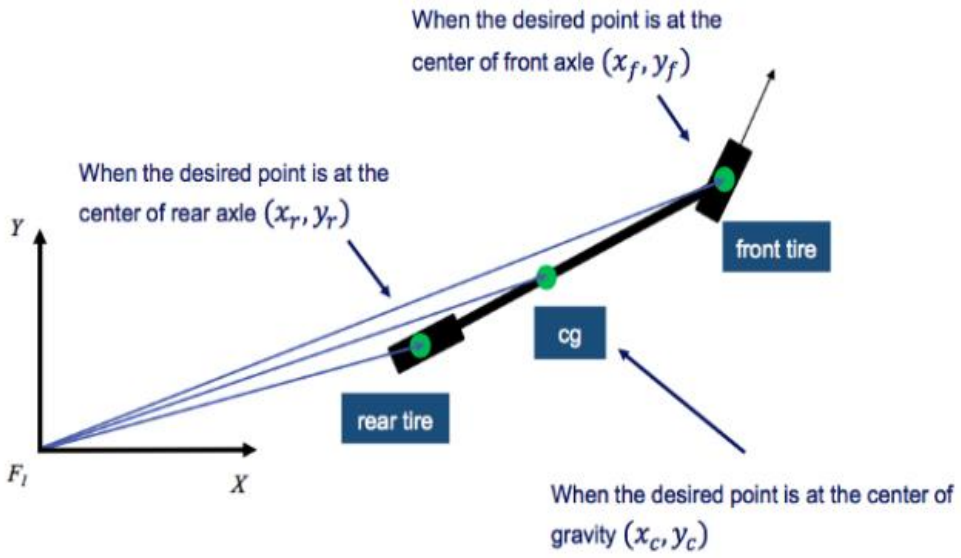


Figure 6. Three reference points of the kinematic model. Extracted from [8].

The bicycle model we'll develop is called the front wheel steering model, since the front wheel orientation can be controlled relative to the heading of the vehicle. Our goal is to compute state **[x, y, θ, δ]**, θ is heading angle, δ is steering angle. Our inputs are **[v, φ]**, v is velocity, φ is steering rate.

If the desired point is at the center of the front axle:

$$\dot{x}_f = v\cos(\theta + \delta)$$
$$\dot{y}_f = v\sin(\theta + \delta)$$
$$\dot{\theta} = \frac{v\sin\delta}{L}$$

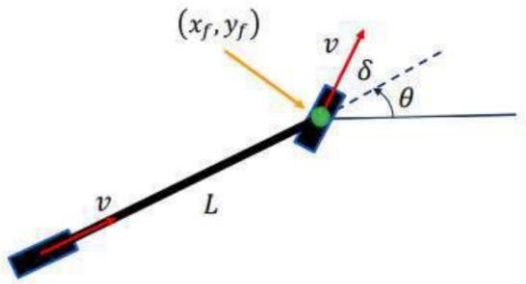


Figure 7. Model analysis of desired point at front axle. Extracted from [8].

If the desired point is at the center of the rear axle:

$$\dot{x}_r = v\cos\theta$$
$$\dot{y}_r = v\sin\theta$$
$$\dot{\theta} = \frac{v\tan\delta}{L}$$

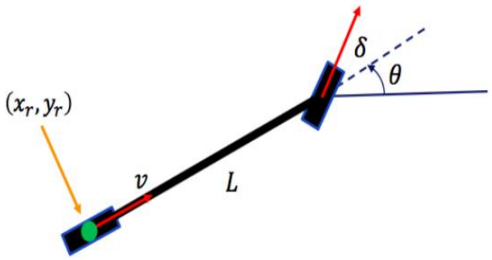


Figure 8. Rear Axle Bicycle Model. Extracted from [8].

We will consider the desired point as the center of gravity by averaging the center of the rear and front axle.

$$\dot{x}_c = v\cos(\theta + \beta)$$
$$\dot{y}_c = v\sin(\theta + \beta)$$
$$\dot{\theta} = \frac{v\cos\beta\tan\delta}{L}$$

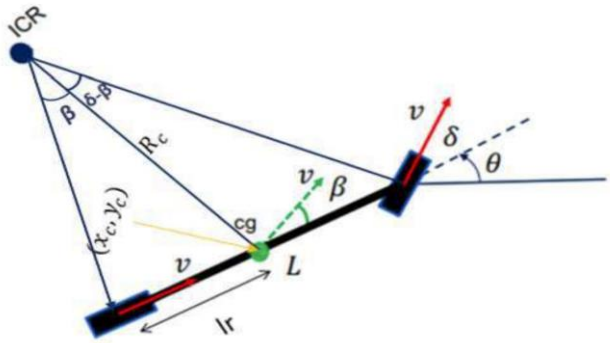


Figure 9. Model analysis of desired point at the center of gravity. Extracted from [8].

To sum up, our model is the bicycle kinematic model as has been analyzed. We will proceed by picking the desired point at the center of gravity for our implementation. States (outputs) are [x, y, θ, δ]. Inputs are [v, φ], v is velocity, φ is steering rate. To get the final state [x, y, θ, δ], we can use discrete time model. State transition equations are :

$$x_{(t+1)} = x_t + \dot{x} * \Delta t$$
$$y_{(t+1)} = y_t + \dot{y} * \Delta t$$
$$\theta_{(t+1)} = \theta_t + \dot{\theta} * \Delta t$$
$$\delta_{(t+1)} = \delta_t + \dot{\delta} * \Delta t$$

C. Industry standards

I. OpenDrive

ASAM OpenDRIVE [9] provides the exchange format specification to describe static road networks for driving simulation applications. The main goal of ASAM OpenDRIVE is the road description containing objects along the road. The OpenDRIVE Specification includes the description on how to model e.g. lanes, roads, junctions. Dynamic content is not covered by ASAM OpenDRIVE.

The format of OpenDRIVE provides a common base for describing road networks with Extensible Markup Language (XML) syntax, with the file extension xodr.
The OpenDRIVE file structure conforms to XML rules; the associated schema file is referenced in the XML. The schema file for the OpenDRIVE® format can be retrieved from https://www.asam.net/standards/detail/opendrive/. The shown XML snippet gives a short overview on how the linking of road segments can be used in ASAM OpenDRIVE.

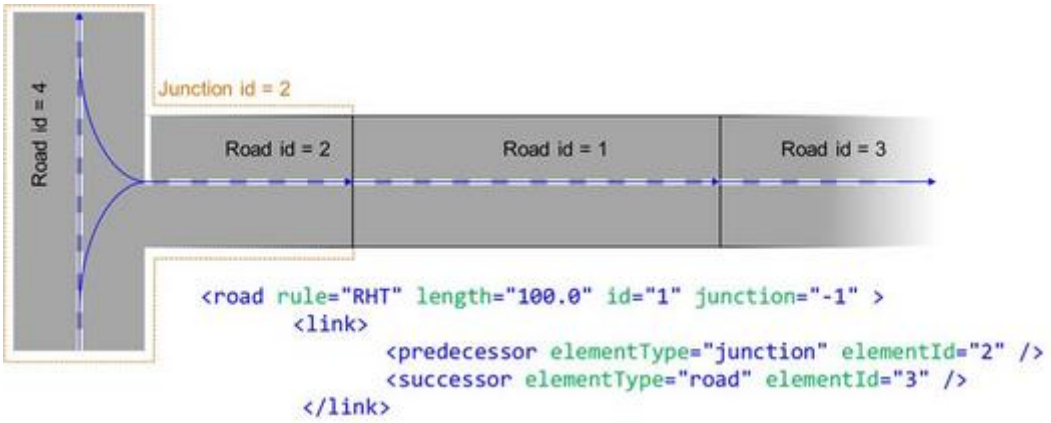


Figure 10. Individual road segments are interconnected. Extracted from [9]

The data that is saved in an OpenDRIVE file describes the geometry of roads in addition to features along the roads that influence the logics, for instance, lanes and signals. The road networks that are included in the OpenDRIVE file can either be synthetic or real. The primary goal of OpenDRIVE is to provide a road network description that can be fed into simulations and to build these road network descriptions exchangeable.

The format is organized in nodes that can be extended with user defined data. This simplifies a high degree of specialization for individual applications (mostly simulations) while maintaining the interoperability that is needed for the exchange of data between different applications.

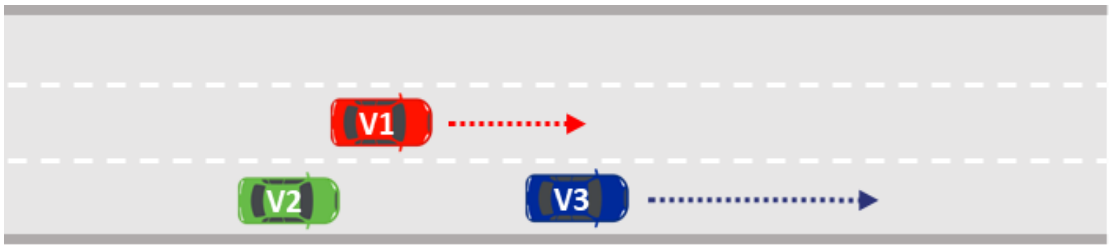
II. OpenScenario

With the advancement of autonomous vehicles, the demand for more sophisticated testing and validation of ADAS and autonomous driving systems is rising exponentially. It is expected that simulation-based testing will replace the unattainable goal of test driving billions of kilometers for validation purposes. Different companies have hitherto published their use of thousands of simulation-based tests to validate their autonomous systems.

OpenSCENARIO 1.0 [5] is a low-level and concrete specification format, mainly designed to be recognized by simulation tools. Users, who create maneuver descriptions and tests, need a higher level of abstraction and potentially different ways to express this than in the current detailed XML format of OpenSCENARIO 1.0.

There are several proposals on how to express maneuver descriptions on a higher level of abstraction, with interfaces to general-purpose programming languages, textual stylized natural-language scenario descriptions, and a domain-specific language (DSL) being among the expressed proposals.

The OpenSCENARIO 2.0 concepts inherit proven features and capabilities of OpenSCENARIO 1.0, like its event-based scenario execution model, and set them in a more familiar and expressive language framework to present as the foundation for both incremental improvements and more innovative enhancements. We can have a look at an example of OpenSCENARIO 2.0 :




```
keep(v1.color == green)                  # A single value
keep(v1.color in [green, blue])          # A choice of two colors
keep(v1.color != v2.color)               # They can't have the same color
keep(v1.color == red or v2.color == red) # One of them must be red
keep(t in [7..12.5]s)                    # By default will use a flat distribution
keep(soft t == normal(10s, 2s)           # Will use a normal distribution with
                                         # a mean of 10 seconds and std deviation of 2
```

Figure 11. OpenSCENARIO 2.0 example. Extracted from [5]

In contrast to OpenSCENARIO 1.0, a more detailed set of actions and attributes for the relevant simulation models shall be defined to enable a more thorough scenario description and improve exchangeability.

3. Reinforcement Learning

Reinforcement learning is learning what to do, how to map situations to actions to maximize a numerical reward signal [10]. The learner is not told which actions to take but instead must discover which actions yield the most reward by trying them.

The basic idea is simply to capture the most critical aspects of the real problem facing a learning agent interacting over time with its environment to achieve a goal. A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment.

All reinforcement learning agents have precise goals, can sense aspects of their environments, and can choose actions to influence their environments. By a complete, interactive, goal-seeking agent, we do not always mean something like a whole organism or robot. These are clearly examples, but a complete, interactive, goal-seeking agent can also be a component of a more extensive behaving system.

Sensation, Action, and Goal

Reinforcement learning is attempting to maximize a reward signal instead of finding hidden structure in the training data. Hence, we regard reinforcement learning to be a third machine learning paradigm, besides supervised learning and unsupervised learning and possibly further paradigms.

The agent must exploit what it has previously experienced to gain a reward, besides that it has to explore to make better action selections in the future. The quandary is that neither exploration nor exploitation can be pursued unaccompanied without experiencing failure at the objective. The agent must attempt a plethora of actions and gradually approve those that appear to be best.

Exploit vs. Explore

An additional essential feature of reinforcement learning is that it distinctly reviews the entire problem of a goal-directed agent interacting with an unknown environment. This conflicts with several approaches that examine subproblems without addressing how they might fit into a wider picture.

Beyond the agent and the environment, we can point out four basic sub-elements of a reinforcement learning system: a policy, a reward signal, a value function, and, by preference, a model of the environment.

Policy, Reward signal, Value function, Environment model

- A **policy** determines the learning agent's way of behaving at a given time. In other words, a policy is a method of mapping from perceived states of the environment to actions to be taken when we arrive in those states.
- A **reward signal** defines the purpose of a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the reward. The agent's exclusive objective is to maximize the overall reward it receives over the long run. The agent continually learns to maximize its reward. Hence, It is vital that the rewards we set up actually indicate what we wish to accomplish. The reward signal is our way of communicating to the robot what we desire to succeed, not how we want it achieved.
- While on the contrary the reward signal shows what is good in an instant sense, a **value function** points out what is good in the future. The value of a state depicts the total amount of reward an agent can expect to accumulate over the long run, starting from that state. In particular, a state may consistently yield a low immediate reward but still have a high value as it is regularly followed by other conditions that yield high rewards. Or the reverse could be true. In a sense, rewards are primary, whereas values, as predictions of rewards, are secondary. Without rewards, there could be no values, and the only goal of estimating values is to achieve more rewards. Nonetheless, it is the values with which we are most interested when making and evaluating decisions. Action choices are made based on value judgments. We look for actions that bring about states of the highest value, not the highest reward, because these actions obtain the most notable amount of reward for us over the long run. The primary role of value estimation is arguably the most crucial thing that has been discovered about reinforcement learning over the last decades.
- **Model of the environment** imitates the behavior of the environment, allowing inferences to be made about how the environment will behave. For instance, given a state and action, the model might foresee the resultant next state and next reward. Methods for solving reinforcement learning problems that implement models and planning are called model-based methods, in opposition to more straightforward model-free methods that are explicitly trial-and-error learners, considered as nearly the opposite of planning.

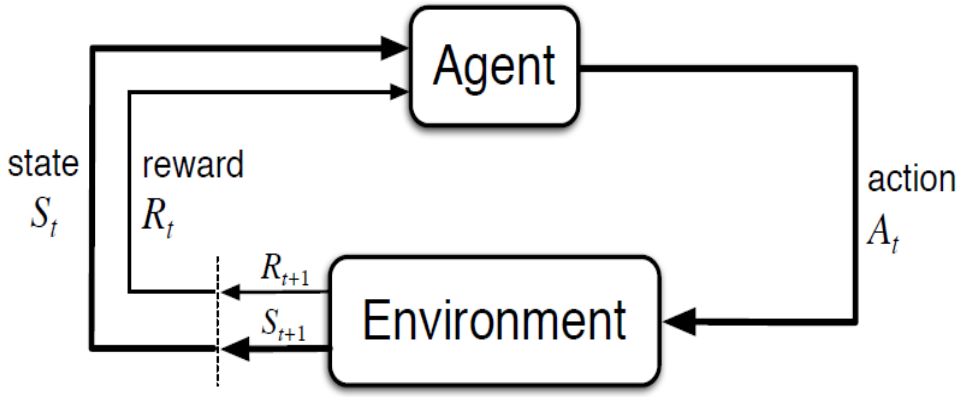


Figure 12. The agent–environment interaction in a Markov decision process. Extracted from [10].

A. Markov Decision Processes

A Markov decision process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker.

A Markov decision process is a 4-tuple (S, A, P_a, R_a), where:

- S is a set of states named the state space,
- A is a set of actions called the action space,
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$, where the probability that action a in state s at time t will lead to state s' at time t+1,
- $Ra(s, s')$ is the immediate reward received after transitioning from state s to state s' due to action a.

The state and action spaces could be finite or infinite, for instance, the set of real numbers. A few processes with countless state and action spaces can be reduced to ones with limited state and action spaces.

A policy function π is a mapping from state space to action space.

B. Q-Value function

A Q-value function (Q) indicates how good a specific action is, given a state, for an agent following a policy. The optimal Q-value function (Q*) provides us maximum return achievable from a given state-action pair by any policy. In other words, The optimal Q-function Q*(s, a) means the highest possible Q value for an agent beginning from state s and choosing action a. There, Q*(s, a) is an indication of how good it is for an agent to select action while being in state s.

C. Deep Q-Learning

Q-learning is a model-free reinforcement learning algorithm seeking the best action to take given the current state. Q-learning is a values-based learning algorithm. Value-based algorithms update the value function based on an equation(particularly the Bellman equation) [11].

When Q-Learning is used combined with Deep Learning (artificial neural nets serve as function approximators), we call this Deep Q-Learning. Though nascent, the method has already helped businesses across industries to achieve significant breakthroughs in a variety of tasks. While on the contrary, policy-based estimates the value function with a greedy policy obtained from the last policy improvement.

4.Control Algorithms

A. Longitudinal Control

Robust and stable control is a necessity for the navigation of autonomous vehicles. The longitudinal controller is in charge of regulating the vehicle's cruise velocity.

We will investigate different training strategies for the longitudinal control agent while taking advantage of Deep Q-Learning and selecting the optimal model after training our agent.

In detail, for our Deep Q- Learning network, we are using the Adam optimizer with the value of learning rate 0.001, the number of hidden nodes is 150, and the number of layers is 2. The input and output layers are organized according to the actions that we have decided and the state variables that we have. The state variables are seven (the velocity, the previous acceleration, the current speed limit, the next speed limit, the distance to the next speed limit, the following (the one after the next) speed limit, the distance to the following speed limit.). The actions are two, three, four, or five (based on what we select). The input nodes are the state variables, and the output nodes are the number of actions we wish to train our agent.

The track length will be fixed to 1000 m, and we will set the speed limits to
50 km/h (0-250 m)
80 km/h (250-500 m)
40 km/h (500-750 m)
50 km/h (750-1000 m)

In the beginning, our primary concern was to provide a Reward a) based on the difference between the speed limit and the current speed and b) based on previous and current position (so we discourage our agent from moving backward). We realized that the agent could learn the goal and move forward only; thus, our reward function was merely focusing on minimizing between the speed limit and the current speed. In other words, if the reward is a small number, then our model is better as it closely follows the speed limit. Nevertheless, we should keep in mind that the final reward, we return, and define the trained models, is an accumulation of the rewards.

Strategies:

- 2 actions, Acceleration = 0.5, Deceleration= - 0.5
  The best reward we receive after training on these two actions is -3546.32. This strategy doesn't allow us to obtain a stable speed and has only two options for acceleration.

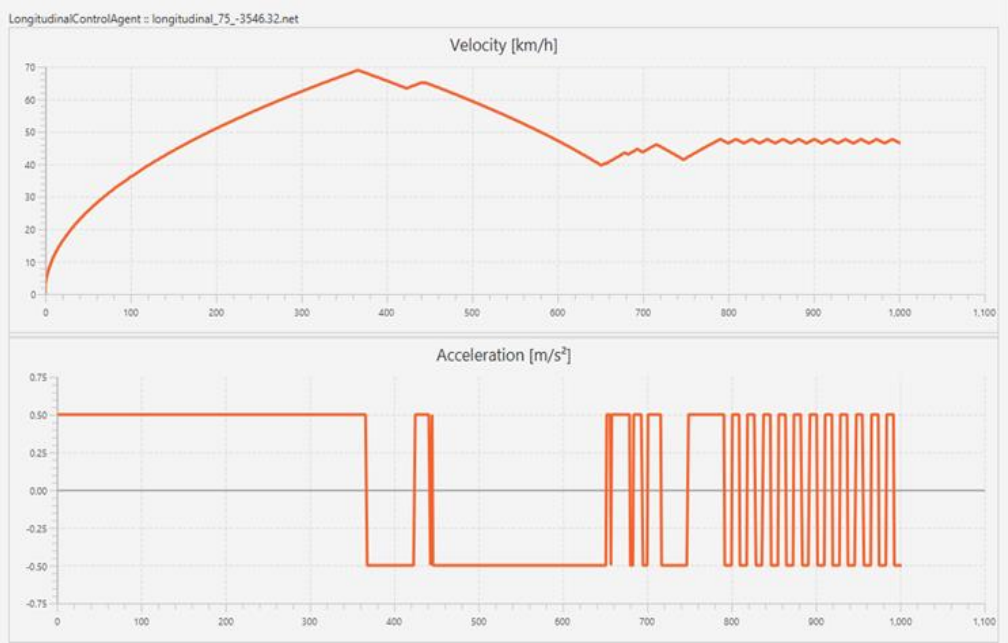


Figure 13. Velocity and Acceleration performance of 2 actions. Extracted from Cooperative Traffic Simulator.

- 2 actions, Acceleration based on factory defaults, Deceleration dependent on speed.

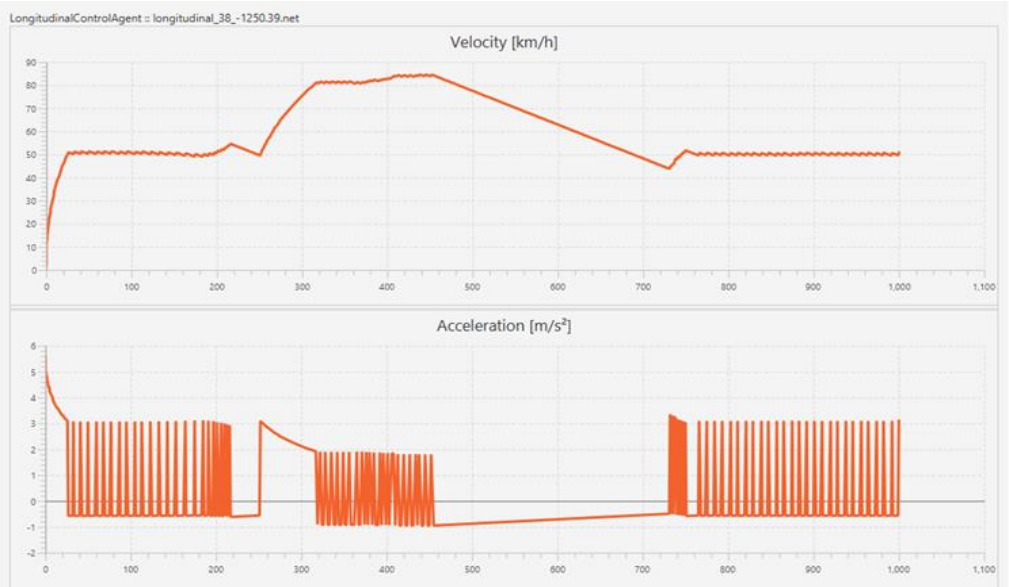


Figure 14. Velocity and Acceleration performance of 2 actions. Extracted from Cooperative Traffic Simulator.

This strategy cannot lead to smooth control, but we have a wider variety of accelerations. The best reward we receive is -1250.39, much smaller than the previous, indicating we are improving our strategy.

- 3 actions, Acceleration based on factory defaults, Deceleration dependent on speed, Zero Acceleration
  This strategy allows us to introduce stable speed with zero acceleration.
  Reward = - 1220.70

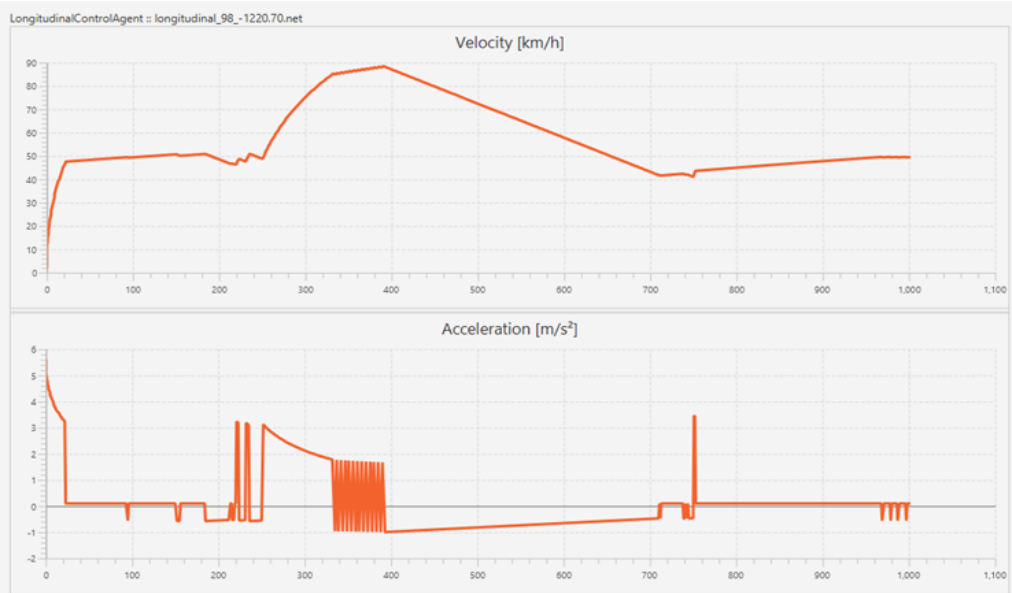


Figure 15. Velocity and Acceleration performance of 3 actions. Extracted from Cooperative Traffic Simulator.

- 4 actions, Acceleration with higher, lower power, Deceleration with higher, lower power
  Action 0 decelerate with higher power/ power++
  Action 1 decelerate with lower power/ power --
  Action 2 accelerate with lower power/ power--
  Action 3 accelerate with higher power/ power ++
  For both acceleration and deceleration, the power is limited in between 0.0 and 1.0, precisely {0.0, 0.05, …, 0.95, 1.0} and it is multiplied with the acceleration. This approach is closer to reality since we can consider the power as how much force we use to push the pedal to accelerate.
  * Power: 0 means not touching the pedal
  * Power: 1 means factory maximum
  Reward = - 899.91.We achieve smoother speed changes.

Figure 16. Velocity and Acceleration performance of 4 actions. Extracted from Cooperative Traffic Simulator.

- 5 actions, Acceleration with higher, lower power, Deceleration with higher, lower power, Zero Acceleration

  Action 0 decelerate with higher power/ power++
  Action 1 decelerate with lower power/ power --
  Action 2 accelerate with lower power/ power--
  Action 3 accelerate with higher power/ power ++
  Action 4 keep acceleration on 0
  We allow instant zero acceleration. Reward = - 871.65. This approach seems an enhanced version of the previous strategy on the grounds that we enable our agent to instantly switch the acceleration to 0, without waiting only for the power to reach zero. In other words, we allow the driver to remove his foot from the pedal immediately and not only slowly.

Figure 17. Velocity and Acceleration performance of 5 actions. Extracted from Cooperative Traffic Simulator.

We will improve our reward function by punishing the existing jerks; thus, fewer jerks mean smaller rewards and better models.Therefore, using the reward mentioned above function and the 5 actions strategy, we achieve the following result, which we consider the optimal model.

Figure 18. Velocity and Acceleration performance of the optimal model. Extracted from Cooperative Traffic Simulator.

The next step is to test the model by adding noise on a) velocity and b) acceleration.

a) Random noise on velocity, for every dt = 0.1 s, we add a random number between [-1.0 , 1.0] kilometers per hour to the velocity.

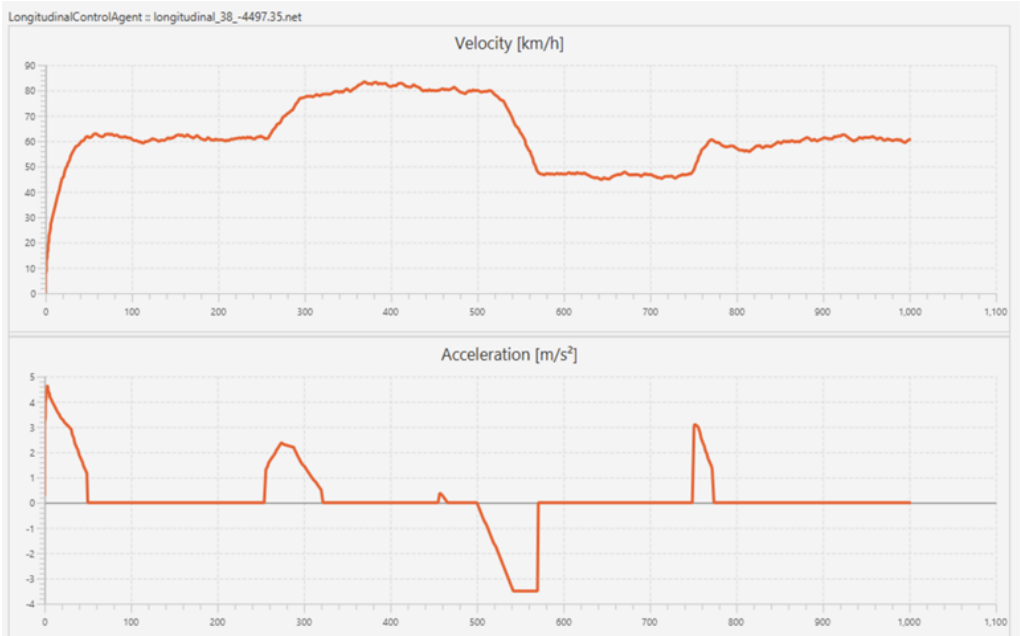


Figure 19. Velocity and Acceleration performance on noise tolerance. Extracted from Cooperative Traffic Simulator.

Compared to the optimal model, we notice that it follows the speed limits with slight disturbances. We expect these disturbances as in a real-life scenario; noise is unavoidable.

Test the model by adding:

b) Random noise on acceleration, for every dt = 0.1 s, we add a random number between [-0.25 , 0.25] meter per second squared to the acceleration. We observe in the following graph that the best-trained agent for this scenario is unable to follow the speed limits with a significant pump in velocity successfully. Consequently, the noise tolerance provides more desired results than the acceleration tolerance.

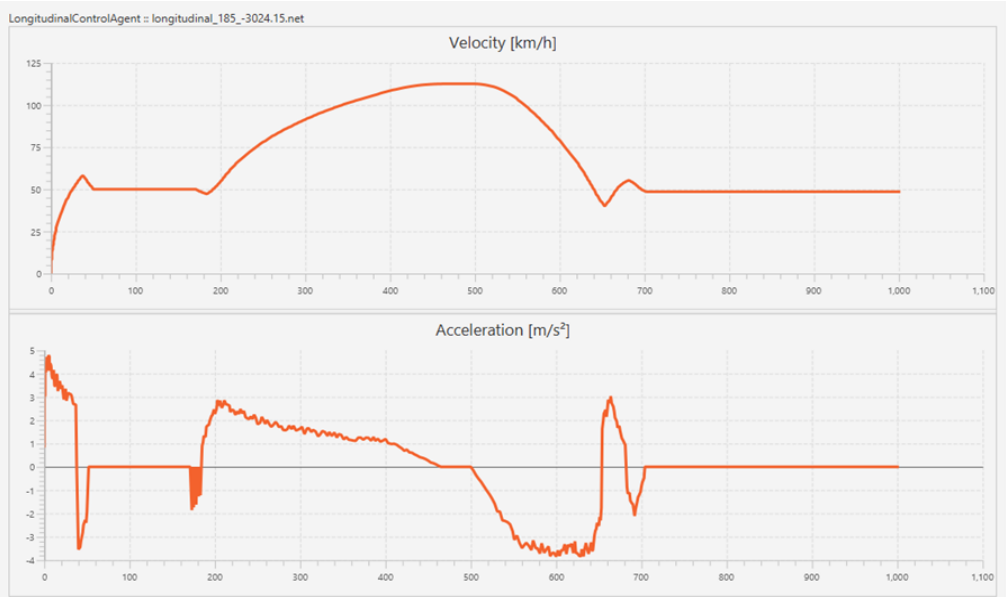


Figure 20. Velocity and Acceleration performance on acceleration tolerance. Extracted from Cooperative Traffic Simulator.

Finally, we train our reward function on different track lengths (800m) and different speed limits. From the results below, we can ensure independence from the track length and the speed limits:

40 km/h (0-250 m)
70 km/h (300-500 m)
50 km/h (500-800 m)

Figure 21. Velocity and Acceleration performance on different track length and speed limits. Extracted from Cooperative Traffic Simulator.

I. Conclusions of Longitudinal Control

We have trained a simple DQL model with a shallow neural network architecture that provided a road infrastructure-independent abstraction of the longitudinal control problem.

The main advantages of our solution are:

- Any OpenDRIVE speed profile can be given for training
    - This suits the scenario descriptors of a given test track
- The low number of parameters
    - We have created a highly configurable and well-performing control with only 7 environment parameters and 4 actions.
- Thanks to the shallow architecture, the trained network model is easy to deploy on any platform.
    - Fast training time (~10 min.)
    - The low computational power needed for inference
    - The solution fits microcontrollers with NN inference support – as driver robots might require using smart actuator concept
- Easy to change the vehicle model
    - The software library was written using definite interfaces that ease the process of adding new vehicle models.
    - Acceleration capabilities were based on factory data that is easy to update to any vehicle type.
- It fits different pedal characteristics.
    - Different vehicles might have other pedal characteristics that have an impact on the acceleration profile.
    - Our solution is noise-tolerant, which means we can deal with a wide range of pedal settings (for both braking and acceleration)
- With our solution, it is easy to tune the control to a target speed profile.
    - The reward function provides complete control over the system
    - We have been focusing on the smoothest possible acceleration/deceleration, so we tuned the reward system to penalize wild changes
    - One might want to reach the allowed top speed with the highest possible acceleration and provide a slow deceleration – it is just a matter of review the reward function
- We have presented a noise-tolerant solution
    - It is easy to add noise to any sensory data and measure its impact
    - We have added higher noise levels as expected in an actual application, but the speed profile still shows smooth control

B. Lateral Control

The purpose of the lateral control is to adjust the steering angle such that the autonomous car follows the reference path. The lateral controller steers the vehicle's wheel for path tracking and minimizes the distance between the current vehicle position and the reference path. We will investigate different training strategies for the lateral control agent while taking advantage of Deep Q-Learning and selecting the optimal model after training our agent.

In detail, for our Deep Q- Learning network, we are using the same network we used for the longitudinal control, the Adam optimizer with the value of learning rate 0.001, the number of hidden nodes is 150, and the number of layers is 2. The input and output layers are organized according to the actions that we have decided and the state variables that we have. The state variables are eleven now : the lane center offset, the car heading angle, the target heading angle, the steering angle, the car point in x coordinate, the car point in y coordinate, the target distance, the next target point in x coordinate, the next car target point in y coordinate, the following (the one after next) car target point in x coordinate, the following car target point in y coordinate. The actions are two, three, four, or five (based on what we select). The input nodes are the state variables, and the output nodes are the number of actions we wish to train our agent.

In addition to this, the reward function will return a number that will decrease when the difference between the current and the desired heading angle of the car is slight and will increase when the car distance rises from the gaussian distribution of the lane center offset of the vehicle. Alternatively, for each step the closer the center's offset of the car is, the higher the reward is. Similarly to the reward of the longitudinal control, the final reward of the trained model is a sum of rewards.

We will use the test tracks below a) a straight line tack, b) a circle-like track, and c) a track with sharp turns. The tracks have been designed based on the OpenDrive requirements, with the help of OpenRoadEd [12]. We wish to teach our agent to complete the tracks on a straight line, on soft and sharp turns, by maintaining the desirable lane offset without exiting the lane width of the road (3.75m).

We will set the vehicle to a constant speed of 5km/h. Similarly to the longitudinal control agent, we will attempt different strategies. The length for the first track, which is a straight line, is 2500 m; for the second circle track, the length is 552 m, and for the last track with the sharp turn, it is 1252 m. We will train our agents mostly on the last track as it can be considered as a combination of the other two, as it has both straight line and turns.

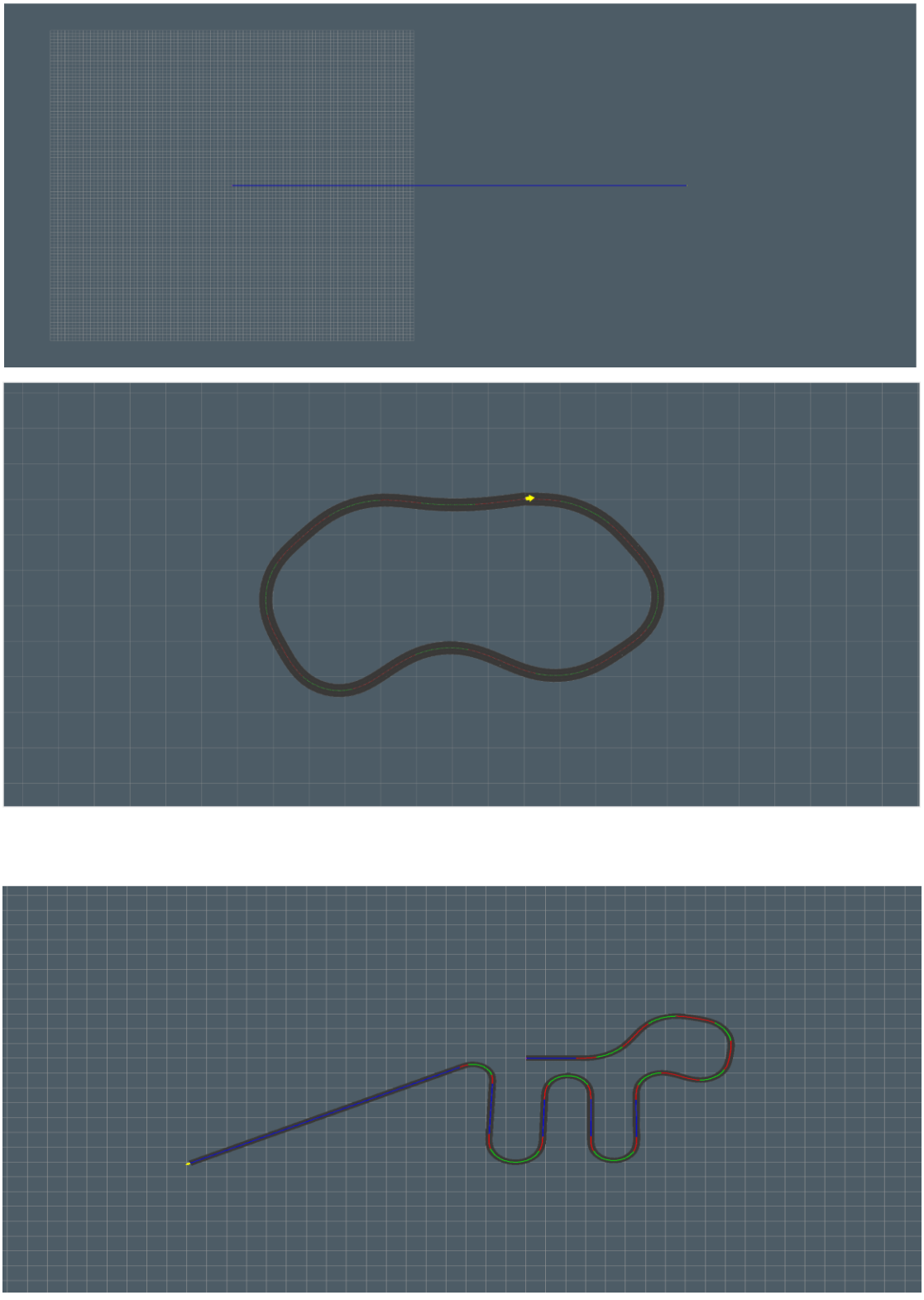


Figure 22. A snippet of test tracks: straight-line track, circle track, track with sharp turns. Extracted from OpenRoadEd.

Strategies:

- 2 actions, where the steering angle is 10 or -10 degrees, translates to turning the steering wheel right or left. This approach doesn't allow us to obtain a stable steering angle as the steering wheel constantly changes to stay in the track.

  Generally, we are more interested in achieving a smooth car heading angle than a smooth steering angle. While steering angle describes the orientation of the steerable wheels and hence the direction of motion of the vehicle under test, heading is concerned with the orientation of the vehicle under test.

  After training our agent, we notice that different models perform better in other tracks; for example, for the first track, we obtain the model from epoch 955 with a reward 195.48 that can finish the whole track.

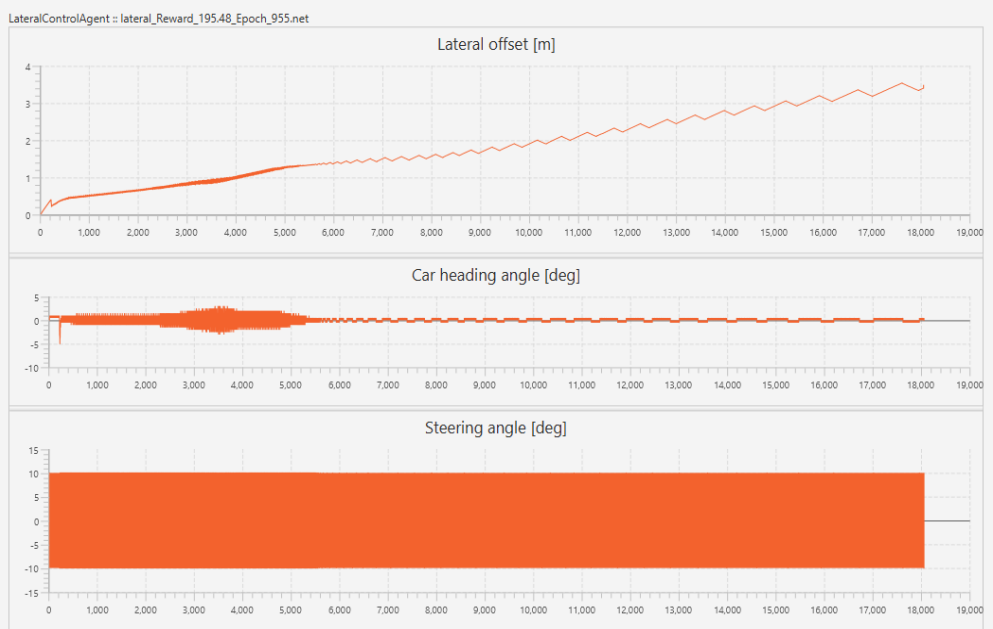


Figure 23. Lateral offset, Car heading angle, Steering angle of 2 actions for the first track. Extracted from Cooperative Traffic Simulator.

We acquire models that can complete both (second and third) tracks. One of them is from 561 epoch with reward -10.36.

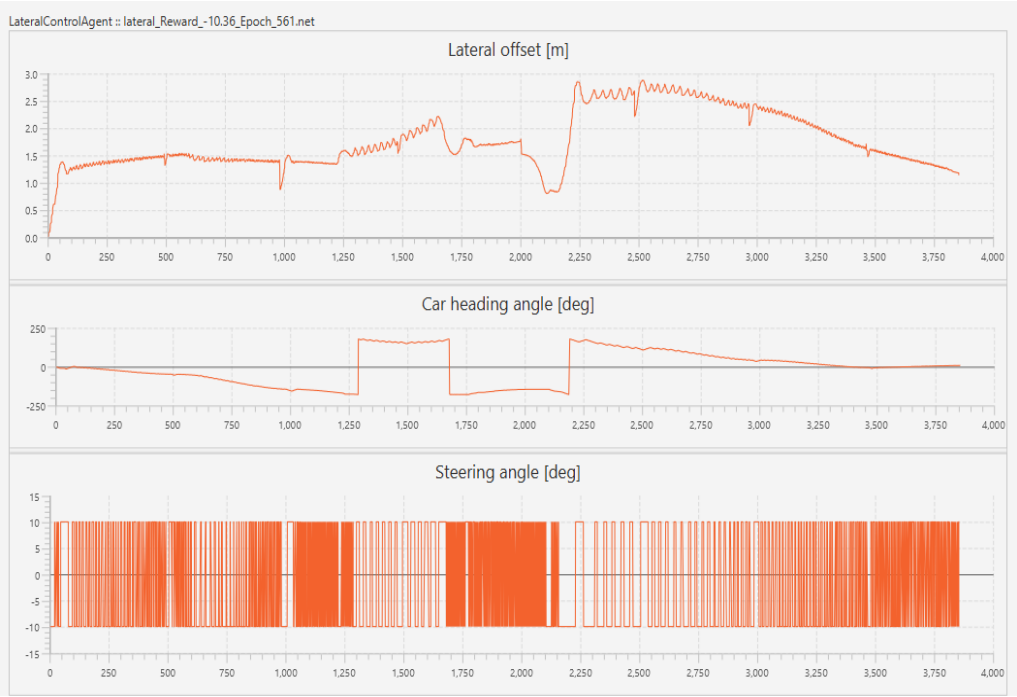


Figure 24. Lateral offset, Car heading angle, Steering angle of 2 actions for the second track. Extracted from Cooperative Traffic Simulator.

Figure 25. Lateral offset, Car heading angle, Steering angle of 2 actions for the third track. Extracted from Cooperative Traffic Simulator.

- 2 actions, where the steering angle is increased by 0.5 degree or decreased by -0.5 degree. This approach doesn't allow us to obtain a stable steering angle, but

in contrast to the previous strategy, we have wider variety in steering angle. The model from epoch 208 was able to complete the first track with reward 176.76.

Figure 26. Lateral offset, Car heading angle, Steering angle of 2 actions for the first track. Extracted from Cooperative Traffic Simulator.

For the second track, we have a model that completes the whole track from epoch 140 with reward 3.35.

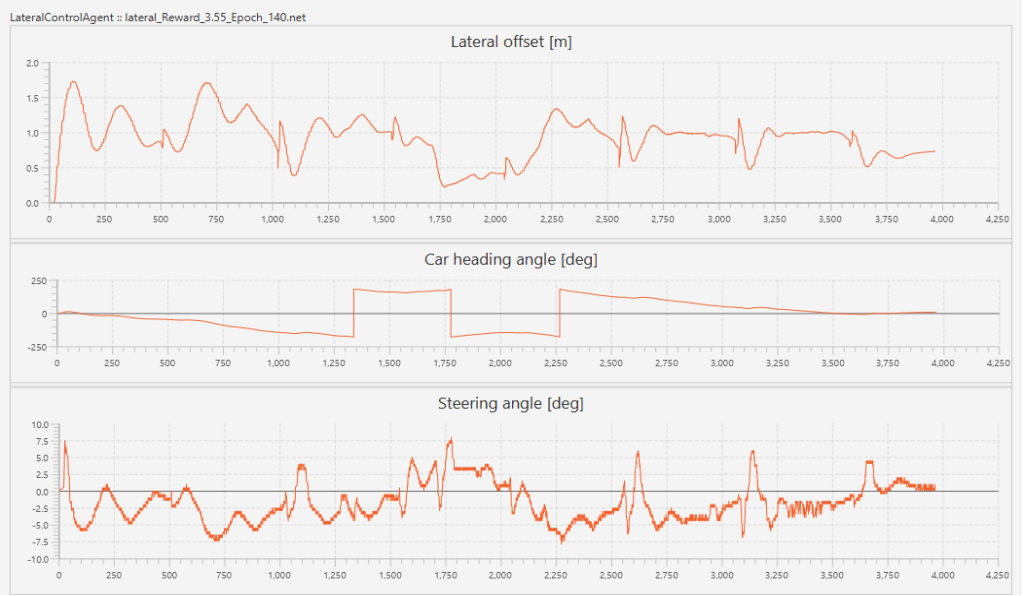


Figure 27. Lateral offset, Car heading angle, Steering angle of 2 actions for the second track. Extracted from Cooperative Traffic Simulator.

For the third track, due to the steep turns, the models we train are unable to complete the track with the change of half a degree. The best performing model reaches 341 m out of 1252 m from epoch 404 with reward 129.88.

Figure 28. Lateral offset, Car heading angle, Steering angle of 2 actions for the third track. Extracted from Cooperative Traffic Simulator.

- 3 actions, where the steering angle is turned right, left by half a degree or doesn't change. For the first track, we obtain the model from epoch 65 with reward 0.68 that is able to complete the entire track, with keeping the steering angle at zero, without any unnecessary turns.

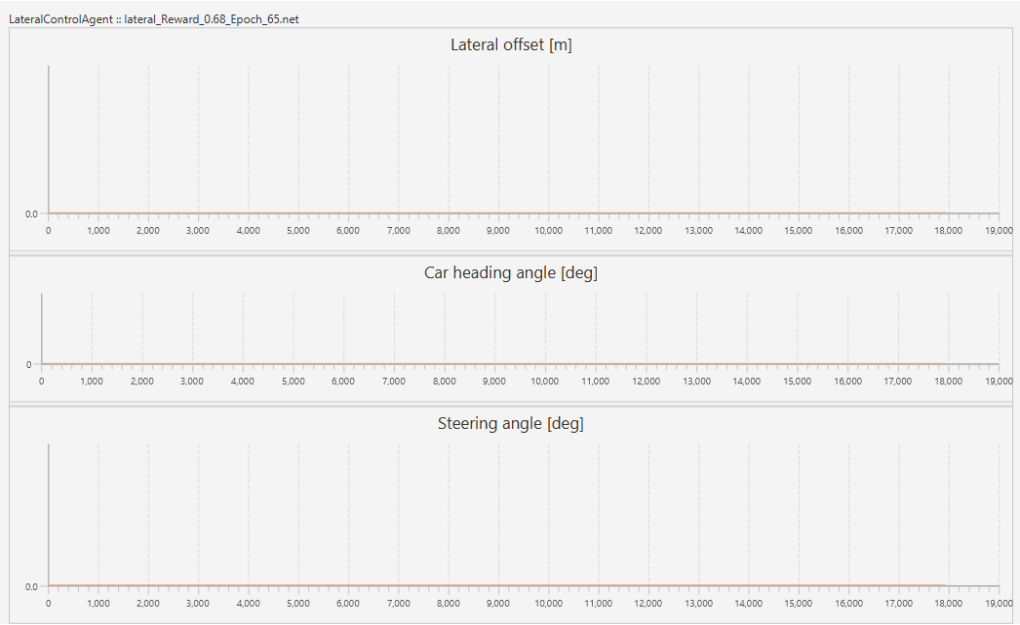


Figure 29. Lateral offset, Car heading angle, Steering angle of 3 actions for the first track. Extracted from Cooperative Traffic Simulator.

For the second track, we have a model that reaches 238 m out of 552 m of the track from epoch 174 with reward 71.06.

Figure 30. Lateral offset, Car heading angle, Steering angle of 3 actions for the second track. Extracted from Cooperative Traffic Simulator.

For the third track, we have a model that reaches 322 m out of 1565 m of the track from epoch 401 with reward 22.72.

Figure 31. Lateral offset, Car heading angle, Steering angle of 3 actions for the third track. Extracted from Cooperative Traffic Simulator.

- 4 actions, turn left with lower, higher power, turn right with lower, higher power,
  Action 0 turn left with higher power/ power++
  Action 1 turn left with lower power/ power --
  Action 2 turn right with lower power/ power--
  Action 3 turn right with higher power/ power ++

  We trained models that could finish all the tracks. One of them was the model from epoch 788 and reward 7.50.

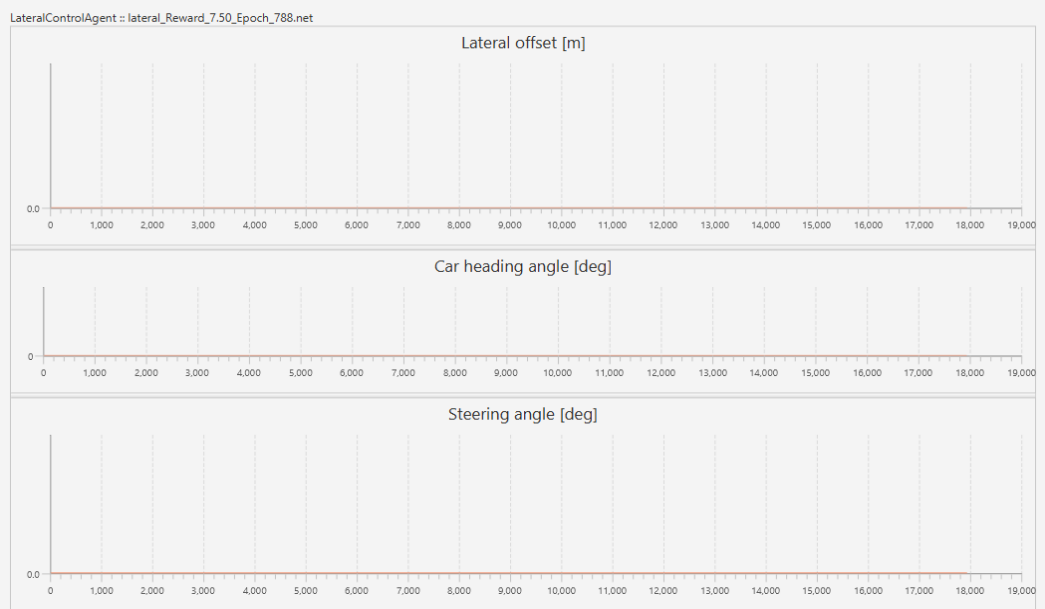


Figure 32. Lateral offset, Car heading angle, Steering angle of 4 actions for the first track. Extracted from Cooperative Traffic Simulator.

Figure 33. Lateral offset, Car heading angle, Steering angle of 4 actions for the second track. Extracted from Cooperative Traffic Simulator.

Figure 34. Lateral offset, Car heading angle, Steering angle of 4 actions for the third track. Extracted from Cooperative Traffic Simulator.

- 5 actions, turn left with lower, higher power, turn right with lower, higher power, keep steering wheel angle
  Action 0 turn left with higher power/ power++
  Action 1 turn left with lower power/ power --
  Action 2 turn right with lower power/ power--
  Action 3 turn right with higher power/ power ++
  Action 4 keep steering wheel angle
  For the first track we obtain the model from epoch 114 with reward 18.36 that is able to complete the full track.

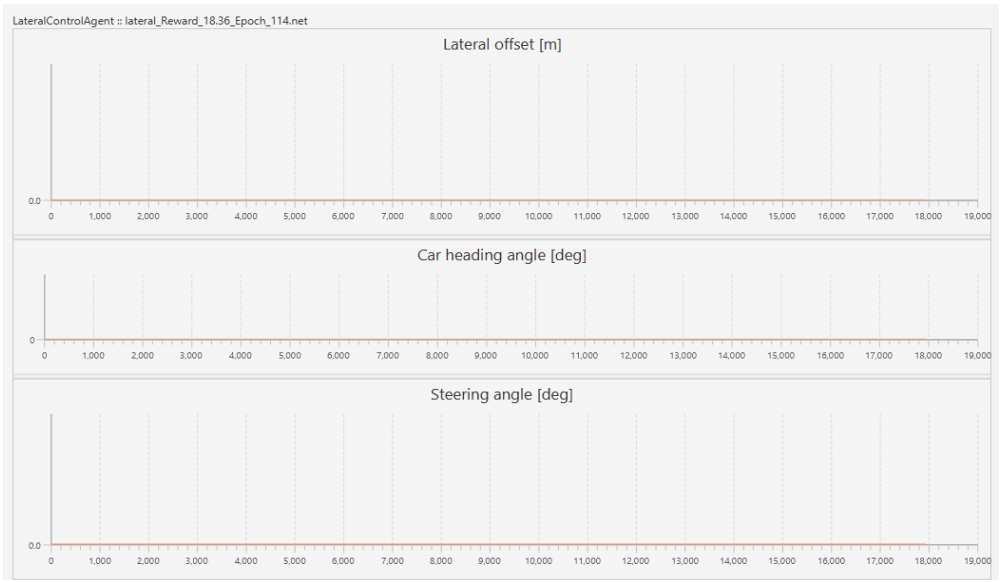


Figure 35. Lateral offset, Car heading angle, Steering angle of 5 actions for the first track. Extracted from Cooperative Traffic Simulator.

We obtain the model from epoch 684 with reward 4.35 that is able to complete both the second and the third track.

Figure 37. Lateral offset, Car heading angle, Steering angle of 5 actions for the second track. Extracted from Cooperative Traffic Simulator.

Figure 38. Lateral offset, Car heading angle, Steering angle of 5 actions for the third track. Extracted from Cooperative Traffic Simulator.

I. Conclusions of Lateral Control

The same DQN architecture as used for longitudinal control was able to learn lateral control as well.

The main advantages of our solution are:

- Any OpenDRIVE road can be trained and followed within the required precision.
    - Easy to add new roads, and the DQN model will learn any curvature or any sequence of geometry types defined by OpenDRIVE.
    - This solution fits perfectly to the original goal of standalone agents following a path on given test tracks.
- The concept of resampled point series to follow can be achieved in real applications.
    - Differential GPS tracks can be resampled with any resolution and fed to the system in real-time to compute lateral offset from the target road we follow.
- High precision maneuverability
    - The slightest difference along the track is followed by the model
    - Easy to simulate obstacles along the track
    - Easy to train low-speed control with precise maneuverability and high-speed control using the same model
- We can highlight the same advantages as with longitudinal control
    - Easy to train
    - The low number of parameters
    - Easy to deploy
    - Easy to change the vehicle model
- One can choose different vehicle model
    - Here we have been using the simplest kinematic bicycle model, but it is easy to change to the desired vehicle dynamics for more realistic needs
- Steering characteristics are fine-tunable
    - For simplicity here, we have selected a one-to-one gear ratio for the steering.
    - One can adapt the angular velocity, the number of turns, and the turn ratio between the steering wheel and the vehicle wheel as needed.
    - Easy to adapt the model to different steering characteristics

C. Physical Vehicle Models

We ensure that the physical vehicle model we have introduced accurately depicts and is close to reality. We have used a 2014 BMW i3 [13] and a 2015 Mercedes-Benz CLA 200 CDI (CLA 200 d) [14], and have set the factory defaults and car specifications for the length, the outer width, the wheelbase, and the top speed of the vehicle. The results for the longitudinal control were based on the 2014 BMW i3 [13], while the results for the later control and the cooperative driving model were based on the 2015 Mercedes-Benz [14]. We performed polynomial fitting to the acceleration factory limits in order to facilitate our problem and make it as close to reality as possible. The data was collected from [14] . The steering system has a minimum to maximum wheel angle from -0.524 to 0.524 radians or -30 to 30 degrees. The steering wheel angular velocity system ranges from 0 to 1.745 radians or 0 to 100 degrees.

5. Multi-agent systems

On the contrary to distributed problem solving, Multi-agent systems (MAS) [15] cope with the behavior of the computing entities accessible to solve a given problem. In a multi-agent system each computing entity is mentioned as an agent. MAS can be described as a network of individual agents that share knowledge and communicate with each other in order to solve a problem that is beyond the sphere of a single agent. Hence, a Multi-agent system, MAS, is a system of autonomous interacting intelligent agents acting in an environment to achieve a shared purpose [15]. The behaviour of the agents range from competitive to collaborative, depending on the circumstances.
We will look into how these scenarios could be implemented for intelligent traffic vehicle (ITV) agents. Unless it is stated otherwise for both collaborative and competitive scenarios the crash (and bump) should be avoided, while traffic rules might be kept.

A. Collaborative Scenarios

The agents pursue a common goal. This common goal could be one of the following outlines.

- Constant distance between 2 agents.
- Single lane environment.
- Infinite rural road section ahead.
- 2 ITV agents should follow a given trajectory while keeping in-between distance constant.
- Reaching the point of interest POI at the same time :
    - Multilane environment
    - City section ahead
    - 2 ITV agents should reach the same POI (or 2 different POIs) at the same time

B. Competitive Scenarios

Competitive agents are solely focused on maximizing their own utility. A few competitive scenarios are listed below.

- Overtake :
    - Multilane environment
    - Finite/infinite highway road section ahead
    - ITV agent 1 should try to overtake ITV agent 2
    - ITV agent 2 should try to avoid overtake
- Reaching POI first :
    - Multilane environment
    - City section ahead
    - 2 ITV agents are compete to should a POI first

6. Cooperative Driving Model

A. In-depth Description of the developed concept

The Cooperative driving model is considered as a promising driving pattern to outstandingly improve transportation efficiency and traffic safety.We will implement an Adaptive Cruise Control (ACC) agent, with the purpose to achieve an active safety system that automatically controls the acceleration and braking of the vehicle to keep safe distance from a trained longitudinal agent.

In detail, for our Deep Q- Learning network, we are using the same network we used for the longitudinal control, the Adam optimizer with the value of learning rate 0.001, the number of hidden nodes is 150, and the number of layers is 2. The input and output layers are organized according to the actions that we have decided and the state variables that we have. The state variables are six now : the velocity of the agent, the previous acceleration of the agent, the peer vehicle position, the peer vehicle acceleration and the distance between the agent and the peer vehicle. The number of actions we select are four. The input nodes are the state variables, and the output nodes are the number of actions we wish to train our agent.

We have implemented the reward function that calculates the distance between the longitudinal control agent and the adaptive cruise control agent we are training. If the distance between the two vehicles is between 0 and 50 meters, we reward our adaptive control agent; else, we punish him.

B. Results

I. Trained Scenario

As for the trained scenario of the cooperative drive model, we will train the adaptive vehicle to follow the peer vehicle as close a distance as possible while avoiding collision and surpassing it. Specifically, we will use the track length of 1000 m in a straight line; where we will introduce the peer vehicle and the adaptive vehicle. For the peer vehicle, we will use a longitudinal model from epoch 76 with reward -2522.81 and one from epoch 44 with reward - 2424.75, trained before that perform best with 4 actions and 5 actions respectively for the track length of 1000 m; following the specific speed limits. We will conclude that four actions were preferred over five actions for training both the adaptive and the peer vehicle, since we received better results, the distance between the two vehicles is smaller. The major difference, between four and five actions, is the instant ability for five actions to reach zero acceleration, a fact that is not crucial since the peer vehicle, which is leading, will constantly have to adjust its speed to the traffic speed signs.

II. Trained Agents

For 4 actions we achieved the best result for the epoch 171 and the reward is -185. We export the following charts : the velocity and acceleration of the adaptive vehicle, the velocity and acceleration of the peer vehicle and the distance between them. The adaptive vehicle is able to follow our peer vehicle , without colliding while keeping a safe distance at most 150 meters. When the peer vehicle reaches the end of the track 1000 meters, the adaptive vehicle reaches 954 meter.

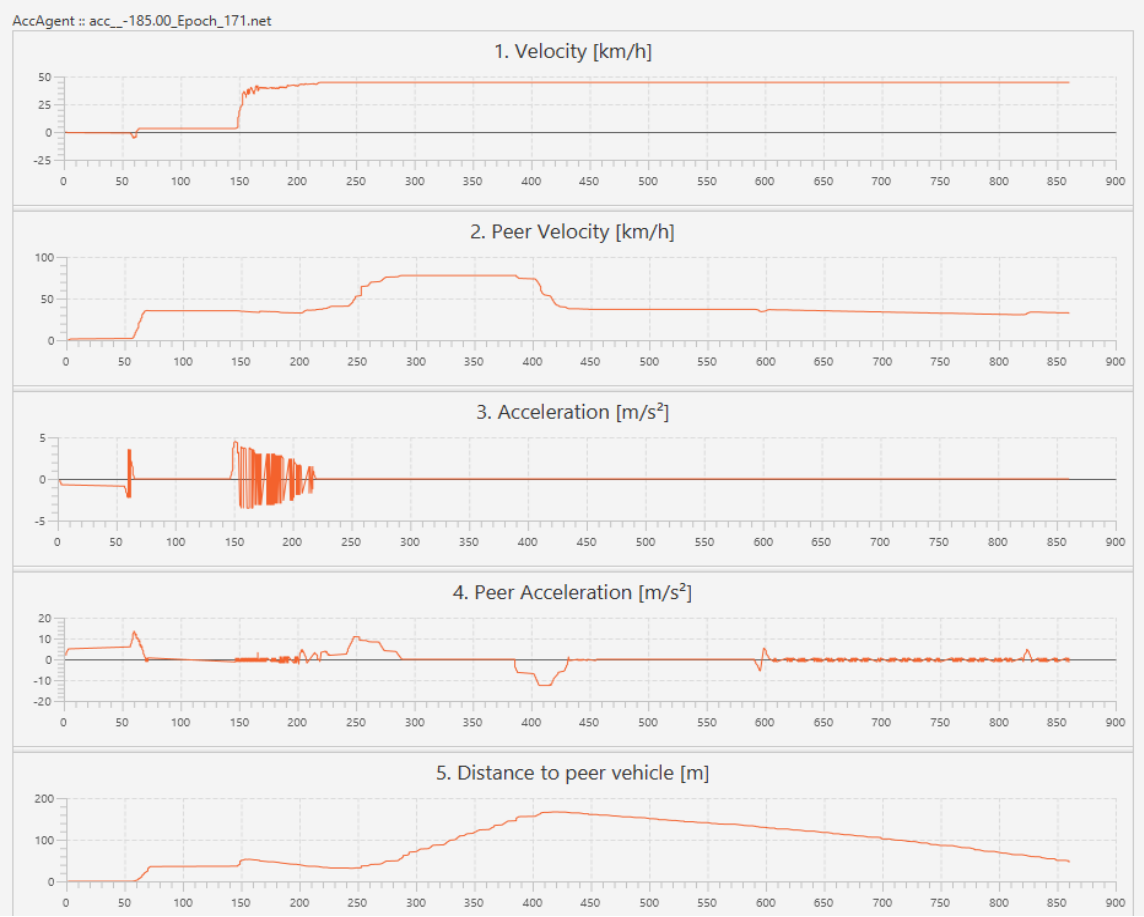


Figure 39. Velocity, Peer Velocity, Acceleration, Peer Acceleration, Distance to peer vehicle for 4 actions. Extracted from Cooperative Traffic Simulator.

For 5 actions we achieved the best result for the epoch 107 and the reward is -141.50. The adaptive vehicle is able to follow our peer vehicle, without colliding while keeping a safe distance at most 200 meters. When the peer vehicle reaches the end of the track 1000 meters, the adaptive vehicle reaches 943 meter. We can understand clearly why the four actions are preferred over five actions. The results are below :

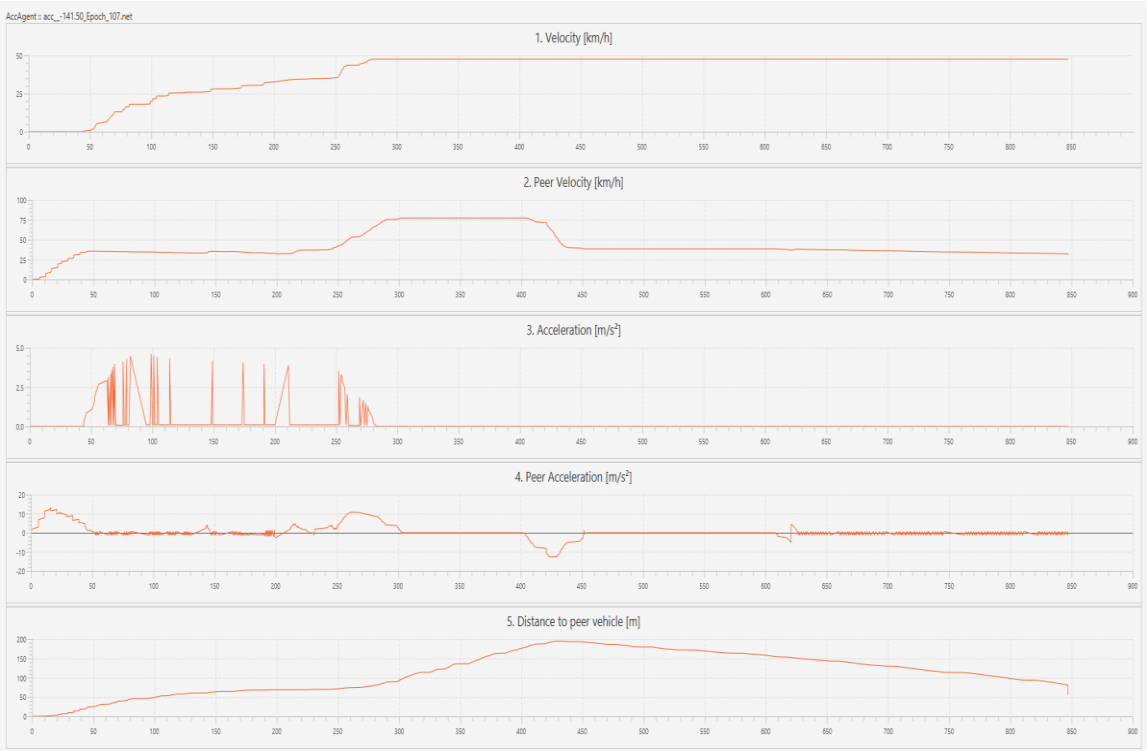


Figure 40. Velocity, PeerVelocity, Acceleration, Peer Acceleration, Distance to peer vehicle for 5 actions. Extracted from Cooperative Traffic Simulator.

C. Answers

I. How far can we rely on Deep Reinforcement Learning (DRL) techniques for stable vehicle control?

Our primary goal was to find an alternative to current control algorithms where the configuration and parameter tuning does require much less effort. We want to be able to support many different vehicle types and that is not an option to design its own fully customized control for each vehicle. Our driving method is somewhat different (pushing the pedals instead of using the vehicle's mechanical actuators) that also requires further adaptations.

Reinforcement learning offers a great alternative and, in this thesis, we have proven its potential. By defining the proper environment variables and a simple rewarding concept the RL agents are able to adapt their speed to any speed profiles (up to their dynamic capabilities) and follow any road segments. Separation of lateral and longitudinal control is essential to us while one longitudinal control agent is pushing the pedals (throttle, brake), and later control agent is driving the steering wheel rotation.

These agents can adapt to extreme curvatures and tricky turn combinations. They can follow different strategies for comfortable acceleration, smooth maneuvers, low consumption, and all these are relatively easy to be programmed into the agent. We have added a large amount of noise into the system and it was still able to achieve its task. That suggests that on top of simulations real applications might benefit from our models.

How far we can rely on these agents now seems like just to be a matter of training effort. Since the complexity of the neural network behind can be increased drastically (right now just using a shallow architecture) the precision and accuracy of the control seem to be limitless.

II. What is the dependency of DRL Agents on the given physical model?

We have been using two different vehicle models BMW i3 [13] and Mercedes CLA [14]. The longitudinal control agent is able to achieve its goals with both variants. Before this study, we thought that one trained model might fit all vehicle types and that would be a great achievement. Today we can see that changing the vehicle model behind the agent is just one line of code and the training effort is relatively small. It turns out that in order to have the best possible control qualities agents should be trained on each vehicle's physical model.

Instead of one size fits all approach we suggest to beneficially retrain each vehicle for the highest precision possible.

III. How can multiple Agents interact within the simulation environment?

We have successfully trained an adaptive cruise control agent that is able to follow agents with their own longitudinal control. The cruise controller has been given a challenging task. Stay 50 m behind the leading vehicle. This is a continuously changing dynamic environment and our RL agent was able to get close to the required precision.

In our controlled vehicle testing environment, we can define a closed wireless network between the ITVs (Intelligent Test Vehicle) and distribute the speed, acceleration and other real-time measured sensory data among all the ITVs and peer vehicles. Thanks to this concept all the traffic participants can have their own goal and their own trained agent accordingly. This technique can successfully provide the Cooperative Driving Model.

D. Lessons Learnt

1. In contrast to supervised learning techniques the training curves are not necessarily reflecting the agent's performance. The accumulated reward is neither a good indicator of the expected outcome of the trained model. During the validation of the agent, one has to provide his own metrics for performance evaluation.

2. The reward function might not contain states derived from the environmental states. If you see a long list of conditional statements in your reward function then the strategy requires a revision. Better to make the reward function stateless while in a complex environment one cannot keep up with the conditions of the overall system.

3. Selected system variables have a big impact on the agents' capabilities. It seems that there is an optimal number of variables describing the environment. Adding more or adding less than needed can lead to failure. It is not a good strategy to add all sensory information available and let's see if the neural network can converge.

4. The action set of the control agent might be kept relatively low (depending on the environment). We have achieved what we wanted with 4 actions in general. Sometimes by adding more actions the performance has been decreased. For most of our cases the action set might be reduced to only 2 but with poorer performance.

5. We have seen that in the trained agent's behavior oscillations might appear just like in other control methods. Chirp frequencies and excitation of ever-increasing amplitudes have been recognized. For a high precision control agent, all aspects of the system require precision just as with traditional control methods. The idea that the neural network will smooth all the imperfections is not necessarily true.

6. As with classical lateral controllers we have been facing the same challenges of training the ability of the agents to pass through wild turn combinations and stay on track on simple straight road sections as well. This was one of the rare points when increasing the number of actions set, we had better results.

E. Summary

In this thesis, we have presented an alternative vehicle control approach focusing on the requirements of a driver robot-driven closed environmental test traffic generator approach. We have built up the foundations of the so-called Cooperative Driver Model that generates complex traffic scenarios for autonomous vehicles. We have been able to successfully enroll in a massive training campaign with gradually increasing complexity. We have trained longitudinal agents obeying the speed limits or adapting their speed according to a given specific distance between the agent in front. The lateral control agent can pass through tricky turn combinations and follow long sections of straight roads as well.

As a next step, we will leave the simulation environment and try these control agents on real ITVs (Intelligent Test Vehicles) on closed test tracks with real sensory inputs.

7. References

[1] Sebastian Thrun. 2010. Toward robotic cars. Commun. ACM 53, 4 (April 2010), 99–106. DOI:https://doi.org/10.1145/1721654.1721679.

[2] S. K. Gehrig and F. J. Stein, "Cartography and dead reckoning using stereo vision for an autonomous car," Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348), 1999, pp. 30-34 vol.4, DOI: 10.1109/ICIP.1999.819461.

[3] Ji Hyun Yang, Jimin Han, and Jung-Mi Park. 2017. Toward Defining Driving Automation from a Human-Centered Perspective. In Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '17). Association for Computing Machinery, New York, NY, USA, 2248–2254. DOI:https://doi.org/10.1145/3027063.3053101.

[4] Sievers, Gregor, et al. "Driving Simulation Technologies for Sensor Simulation in SIL and HIL Environments." DSC 2018 Europe (2018).

[5] ASAM e. V.: OpenSCENARIO 2.0.0. ASAM OpenSCENARIO: Version 2.0.0 Concepts.

[6] Pellegrini, Paola, Grégory Marlière, and Joaquin Rodriguez. "Real time railway traffic management modeling track-circuits." ATOMOS 2012, 12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems. 2012.

[7] Pellegrini, Paola, et al. "RECIFE-MILP: An effective MILP-based heuristic for the real-time railway traffic management problem." IEEE Transactions on Intelligent Transportation Systems 16.5 (2015): 2609-2619.

[8] Waslander, S. & Kelly, J., n.d. Introduction to Self-Driving Cars, Toronto: Coursera.

[9] ASAM e. V.: Von OpenDRIVE 1.6.1. OpenDRIVE 1.6.1 (asam.net)

[10] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[11] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

[12] OpenRoadEd.net (2018). OpenRoadEd - Home. [online] Available at: https://sourceforge.net/projects/openroaded/

[13] "2015 Mercedes-Benz CLA 200 CDI (CLA 200 d) Shooting Brake" https://www.automobile-catalog.com/car/2015/2107640/mercedes-benz_cla_200_cdi_shooting_brake.html.

[14] "2014 BMW i3 with Range Extender (aut. 1) (model for Europe) car specifications & performance data review" https://www.automobile-catalog.com/car/2014/1940060/bmw_i3_with_range_extender.html

[15] Balaji, P. G., and D. Srinivasan. "An introduction to multi-agent systems." Innovations in multi-agent systems and applications-1. Springer, Berlin, Heidelberg, 2010. 1-27.