Thesis for the degree of doctor of philosophy

Decision-Making in Autonomous Driving using Reinforcement Learning

Introducing an uncertainty-aware approach

CARL-JOHAN HOEL

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Decision-Making in Autonomous Driving using Reinforcement Learning
Introducing an uncertainty-aware approach
CARL-JOHAN HOEL
ISBN 978-91-7905-584-4

© CARL-JOHAN HOEL, 2021

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5051
ISSN 0346-718X

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE–412 96 Gothenburg
Sweden
Telephone: +46 (0)31–772 1000

Chalmers Digitaltryck
Gothenburg, Sweden 2021

Decision-Making in Autonomous Driving using Reinforcement Learning
Introducing an uncertainty-aware approach
CARL-JOHAN HOEL
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

The main topic of this thesis is tactical decision-making for autonomous driving. An autonomous vehicle must be able to handle a diverse set of environments and traffic situations, which makes it hard to manually specify a suitable behavior for every possible scenario. Therefore, learning-based strategies are considered in this thesis, which introduces different approaches based on reinforcement learning (RL).

A general decision-making agent, derived from the Deep Q-Network (DQN) algorithm, is proposed. With few modifications, this method can be applied to different driving environments, which is demonstrated for various simulated highway and intersection scenarios. A more sample efficient agent can be obtained by incorporating more domain knowledge, which is explored by combining planning and learning in the form of Monte Carlo tree search and RL. In different highway scenarios, the combined method outperforms using either a planning or a learning-based strategy separately, while requiring an order of magnitude fewer training samples than the DQN method.

A drawback of many learning-based approaches is that they create black-box solutions, which do not indicate the confidence of the agent's decisions. Therefore, the Ensemble Quantile Networks (EQN) method is introduced, which combines distributional RL with an ensemble approach, to provide an estimate of both the aleatoric and the epistemic uncertainty of each decision. The results show that the EQN method can balance risk and time efficiency in different occluded intersection scenarios, while also identifying situations that the agent has not been trained for. Thereby, the agent can avoid making unfounded, potentially dangerous, decisions outside of the training distribution.

Finally, this thesis introduces a neural network architecture that is invariant to permutations of the order in which surrounding vehicles are listed. This architecture improves the sample efficiency of the agent by the factorial of the number of surrounding vehicles.

Keywords: Autonomous driving, reinforcement learning, tactical decision-making, Monte Carlo tree search, aleatoric uncertainty, epistemic uncertainty, neural networks.

To my family.

Acknowledgments

The road to a PhD degree is long and winding. Sometimes things work out easier than expected, but more often than not, unexpected obstacles are lurking behind the next corner. Therefore, I am highly grateful to all the people who in different ways have taken part in this journey, by providing guidance, support, or sometimes just a much-needed laugh.

First of all, I would like to express gratitude to my supervisors Prof. Krister Wolff and Prof. Leo Laine for guiding me through difficult choices and always believing in me. I am also thankful to my examiner Prof. Mattias Wahde for welcoming me as a PhD student and for raising the bar of my scientific writing skills. I would not have been able to start my PhD studies without the support from my former managers at Volvo, Inge Johansson and Stefan Edlund. Later Anna Wrige Berling and currently Julia Nilsson have also been excellent managers who continued to provide a solid base at Volvo.

I would like to thank my colleagues at AI Sweden, the VEAS division at Chalmers, and the Driver and Vehicle analysis groups at Volvo for interesting discussions and a friendly working environment. Especially during the pandemic, the virtual 'fika' sessions have provided revitalizing breaks from the long hours of writing. I am also very grateful to Prof. Mykel Kochenderfer and Prof. Katie Driggs-Campbell at Stanford for welcoming me to SISL for an inspiring semester, which gave me many valuable insights and new perspectives.

Last but not least, this work was financially supported by Volvo Group, the Wallenberg Artificial Intelligence, Autonomous Systems, and Software Program (WASP), and Vinnova FFI. This support is gratefully acknowledged.

List of included papers

The following publications form the foundation of this thesis. The author of the thesis was the main contributor to Paper I - IV and VI. Paper V was written in cooperation with Tommy Tram and both authors contributed equally to this work.

I. C. J. Hoel, M. Wahde, and K. Wolff, *An evolutionary approach to general-purpose automated speed and lane change behavior*, in Proceedings of the 16th IEEE International Conference on Machine Learning and Applications, Cancun, Mexico, 2017, pp. 743-748.

II. C. J. Hoel, K. Wolff, and L. Laine, *Automated speed and lane change decision making using deep reinforcement learning*, in Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems, Maui, HI, USA, 2018, pp. 2148-2155.

III. C. J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, *Combining planning and deep reinforcement learning in tactical decision making for autonomous driving*, in IEEE Transactions on Intelligent Vehicles, vol. 5, no. 2, pp. 294-305, 2020.

IV. C. J. Hoel, K. Wolff and L. Laine, *Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation*, in Proceedings of the IEEE Intelligent Vehicles Symposium, Virtual conference, 2020, pp. 1563-1569.

V. C. J. Hoel, T. Tram, and J. Sjöberg, *Reinforcement Learning with Uncertainty Estimation for Tactical Decision-Making in Intersections*, in

Proceedings of the 23rd IEEE International Conference on Intelligent Transportation Systems, Virtual conference, 2020, pp. 1-7.

VI. C. J. Hoel, K. Wolff, and L. Laine, *Ensemble Quantile Networks: Uncertainty-Aware Reinforcement Learning with Applications in Autonomous Driving*, submitted to IEEE Transactions on Neural Networks and Learning Systems, 2021.

Table of Contents

Abstract **i**

Acknowledgments **v**

List of included papers **vii**

1 Introduction **1**
1.1 Decision-making for autonomous driving 1
1.2 Approach 2
1.3 Limitations 5
1.4 Contributions 5
1.5 Thesis outline 6

2 Related work **7**
2.1 Rule-based and planning-based methods 7
2.2 Learning-based methods 9

3 Technical background **13**
3.1 Markov decision processes 13
3.2 Reinforcement learning 15

4 Model-free RL approaches **17**
4.1 Simulated driving scenarios 18
4.2 Value-based RL, DQN agent 19
4.2.1 Approach 19
4.2.2 Results and discussion 21

4.3 Policy-based RL, GA agent 24
4.3.1 Approach 24
4.3.2 Results and discussion 26

5 Combining planning and RL **29**
5.1 Approach 30
5.2 Simulated experiments 32
5.3 Results and discussion 34

6 Uncertainty of RL-based agents **37**
6.1 Approach 38
6.1.1 Epistemic uncertainty 38
6.1.2 Aleatoric uncertainty 40
6.1.3 Aleatoric and epistemic uncertainty 42
6.2 Simulated experiments 43
6.3 Results and discussion 45

7 Discussion **51**
7.1 Generality 51
7.2 Sample and computational complexity 52
7.3 Safety 54
7.4 MDP formulation 54
7.5 Neural network architecture 55

8 Conclusions and future work **57**
8.1 Concluding remarks 57
8.2 Future research directions 59

Bibliography **61**

INCLUDED PAPERS

Chapter 1

Introduction

Autonomous driving technology is expected to have a massive effect on the current transportation system and benefit society in many ways. For example, over one million people are killed in traffic-related accidents each year, where the vast majority of the accidents are caused by human mistakes [66, 59]. By removing humans from the control of the vehicles, autonomous driving could significantly improve the traffic safety. Furthermore, the productivity of commercial heavy vehicles is increased when fewer human drivers are required, and the road infrastructure could be utilized more efficiently by scheduling transports outside of rush hours, for example during nights [17].

Major progress towards deploying autonomous vehicles has been made during the last decade. The perception systems have been remarkably improved, largely due to the success of deep learning techniques [27]. The low-level control of the vehicle is a mature research area and can be solved with classical control theory methods [49]. However, how to approach the high-level decision-making in complex traffic situations is less explored and forms the main topic of this thesis.

1.1 Decision-making for autonomous driving

The decision-making task of an autonomous vehicle is commonly divided into three different levels: strategic, tactical, and operational decision-making [41], also referred to as navigation, guidance, and stabilization [70].

The strategic level considers the high-level goals of a transport mission and handles the route planning, whereas the tactical decision-making level modifies the strategic plan in order to adapt to the current traffic situation. The tactical decisions could for example consider when to change lanes in a highway driving situation, or whether to stop at, or drive through, an intersection. Finally, the operational decision-making level translates the desired maneuvers of the tactical level into a detailed trajectory, which is used to control the actuators of the vehicle. The focus of this thesis is the tactical decision-making level.

A tactical decision-making agent for an autonomous vehicle needs to manage a diverse set of environments, which could range from structured highway driving to complex urban surroundings. To anticipate all possible traffic situations that an autonomous vehicle should be able to handle, and manually code a suitable behavior for all of them, would be extremely time consuming and error prone, if at all possible. Furthermore, a tactical decision-making agent needs to interact with other traffic participants and consider a varying degree of uncertainty of the current state. Sensor imperfections and occlusions provide an incomplete view of the physical state of the environment, while the intentions of the surrounding traffic participants are not directly observable. This limited information results in a high uncertainty of the future development of a traffic scene, which needs to be considered when making decisions.

1.2 Approach

Many promising methods for tactical decision-making in autonomous driving already exist, see Chapter 2 for a broad overview. However, most of these methods are tailored for limited driving scenarios and do not scale to the complexity of the real world. Different methods could be combined for different scenarios, but such an iterative approach, if at all feasible, would result in an extremely complex system. An illustrative analogy, partly adapted from the book by Russell and Norvig [53], would be to try to reach the moon by building a ladder. It may seem like the project makes progress every day when a new section is added, but nevertheless, the ladder will never reach the goal. Instead, one may take a step back and approach the task in a fundamentally different way, by building a rocket. The progress towards the goal will be less apparent during the construction process, compared to extending

the ladder, but when finished, the rocket has the potential to actually reach the moon.

This thesis is based on the assumption that it may not be feasible to create a tactical decision-making agent by incrementally improving manually specified systems. Instead, strategies that are based on learning a suitable behavior from data are explored, which have the potential to generalize to all types of driving scenarios. From a machine learning perspective, supervised learning or reinforcement learning (RL) techniques could be considered. To use a supervised learning approach for directly learning how to make decisions from human driver demonstrations suffers from a distributional shift between training and test data, which is further discussed in Chapter 2. In contrast, RL deals with sequential decision-making problems where some form of utility is maximized [63], which is a natural way to formulate the task of a decision-making agent for autonomous driving. Therefore, the overarching purpose of this thesis is to investigate how RL methods can be used in practice to create a tactical decision-making agent for autonomous vehicles, and thereby supply a few early components to the rocket that will hopefully yield a general decision-making agent. More specifically, the following research questions (RQs) are considered:

RQ 1: How can a general decision-making agent for autonomous driving be created through RL? (Chapter 4, Paper I and II)
This thesis introduces, analyses, and compares a policy-based and a value-based RL algorithm, which are tested in different simulated highway driving scenarios. The first approach uses a genetic algorithm (GA) [24] to automatically generate a rule based decision-making agent, whereas the second approach is based on a Deep Q-Network (DQN) agent [42]. The second approach also introduces a new way of applying a convolutional neural network architecture [38] to a high-level state description of interchangeable objects, which significantly improves both the training speed and the quality of the final agent. Both approaches manage to navigate around 10% faster through dense traffic compared to a commonly used baseline method, consisting of the Intelligent Driver Model (IDM) [67] for the longitudinal motion and Minimizing Overall Braking Induced by Lane changes (MOBIL) model [31] for the lateral motion. However, the main benefit of the two approaches, compared to classical methods, is that they are general, i.e., not limited to a specific driving scenario.

RQ 2: In what way can domain knowledge be incorporated into an RL agent? (Chapter 5, Paper III)
Model-free RL approaches to decision-making problems provide general methods that use little domain knowledge. Nevertheless, in autonomous driving, some kind of model of the surrounding traffic is often available and could be exploited to improve the quality of the decisions. Therefore, this thesis introduces a third approach, which incorporates more domain knowledge and combines the concepts of planning and learning, in the form of Monte Carlo tree search (MCTS) [9] and deep reinforcement learning. This method is inspired by the AlphaGo Zero algorithm [58], which is here first extended to a domain with a continuous state space and where self-play cannot be used, and then adopted to the autonomous driving domain. A general tactical-decision making framework is introduced and tested in different simulated highway driving scenarios, where it performs significantly better than the IDM/MOBIL model. The strength of combining planning and learning is also illustrated by a comparison to using the Monte Carlo tree search or the learned policy separately.

RQ 3: How can an RL-based agent provide an uncertainty estimate of its decisions? (Chapter 6, Paper IV, V, and VI)
Previous RL-based approaches provide black-box solutions, which do not offer information on how confident the trained agent is about its decisions. An estimate of the agent's uncertainty is fundamental for real-world applications of autonomous driving. Uncertainty stem from stochasticity of the environment (aleatoric uncertainty), e.g., in situations with occlusions, or lack of knowledge (epistemic uncertainty) in situations that the agent has not been trained for [33]. This thesis introduces the Ensemble Quantile Networks (EQN) method, which combines distributional RL with an ensemble approach, to obtain an estimate of both the aleatoric and the epistemic uncertainty. A criterion for classifying which decisions that have an unacceptable uncertainty is also introduced. The results show that the EQN method can balance risk and time efficiency in different occluded intersection scenarios, by considering the estimated aleatoric uncertainty. Furthermore, in both different highway and intersections scenarios, it is shown that the epistemic uncertainty information of the trained agent can be used to identify situations that are outside of the training distribution. By applying a backup policy in such situations, arbitrary decisions are avoided, which increases the safety of the agent.

1.3 Limitations

The following aspects of using RL to create a tactical decision-making agent for autonomous driving are not considered in this thesis:

- Whether it is possible to guarantee safety through a learning-based method is an open question, and out of the scope of this thesis. While the uncertainty estimation methods of Chapter 6 can be used to increase the safety, an underlying safety layer is assumed to always monitor the decisions of the RL-based agents.
- The presented algorithms have been tested in simulated environments. How an agent that has been trained in simulation can be transferred into the real world is an interesting research topic, but is out of the scope of this thesis, although some aspects are touched upon in Chapter 6.
- This work only considers control of one vehicle and does not include multi-agent control aspects.
- Simple Markov decision process (MDP) formulations have been used in this thesis, to provide clear interpretations and analyses of the results. In a real-world application, the representation of the state, the choice of action space, and the composition of the reward function will likely require more complex design choices, which is further discussed in Section 7.4.

1.4 Contributions

The main contributions of this thesis are:

- Two conceptually general approaches to creating a tactical decision-making agent, which both show promising results by outperforming the IDM/MOBIL model in different highway driving scenarios (Chapter 4 and Paper I, II).
- An extension of the AlphaGo Zero algorithm, which allows it to be used in domains with a continuous state space and where self-play cannot be used (Chapter 5 and Paper III).

- The extension of two RL methods, which provide an estimate of the aleatoric or epistemic uncertainty, respectively, together with confidence criteria, which can be used to identify situations with high uncertainty (Chapter 6 and Paper IV, V, VI).
- The EQN algorithm, which simultaneously quantifies both the aleatoric and epistemic uncertainty of a trained agent (Chapter 6 and Paper VI).
- A neural network architecture that is invariant to permutations of the order in which surrounding traffic participants are listed, which speeds up the training process and improves the quality of the trained agent (Chapter 4 and Paper II).
- A comparison and analysis of the properties of the introduced approaches (Chapter 7).

1.5 Thesis outline

A review of related work is presented in Chapter 2, which is followed by a brief background to Markov decision processes and reinforcement learning in Chapter 3. This chapter also introduces the notation and terminology that are used in the subsequent chapters. Chapters 4, 5, and 6 describe the respective research questions of Section 1.2 in more detail and present approaches to address them, together with an overview of the main results. The properties of the different approaches are discussed and compared in Chapter 7. Finally, Chapter 8 provides some concluding remarks and future research directions.

Chapter 2

Related work

A large variety of approaches to the sequential decision-making task of autonomous driving has been presented in the literature. This chapter gives a broad introduction to the main research directions, but it does not aspire to provide a comprehensive survey of each class of approaches.

2.1 Rule-based and planning-based methods

Early approaches to tactical decision-making for autonomous vehicles often used rule-based methods, commonly implemented as handcrafted state machines. For example, during the DARPA Urban Challenge, a rule-based approach was adopted by the winning team from the Carnegie Mellon University, where different modules handled the behavior for the different driving scenarios that were encountered [72]. Other participants, such as Stanford and Virginia Tech, used similar strategies [2, 43]. While successful for a limited and controlled environment such as the Urban Challenge event, it is unlikely that rule-based approaches could scale to handle the complexity and diversity of real-world driving.

Another group of algorithms treats the decision-making task as a motion planning problem. Commonly, a prediction model is used to predict the motion of the other agents, and then the behavior of the vehicle that is being controlled, henceforth referred to as the ego vehicle, is planned accordingly. This results in a reactive behavior, since the predictions are independent of the ego vehicle plan. Therefore, interaction between the ego vehicle and other

agents is not explicitly considered, but may happen implicitly by frequent replanning. Search-based planners typically discretize the state space and then apply Dijkstra's algorithm [15] or one of the algorithms from the A* family [19]. These techniques were also commonly used during the DARPA Urban Challenge [2, 43]. However, since real-time performance can be hard to achieve with graph-search algorithms, sampling-based algorithms such as rapidly-exploring random trees [37] have been used for motion planning, e.g., by Karaman et al. [28]. A third approach to solve the motion planning problem is to use optimization-based techniques, for example optimal control, which was applied to highway driving scenarios by Werling et al. [76]. Since human behavior is complex and varies between individuals, some algorithms use a probabilistic prediction as input to the motion planning. This is for example shown in a study by Damerow et al. [14], which aims to minimize the risk during an intersection scenario. Additional approaches to motion planning for autonomous driving are provided in the surveys by Gonzáles et al. [18] and Paden et al. [49].

It is common to model decision-making problems as Markov decision processes or partially observable Markov decision processes (POMDPs) [34]. This mathematical framework allows modeling of uncertainty in the current state, uncertainty in the future development of the traffic scene, and modeling of an interactive behavior. The task of finding the optimal policy for a POMDP is most often intractable, but many approximate methods exist. One way to group these methods is in offline and online methods. There are powerful offline algorithms for planning in POMDPs, which can solve complex situations. One example is shown by Brechtel et al., which proposes a solution to how measurement uncertainty and occlusions in an intersection can be handled [8]. In their work, an offline planner precomputes the policy by using a state representation that is learned for the specific scenario. A similar approach is adopted by Bai et al. for an intersection scenario [3]. The main drawback of these offline methods is that they are designed for specific scenarios. Due to the large number of possible real-world scenarios, it is challenging to precalculate a policy that is generally valid.

Online methods compute a policy during execution, which makes them more versatile than offline methods. However, the limited available computational resources require a careful problem formulation and limit the solution quality. Ulbrich et al. use a POMDP framework to make decisions on when to change lanes during highway driving [69]. In order to make it possible to solve the problem with an exhaustive search, a problem-specific high-level

state space is created, which consists of states that represent whether or not a lane change is possible or beneficial. However, due to the specialized state space, it is hard to generalize this method. Another online approach for solving a POMDP is the family of Monte Carlo tree search algorithms [9], which is used by Sunberg et al. to make lane changing decisions on a highway [62]. In order to handle the continuous state description, the tree search is extended with a technique called progressive widening [11]. Furthermore, other drivers' intentions are estimated with a particle filter. A hybrid approach between offline and online planning is pursued in a study by Sonu et al., where a hierarchical decision-making structure is used [60]. The decision-making problem is modeled on two levels as MDPs, since full observability is assumed. The high-level MDP is solved offline by value iteration and the low-level MDP is solved online with MCTS.

2.2 Learning-based methods

In planning-based methods, different algorithms are used to find a policy that maximizes the utility of the agent's behavior by using a model of the system. However, data-driven approaches are fundamentally different, since with this class of methods, an agent instead learns how to behave from observing data. This section describes different types of data-driven approaches that have been explored for autonomous driving.

An intuitive approach is to collect data from when an expert is performing a task and then use supervised learning to imitate the behavior of the expert. This method is often referred to as behavioral cloning and was first applied to autonomous driving in the ALVINN project [51]. Unfortunately, for many cases, behavioral cloning suffers from compounding errors, which refers to the problem when small mistakes gradually push the agent further away from the training distribution, into states from which the agent does not know how to recover. This problem can be mitigated by an active learning approach, where an expert can be queried during the training process [52], which for example has been applied to autonomous driving by Kelly et al. [29]. An alternative is to synthesize data by perturbing the expert's driving, which was done in a study by Bansal et al. [4]. Generative adversarial imitation learning [21] provides another method to handle the compounding error problem and has showed promising results in different highway driving scenarios [36].

Reinforcement learning is conceptually different from supervised learning, since labeled input-output samples are not available. Instead, the agent learns how to make decisions from interacting with the environment. RL methods are versatile, and have proven successful in various domains, such as playing Atari games [42], in continuous control [40], reaching a super human performance in the game of Go [58], and beating the best chess computers [57]. One advantage of RL methods, compared to planning based methods, is that a model of the environment is not required, i.e., the transition probabilities between different states are not assumed to be known. Furthermore, many RL methods provide a general framework and an agent could, in theory, learn how to behave correctly in all possible driving situations. During the last few years, many papers have addressed the task of applying RL approaches to autonomous driving. For example, DQN-based agents were used by Tram et al. [65] and Isele et al. [26] for navigating through different intersection scenarios, with varying driver intentions and occlusions. Commonly, a high-level action space is used together with the DQN algorithm. Other studies use policy gradient RL methods to directly control the speed and the steering angle of the vehicle, for example in lane changing and urban scenarios [73, 10]. Safety of the RL-based agents has been addressed by restricting dangerous actions, either by heuristics [44], linear temporal logic [7], or using an underlying motion planner with hard constraints [55]. A majority of these studies perform both the training and evaluation in simulated environments, whereas some train the agent in a simulator and then apply the trained agent in the real world [50, 4], or for some limited scenarios, the training itself is also performed in the real world [30]. Overviews of RL-based studies for autonomous driving are given by Kiran et al. [32] and by Ye et al. [77].

Both planning-based and RL-based methods use a reward function to find a policy that maximizes the cumulative future reward. However, for complex tasks such as autonomous driving, the design of the reward function is in itself a complicated task. For limited scenarios, the reward function can be manually specified and tuned until the agent finds a desired behavior, which is referred to as reward shaping [45]. However, for realistic scenarios, the number of possible reward features is massive and how to balance rewards related to, e.g., safety and efficiency is a complex issue. A practical approach is to instead learn the reward function from the behavior of human drivers by inverse reinforcement learning (IRL) [46]. An IRL approach was for example used by Kuderer et al. to learn the individual preferences of human drivers

with different driving styles [35]. Sharifzadeh et al. combined IRL with DQN and Wang et al. used an adversarial IRL approach to simultaneously obtain both the reward function and the policy for different lane-changing scenarios [56, 74]. Zhu et al. [78] provide an overview of IRL applied to autonomous driving. Except for performing planning and RL, the learned reward function can also be used to predict the behavior of other traffic participants. IRL for predictions was for example used by Ziebart et al. for pedestrians [79], by Sun et al. for human drivers in intersections [61], and by Sadigh for highway driving situations [54].

Chapter 3

Technical background

This chapter provides a brief introduction to Markov decision processes and reinforcement learning. The purpose of the chapter is to summarize the most important concepts and introduce the notation that are used in the subsequent chapters. A comprehensive overview of MDPs and RL is given in the books by Kochenderfer [34] and Sutton and Barto [63], upon which this chapter is based.

3.1 Markov decision processes

Sequential decision-making problems in stochastic environments are commonly modeled as MDPs. Importantly, an MDP satisfies the Markov property, which requires that the probability distribution of the next state only depends on the current state and the action taken by the agent, i.e., not on the history of previous states or actions. An MDP is formally defined as the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, which is described in the following list [34, Ch. 4]:

- The state space $\mathcal{S}$ represents the set of all possible states of the environment. This set could consist of both discrete and continuous states.

- The action space $\mathcal{A}$ represents the set of all valid actions the agent can take. This set could also consist of both discrete and continuous actions. However, since this thesis focuses on high-level decision-making, only discrete actions are considered here.

- The state transition model $T(s'|s,a)$ describes the probability that the system transitions to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ when action $a \in \mathcal{A}$ is taken.
- The reward function $R(s,a)$ returns a scalar reward r for each state-action pair.
- The discount factor $\gamma \in [0,1)$ is a scalar that discounts the value of future rewards. For a finite horizon MDP, γ could also take the value 1.

A policy π is a mapping from a state to an action, which could either be deterministic $a = \pi(s)$ or probabilistic $a \sim \pi(a|s)$. The value of being in a state while following a policy is described by the value function

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) | s_0 = s, \pi\right]. \tag{3.1}$$

The goal of the agent is to find a policy which maximizes the value of each state.

In many decision-making problems, the agent does not have direct access to the state of the environment. Such a problem is commonly modeled as a partially observable Markov decision process, which is an extension to the MDP framework that also models state uncertainty. A POMDP is formally defined as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, \gamma)$, where the state space, action space, transition model, reward function, and discount factor are defined as for an MDP. A POMDP has two additional components [34, Ch. 6]:

- The observation space $\mathcal{O}$, which is the set of possible observations.
- The observation model $O(o|s',a)$, which describes the probability of observing $o \in \mathcal{O}$ in state s' after action a has been taken.

Since the agent does not have direct access to the current state in a POMDP, the agent needs to reason about the history of observations and actions. This history is often merged in a belief state b, which represents a probability distribution over the state space. In this case, the policy is a mapping from beliefs to actions $\pi(b)$.

For many real-world problems, it is not possible to represent the probability distributions T or O explicitly. For some solution approaches, only samples are needed, and then it is sufficient to define a generative model

G that samples a new state or observation from a given state and action, i.e., $s' \sim G(s, a)$ for the MDP case [34, Ch. 4] and $(s', o) \sim G(s, a)$ for the POMDP case [34, Ch. 6].

3.2 Reinforcement learning

If all the elements $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ of an MDP are known, an agent can use this model to directly compute an optimal policy. Such a problem is often considered a planning problem. For small MDPs, dynamic programming[1] techniques can provide an exact solution, which is calculated offline, i.e., before the agent is deployed in the environment. For example, in value iteration [34, Ch. 4], the Bellman operator is iteratively applied to the value function for all states,

$$V_{n+1}(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} T(s'|, a, s) V_n(s') \right]. \tag{3.2}$$

As n goes to infinity, V_n converges to the unique optimal value function V^*, and an optimal policy (not necessarily unique) is extracted by

$$\pi(s) = \arg\max_a \left[R(s, a) + \gamma \sum_{s'} T(s'|, a, s) V^*(s') \right]. \tag{3.3}$$

However, for many real-world problems with high dimensional state spaces, it is intractable to compute and store a policy offline. Contrarily to offline methods, online search methods perform planning from the current state up to some horizon, when the agent has been deployed. Thereby, the agent can limit the computation to states that are reachable from the current state, which is often significantly smaller than the full state space [34, Ch. 4].

In many problems, the state transition probabilities or the reward function are not known. These problems can be solved by reinforcement learning techniques, in which the agent learns how to behave from interacting with the environment [34, Ch. 5], see Figure 3.1. Compared to supervised learning, reinforcement learning presents some additional challenges. Since the data that are available to an RL-agent depends on its current policy, the agent

[1] Dynamic programming refers to simplifying a complex problem by breaking it down into smaller sub-problems, often in a recursive manner.

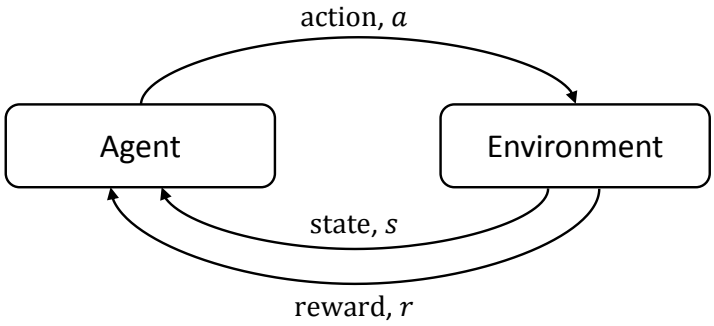


Figure 3.1: *In a reinforcement learning problem, an agent learns a policy π by interacting with its environment. The agent collects experience by repeatedly taking actions and then observing the resulting state and reward.*

must balance exploring the environment and exploiting the knowledge it has already gained. Furthermore, a reward that the agent receives may depend on a crucial decision that was taken earlier in time, which makes it important to assign rewards to the correct decisions.

RL algorithms can be divided into model-based and model-free approaches [34, Ch. 5]. In the model-based versions, the agent first tries to estimate a representation of the state transition function T and then use a planning algorithm to find a policy. On the contrary, as the name suggests, model-free RL algorithms do not explicitly construct a model of the environment to decide which actions to take. The model-free approaches can be further divided into value-based and policy-based techniques. Value-based algorithms, such as Q-learning, aim to learn the value of each state and thereby implicitly define a policy. Policy-based techniques instead search for the optimal policy directly in the policy space, either by policy gradient methods or gradient-free methods, such as evolutionary optimization. There are also hybrid techniques that are both policy and value-based, such as actor critic methods.

RL algorithms generally assume that the environment is modeled as an MDP, i.e., that the state of the environment is known by the agent. However, in many cases of interest, only partial information about the state of the environment is available, which is modeled in the POMDP framework. For such cases, it is common to approximate the state by either the observation or a finite history of observations [63, Ch. 17]. The latter is referred to as a k-Markov approximation, where k defines the length of the included history. For a sufficiently long history, the Markov property is assumed to approximately hold, even though the environment is partially observable.

Chapter 4

Model-free RL approaches

RQ 1: How can a general decision-making agent for autonomous driving be created through RL?

Returning to the analogy of constructing a rocket to reach the moon, this chapter provides the first sketches of the outline of this rocket by adapting two existing RL methods to autonomous driving and investigating their performance. To provide a broad comparison, a policy-based and a value-based RL method are chosen. Q-learning has proven successful in tabular settings [75], and with the renaissance of neural networks during the last decade, the DQN algorithm extended the concept of Q-learning to high-dimensional domains [42]. The DQN algorithm also showed a super-human performance in a diverse set of Atari games. Due to this flexibility and high performance, the DQN method is a natural candidate to consider for the autonomous driving domain.

Genetic algorithms belong to a family of optimization methods that are inspired by the evolutionary mechanisms of natural selection [24]. In general, GAs are suitable for solving optimization problems where the objective function is non-differentiable, or even when an explicit mathematical model does not exist and only a simulation is available. Due to its flexibility, a GA is here used as a policy-based RL method, where the GA optimizes a structure of rules and actions to form a general decision-making agent.

This chapter presents how a DQN and GA agent, respectively, can be constructed to learn how to make tactical decisions in autonomous driving by interacting with a simulated environment. Both methods provide a general

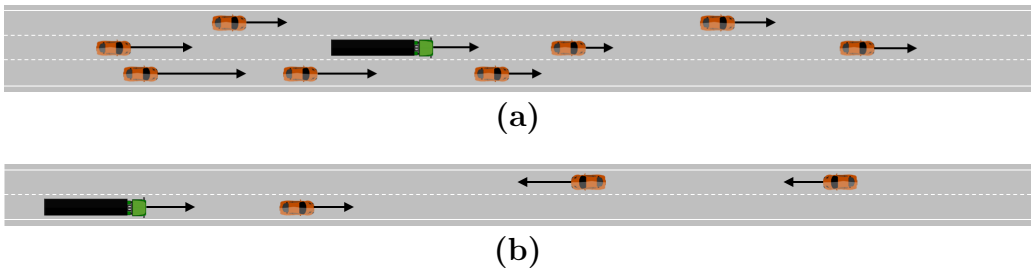


Figure 4.1: *This figure shows the two test scenarios that are used to evaluate the GA and DQN agents. Panel (a) displays an example of an initial traffic situation for the one-way highway driving scenario, whereas panel (b) shows an example of a traffic situation for the overtaking scenario with oncoming traffic, displayed* 10 *seconds from the initial state. The ego vehicle, which consists of a truck-trailer combination, is shown in green and black. The arrows represent the velocities of the vehicles.*

approach, which with few modifications could be applied to any type of driving scenario.

4.1 Simulated driving scenarios

The DQN and GA agents are applied to and evaluated in different highway driving scenarios in this study. The main scenario consists of continuous driving on a one-way highway with three lanes (Figure 4.1a). To illustrate the generality of these approaches, the agents are also tested on an overtaking scenario with oncoming traffic (Figure 4.1b).

In all the test cases, the surrounding vehicles are randomly generated, with random initial positions, velocities, and intentions. To create interesting traffic situations, the vehicles that are initialized behind the ego vehicle are driving faster than the ego vehicle, and the vehicles that start in front of the ego vehicle are driving slower. The drivers of the surrounding vehicles are modeled by using the Intelligent Driver Model [67] and Minimizing Overall Braking Induced by Lane changes model [31]. The IDM is a type of adaptive cruise control (ACC) model, which keeps a desired speed when there is no vehicle in front of the ego vehicle, and otherwise maintains a gap to the preceding vehicle. The MOBIL model is a lane changing model, which makes lane changes with the aim of maximizing the acceleration of all the vehicles that are involved in the traffic situation. A politeness factor

controls the balance between the gains and losses of the ego vehicle and the surrounding vehicles. The combination of the IDM and MOBIL model is also used as a baseline method when evaluating the performance of the different RL agents. Further details on exactly how the simulation environment of the test scenarios is set up and how the episodes are initialized are given in Paper I and II.

4.2 Value-based RL, DQN agent

Paper II introduces a model-free, value-based, RL approach to tactical decision-making, based on the DQN algorithm, which is evaluated in two different highway scenarios (Figure 4.1). This section outlines the method and the main results, whereas further details are given in Paper II.

4.2.1 Approach

Q-learning [75] is a model-free and value-based branch of reinforcement learning, where the objective of the agent is to learn the optimal state-action value function $Q^*(s,a)$. This function is defined as the expected return when taking action a from state s and then following the optimal policy π^*, i.e.,

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k,a_k) | s_0 = s, a_0 = a, \pi\right]. \tag{4.1}$$

The optimal state-action value function follows the Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim T(s'|s,a)}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a')\right], \tag{4.2}$$

which recursively defines the Q-values of the state-action pairs (s,a). The equation can intuitively be understood by the fact that if Q^* is known, the optimal policy is to select the action a' that maximizes $Q^*(s',a')$.

In the DQN algorithm, a neural network with weights θ is used as a function approximator of the optimal state-action value function, $Q(s,a;\theta) \approx Q^*(s,a)$ [42]. The weights of the network are adjusted to minimize the temporal difference (TD) error in the Bellman equation, typically with some kind of stochastic gradient descent (SGD) algorithm. Mini-batches with size M

of experiences, $e = (s, a, r, s')$, are drawn from an experience replay memory, and the loss is calculated as

$$L(\theta) = \mathbb{E}_{\mathrm{M}}\Big[(r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta))^2\Big]. \tag{4.3}$$

Here, θ^{-} represents the neural network parameters of a target network, which is kept fixed for a number of training steps, in order to stabilize the training process. Several of improvements to this vanilla form of DQN have been proposed and are compared by Hessel et al. [20]. See Paper II for the details of the DQN implementation in this study.

Two different agents, which are based on different MDP formulations, are compared in this work. Hereafter, they are referred to as Agent1 and Agent2. Both agents use the same state description, which consists of the ego vehicle state, a description of the lanes of the road, and the relative position and speed of the up to $N_{\max}$ surrounding vehicles. There are m_{ego} states that describe the ego vehicle and the road, and m_{veh} states that describe each of the surrounding vehicles. Agent1 only controls the lane change decisions and the longitudinal control is handled by the IDM, whereas Agent2 controls both the speed and the lane changes. An overview of the available actions is given in Table 4.1. Both agents take decisions every $\Delta t = 1$ s. A positive reward, proportional to the speed, is given for every time step, and a negative reward is given for collisions or driving off the road. Additionally, to limit the number of lane changes, a small negative reward is given when choosing the lane changing action.

A natural assumption is that the policy of a trained agent should be invariant to permutations of the order in which the surrounding traffic participants are listed. The agent should also be able to handle a varying number of traffic participants. A common way to address these requirements is to use an occupancy grid, i.e., a discretized top view of the traffic scene with features that contain presence information [26]. However, the quantisation step of this approach results in a trade-off between accuracy and dimensionality. To fulfill the requirement of permutation invariance without limiting the accuracy, this thesis instead proposes a new way of using a one-dimensional convolutional neural network (CNN) [38] together with a maxpooling layer, applied to the part of the input that describes interchangeable objects, see Figure 4.2. The input that describes the surrounding vehicles is passed through convolutional layers, which are designed to keep identical weights for the inputs of each vehicle, and finally a max pooling layer makes the output independent

Table 4.1: *Action spaces of the two DQN agents.*

	Agent1
a_1	Stay in current lane
a_2	Change lanes to the left
a_3	Change lanes to the right
	Agent2
a_1	Stay in current lane, keep current speed
a_2	Stay in current lane, accelerate with –2 $\mathrm{m/s^2}$
a_3	Stay in current lane, accelerate with –9 $\mathrm{m/s^2}$
a_4	Stay in current lane, accelerate with 2 $\mathrm{m/s^2}$
a_5	Change lanes to the left, keep current speed
a_6	Change lanes to the right, keep current speed

on the ordering of the vehicles. This permutation invariant architecture reduces the input space that the RL agent needs to explore by a factor of $\frac{1}{N_{\max}!}$, compared to an architecture where the ordering matters. Furthermore, the problem of specifying a small fixed input vector size is mitigated. The input vector can instead be made larger than necessary and padded with dummy values for the extra slots, since this additional input will be filtered out by the max pooling layer. Further details on this permutation invariant CNN architecture are explained in Paper II. For comparison, Agent1 and Agent2 are trained with both the CNN architecture and a fully connected neural network (FCNN) architecture.

4.2.2 Results and discussion

Five training runs with different random seeds were carried out for the two agent variants and the two network architectures, here called $\mathrm{Agent1_{FCNN}}$, $\mathrm{Agent1_{CNN}}$, $\mathrm{Agent2_{FCNN}}$, and $\mathrm{Agent2_{CNN}}$. The agents were trained for two million training steps, where an experience is added to the experience replay memory and the neural network weights are updated at every step. At every 50,000 training steps, the agents are evaluated on 1,000 random episodes, which are not present during the training.

Figure 4.3 shows the average proportion of successfully completed episodes for the four agent variants during the training process, in the one-

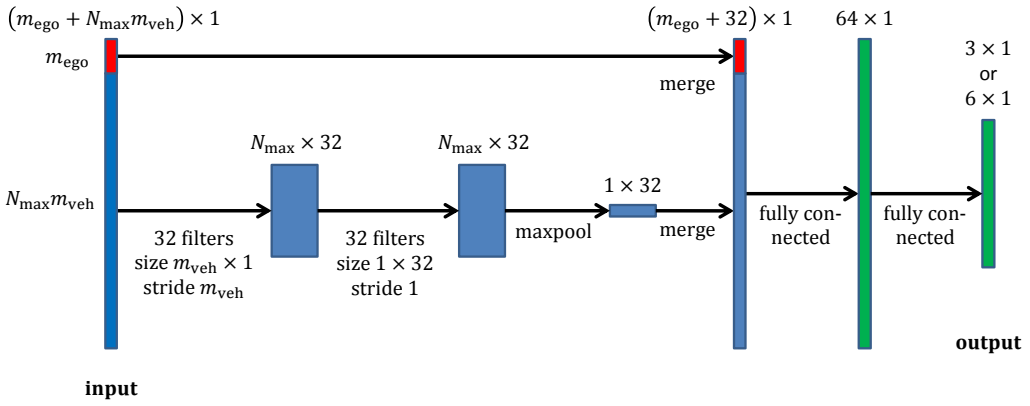


Figure 4.2: *The proposed neural network architecture, which uses convolutional layers and max pooling to make the output invariant to permutations of the vehicles in the current traffic situation. The output consists of three and six Q-values for Agent1 and Agent2, respectively, see the main text for further explanations.*

way highway driving scenario. In Figure 4.4, the performance of the agents is compared to the baseline method through a performance index $\tilde{p}$, defined as

$$\tilde{p} = (d/d_{\max})(\bar{v}/\bar{v}_{\text{ref}}), \tag{4.4}$$

where d is the distance driven by the ego vehicle, which is limited by a collision or the episode length $d_{\max} = 800$ m. The average speed of the ego vehicle is denoted $\bar{v}$, and $\bar{v}_{\text{ref}}$ is the average speed of the ego vehicle when it is controlled by the baseline IDM/MOBIL model.

Agent1$_{\text{CNN}}$ quickly learns to solve all the episodes without any collisions (Figure 4.3) by always staying in its lane. Such a behavior leads to that the ego vehicle often gets blocked by slower vehicles and therefore gives a performance index below 1 (Figure 4.4). However, after around 600,000 training steps, Agent1$_{\text{CNN}}$ learns to perform lane changes to overtake slow vehicles and performs similar to the IDM/MOBIL model.

Agent2$_{\text{CNN}}$ learns how to control the speed and make lane changes without collisions in all of the episodes after around 400,000 training steps. At this point, its performance index is on par with the IDM/MOBIL model. With more training, the agent learns even better strategies and after 1,000,000 training steps, which corresponds to around 300 hours of simulated driving, the performance index stabilizes at 1.1, i.e., Agent2$_{\text{CNN}}$ navigates through

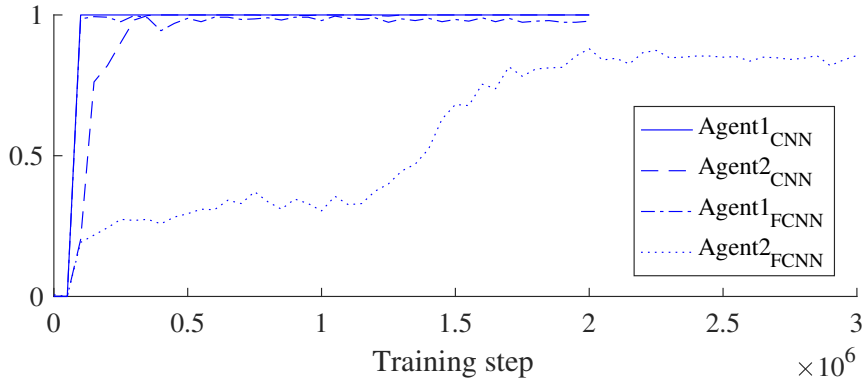


Figure 4.3: *Proportion of episodes that are solved without collisions by the different agents during the training process.*

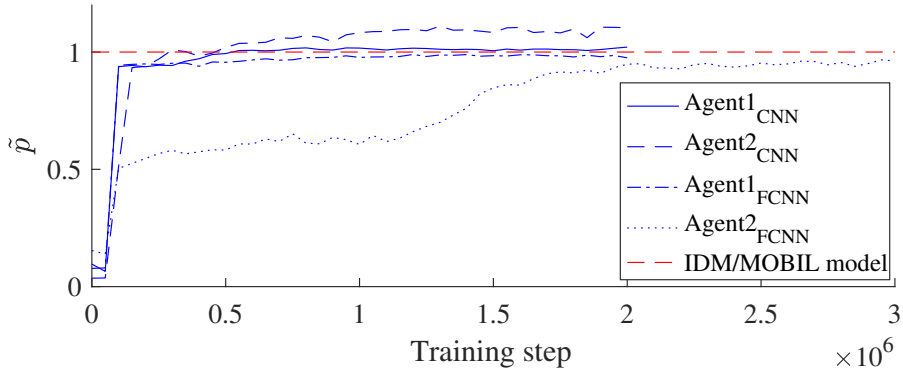


Figure 4.4: *Performance index of the different agents during the training process.*

traffic around 10% faster than the baseline model. These results confirm the intuition that an agent can learn more efficient strategies when controlling both the longitudinal and lateral motions simultaneously.

The agents with the fully connected neural network, $\text{Agent1}_{\text{FCNN}}$ and $\text{Agent2}_{\text{FCNN}}$, perform worse than the CNN agents, both in aspects of sample efficiency, proportion of collision free episodes, and in performance index. This difference in performance illustrates the practical usefulness of applying a permutation invariant neural network structure, compared to a standard fully connected network structure. Table 4.2 sums up the results for the four

Table 4.2: *Performance of the DQN agents for the two test scenarios.*

	One-way highway scenario		Overtaking scenario	
	Collision free episodes	Performance index, $\tilde{p}$	Collision free episodes	Performance index, $\tilde{p}_o$
$\text{Agent1}_{\text{CNN}}$	100%	1.01	100%	1.06
$\text{Agent2}_{\text{CNN}}$	100%	1.10	100%	1.11
$\text{Agent1}_{\text{FCNN}}$	98%	0.98	-	-
$\text{Agent2}_{\text{FCNN}}$	86%	0.96	-	-

agent variants. It also shows that the CNN agents solve all of the test episodes in the overtaking scenario, with a better performance than the IDM/MOBIL model.

4.3 Policy-based RL, GA agent

Paper I introduces a model-free, policy based, RL approach to tactical decision-making, where a genetic algorithm is used to optimize a structure of rules and actions, and their parameters. The method and the main results are outlined in this section, whereas further details are given in Paper I.

4.3.1 Approach

A GA with length-varying chromosomes is used to train a rule-based driver model for the two highway scenarios that are introduced in Section 4.1. A chromosome encodes a set of instructions, which are represented by four genes, $g_1, \ldots, g_4$, described in Table 4.3. Each instruction can encode either a rule or an action. For example, the instruction $[0, 1, 0.2, 0.7]$ would be translated to the rule: *If there is a vehicle in the left lane, in the interval* –60 *m to* 40 *m longitudinally, relative to the ego vehicle, then ...* A chromosome is constructed from a variable set of instructions, which generates a driver

Table 4.3: *Encoding of instructions.*

Gene	Value	Interpretation
g_1	0	Rule: If there is a vehicle in lane g_2, in the interval g_3 to g_4 longitudinally, relative to the ego vehicle
	1	Rule: If there is no vehicle in lane g_2, in the interval g_3 to g_4 longitudinally, relative to the ego vehicle
	2	Action: Change to relative lane g_2, brake or accelerate according to g_3, using pedal level g_4
g_2	-1	Right lane
	0	Current lane
	1	Left lane
g_3	$v_3 \in [0, 1]$	if $g_1 = 0$ or 1, map value v_3 to $[-100, 100]$ m if $g_1 = 2$, g_3 represents braking if $v_3 < 0.5$ and acceleration if $v_3 \geq 0.5$
g_4	$v_4 \in [0, 1]$	if $g_1 = 0$ or 1, map value v_4 to $[-100, 100]$ m if $g_1 = 2$, g_4 represents pedal level

model on the following form:

- *Rule 1*
  - *Rule 2*
    - ▷ *Action 1*
- *Rule 3*
  - ▷ *Action 2*
- ▷ *Action 3*

When the driver model that is generated by the chromosome is to make a decision, it first considers the rules that precedes the first action. If all of them are fulfilled, the first action is executed. If one rule is not fulfilled, the rules that precedes the next action are considered, and so on. If there are no rules associated with an action, as for Action 3 in the example above, this action will be executed if no preceding action is chosen. Decisions are taken at an interval of $\Delta t_{\text{GA}} = 0.1$ s.

In a GA setting, the quality of an individual in a generation is referred to as fitness, which in this case corresponds to the sum of rewards of an

episode. The fitness of an evolved driver model is evaluated in a simulated environment. In short, the agent gets a score of 1 if it completed a 500 m long episode without collisions and at least as fast as the IDM/MOBIL model, and less if it collides or drives slower than the IDM/MOBIL model. If the agent solves an episode without collisions, it is presented with a new one, up to 500 different episodes. The total fitness of the driver model is defined as the sum of the individual episode scores.

In each GA run, the chromosomes are initialized randomly and the fitness of each individual in the generation are evaluated. Tournament selection with a non-homologous two-point crossover operator is used, which allows the length of the chromosomes to vary over generations. Mutations of the parameters and inserting or deleting instructions further diversifies the population. Finally, to guarantee that the performance does not degrade over the generations, elitism is used, which means that the best individual of each generation is copied to the next generation without modification.

4.3.2 Results and discussion

Three optimization runs were carried out for the one-way highway driving scenario, with different random seeds of the GA. The fitness of the best individual of each generation is shown in Figure 4.5. After 1,500 generations, which corresponds to around 100,000 hours of driving, all of the three GA runs solve all the 500 test episodes. The final driver models were then applied to 500 new test episodes, which are different from the ones the agents saw during training, and solved all of them without collisions.

The final decoded driver model from one of the GA runs is shown in Table 4.4. In principle, the evolved driver model generates a behavior where the vehicle stays in its lane and accelerates if no vehicle is close in front of it. If a vehicle is present, but there is no vehicle in the left lane, it changes lanes to the left lane. If also the right lane is occupied, it stays in its own lane and brakes. Otherwise, it changes to the right lane. This behavior resembles a gap acceptance model [1].

The same method was applied to the overtaking scenario, with the only difference being a slightly modified fitness function. All test episodes, and an additional 500 unseen episodes, were solved for this scenario too, but it required around four times as many generations.

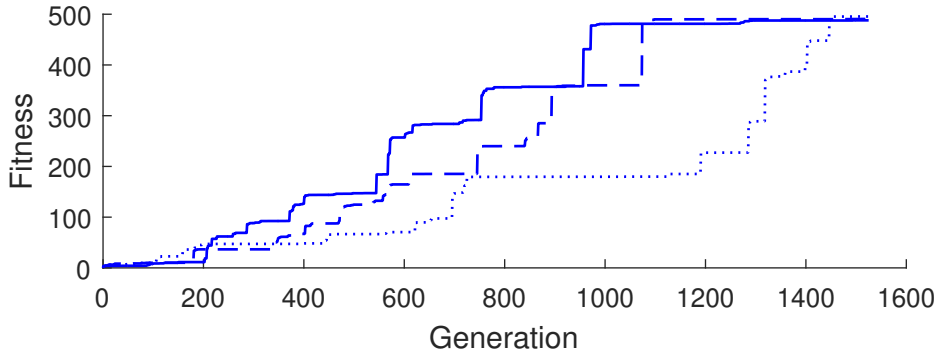


Figure 4.5: *Fitness variation of the best individual in the population for three optimization runs with different random seeds.*

Table 4.4: *Evolved driver model.*

- If no vehicle in ego lane is within $[-3.4, 21.5]$ m
  - If no vehicle in right lane is within $[63.3, 99.7]$ m
    - ▷ Keep lane, accelerate with pedal level 0.94
- If no vehicle in left lane is within $[-17.8, 77.2]$ m
  - ▷ Do lane change to the left, accelerate with pedal level 0.97
- If no vehicle in ego lane is within $[-2.2, 38.1]$ m
  - ▷ Keep lane, accelerate with pedal level 1.00
- If vehicle in right lane is within $[-18.1, 40.9]$ m
  - ▷ Keep lane, brake with pedal level 0.88

▷ Do lane change to the right, accelerate with pedal level 0.78

- If vehicle in left lane is within $[-75.7, 46.6]$ m
  - ▷ Keep lane, brake with pedal level 0.88

▷ Do lane change to the left, accelerate with pedal level 0.86

Chapter 5

Combining planning and RL

RQ 2: In what way can domain knowledge be incorporated into an RL agent?

The methods presented in Chapter 4 can be used to create general decision-making agents for autonomous driving. However, both methods require extensive training to learn a suitable policy. In practice, a lot of domain knowledge is available for autonomous driving, often in the form of models of the driving scenarios. These models can be used to perform a search for the best sequence of actions. Unfortunately, the curse of dimensionality makes an exhaustive search infeasible. Sampling bases search methods, such as MCTS [9], reduce the search complexity and were applied to autonomous driving by Sunberg et al. [62]. However, MCTS still requires a massive amount of computation to search deep into the tree and thereby perform planning over a long time horizon. Furthermore, such a purely model-based search algorithm cannot improve its behavior by learning from data.

To mitigate the mentioned problems, Paper III introduces a general framework for tactical decision-making, which combines the concepts of planning and learning in the form of MCTS and RL. This framework is based on the AlphaGo Zero algorithm [58], which is first extended to a domain with a continuous state space, a not directly observable state, and where self-play cannot be used, and then applied to two different highway driving scenarios (Figure 5.1). This chapter outlines the method and the main results, whereas further details are presented in Paper III.

5.1 Approach

The decision-making framework of this study uses a neural network f_θ, with parameters θ, to improve the MCTS by guiding the search to the most promising parts of the tree. At the same time, the MCTS improves the training process of the neural network by finding long sequences of actions that are necessary in situations that require a long planning horizon. For each state s, the neural network estimates the value $V(s, \theta)$ and a prior probability $\boldsymbol{p}(s, \theta)$ of taking different actions,

$$(\boldsymbol{p}(s, \theta), V(s, \theta)) = f_\theta(s). \tag{5.1}$$

If $P(s, a, \theta)$ represents the prior probability of taking action a, then $\boldsymbol{p}(s, \theta) = (P(s, a_1, \theta), \ldots, P(s, a_{m_{\text{act}}}, \theta))$, for the m_{act} possible actions.

The SelectAction function of Algorithm 1 is used to decide which action to take from a given state s_0. Through n iterations, the function builds a search tree, in which the state-action nodes store the set $\{N(s, a), Q(s, a), C(s, a)\}$, where $N(s, a)$ is the number of node visits, $Q(s, a)$ is the estimated state-action value, and $C(s, a)$ contains the set of child nodes. When traversing the tree, the algorithm chooses to expand the action that maximizes the UCB condition

$$UCB(s, a, \theta) = \frac{Q(s, a)}{Q_{\max}} + c_{\text{puct}} P(s, a, \theta) \frac{\sqrt{\sum_b N(s, b) + 1}}{N(s, a) + 1}, \tag{5.2}$$

where c_{puct} is a parameter that controls the exploration, and $Q_{\max}$ is a normalization parameter.

A progressive widening criterion limits the growth of new state nodes by sampling an old state note if $|C(s, a)| > kN(s, a)^\alpha$, where k and α are parameters that control the width of the search tree. If $|C(s, a)| \leq kN(s, a)^\alpha$, a new state s' is sampled from a generative model $G(s, a)$ of the environment. The new state and reward are added to the set of child nodes and the value of this node is estimated by the neural network as $V(s, \theta)$. Finally, at the end of each iteration, the visit count $N(s, a)$ and Q-values $Q(s, a)$ are updated by a backwards pass through the tree.

When the tree search it completed, after n iterations, an action is sampled proportionally to the visit count of the action nodes of the root node

$$\pi(a \mid s) = \frac{N(s, a)^{1/\tau}}{\sum_b N(s, b)^{1/\tau}}, \tag{5.3}$$

Algorithm 1 Monte Carlo tree search, guided by a neural network policy and value estimate.

function SELECTACTION(s_0, n, θ)
  for $i \in 1 : n$
    SIMULATE(s_0, θ)
  $\pi(a \mid s) \leftarrow \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}$
  $a \leftarrow$ sample from π
  return a, π
function SIMULATE(s, θ)
  if s is terminal
    return 0
  $a \leftarrow \arg\max_a \left(\frac{Q(s,a)}{Q_{\max}} + c_{\text{puct}} P(s,a,\theta) \frac{\sqrt{\sum_b N(s,b)+1}}{N(s,a)+1} \right)$
  if $|C(s,a)| \leq kN(s,a)^{\alpha}$
    $s' \sim G(s,a)$
    $r \leftarrow R(s,a,s')$
    $C(s,a) \leftarrow C(s,a) \cup \{(s', r)\}$
    $v \leftarrow \begin{cases} 0, & \text{if } s' \text{ is terminal} \\ V(s', \theta), & \text{otherwise} \end{cases}$
    $q \leftarrow r + \gamma v$
  else
    $(s', r) \leftarrow$ sample uniformly from $C(s,a)$
    $q \leftarrow r + \gamma$SIMULATE(s', θ)
  $N(s,a) \leftarrow N(s,a) + 1$
  $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q

where τ is a parameter that controls the exploration. During evaluation, the most visited action is greedily chosen, which corresponds to $\tau \to 0$.

Training data are generated from a simulated environment. When an episode ends, after N_s steps, the target values z_i for each step $i = 0, ..., N_s - 1$ are calculated as the received discounted return, according to

$$z_i = \sum_{k=i}^{N_s - 1} \gamma^{k-i} r_k + \gamma^{N_s - i} v_{\text{end}}, \tag{5.4}$$

where $v_{\text{end}} = 0$ if s_{N_s} is a terminal state, and otherwise $v_{\text{end}} = V(s_{N_s}, \theta)$. The target action distribution is given by the tree search as $\boldsymbol{\pi}_i = (\pi(a_1 \mid s_i), \ldots, \pi(m_{\text{act}} \mid s_i))$. The tuples $(s_i, \boldsymbol{\pi}_i, z_i)$ are added to a memory, and then the neural network is trained on the loss function

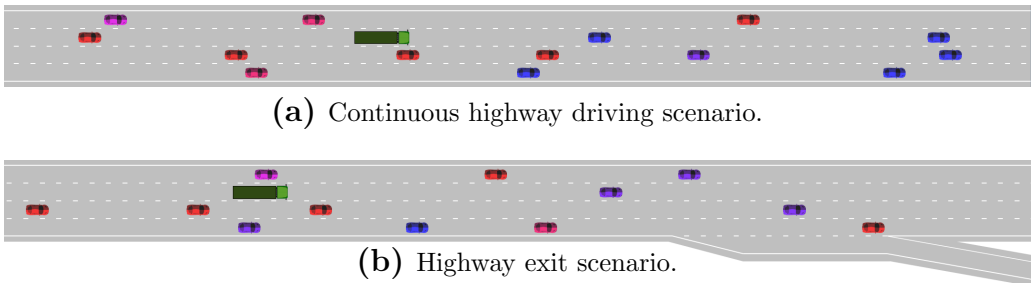
$$\ell = c_1(z - V(s, \theta))^2 - c_2 \boldsymbol{\pi}^\top \log \boldsymbol{p}(s, \theta) + c_3 \|\theta\|^2, \tag{5.5}$$

which consists of the sum of the mean-squared value error, the cross-entropy loss of the policy, and an L_2 weight regularization term. The parameters c_1, c_2, and c_3 balance the different parts of the loss function. A permutation invariant neural network architecture is used, similar to the architecture of the DQN agent, described in Section 4.2. However, in this case, the network has two output heads, which estimate both the value and the action distribution of the input state.

5.2 Simulated experiments

The presented decision-making framework is tested in different highway driving scenarios, shown in Figure 5.1. The first scenario consists of continuous driving on a one-way highway with four lanes, similar to the main test scenario of the DQN and GA agents (Section 4.1). In the second scenario, the same road and traffic is used, but the ego vehicle starts in the leftmost lane and aims to reach an exit on the right side of the road after 1,000 m, which requires planning over a longer time horizon compared to the continuous driving scenario. The surrounding traffic is controlled by the combination of the IDM and MOBIL model, with randomized driver parameters. More information about the simulated environment is given in Section 4.1 and all the details are given in Paper III.

A state description and reward model that is similar to the DQN agent, described in Section 4.2, is used here. The agent can only observe the physical state of the surrounding vehicles, but not the driver intentions. A particle filter is therefore used to estimate the parameters of the surrounding drivers, which are assumed to behave according to the IDM/MOBIL model. The most likely state is then used as input to Algorithm 1 and to the generative model. The action space consists of high-level actions, which can modify the setpoint of an ACC (based on the IDM) or change lanes. An overview is given in Table 5.1. The action space is also pruned at every time step, such



(a) Continuous highway driving scenario.

(b) Highway exit scenario.

Figure 5.1: *The two test scenarios that are used to evaluate the MCTS/NN agent. Panel (a) shows an initial state for the continuous highway driving scenario, whereas panel (b) shows the exit scenario, when the ego vehicle is approaching the exit on the right side of the road. The ego vehicle is the green truck, whereas the colors of the surrounding vehicles represent the aggressiveness level of their corresponding driver models. Red is an aggressive driver, blue is a timid driver, and the different shades of purple represent levels in between. The exact interpretation of aggressiveness level is given in Paper III.*

Table 5.1: *Action space of the MCTS/NN agent.*

a_1	Stay in current lane, keep current ACC setpoint
a_2	Stay in current lane, decrease ACC setpoint
a_3	Stay in current lane, increase ACC setpoint
a_4	Change lanes to the right, keep current ACC setpoint
a_5	Change lanes to the left, keep current ACC setpoint

that all actions that lead to collisions or driving off the road are removed in the tree search.

The decision-making framework of Section 5.1, hereafter referred to as the MCTS/NN agent, was trained in the two simulated highway environments for 250,000 training steps. An evaluation phase was run at every 20,000 steps, where the agent was tested on 100 random episodes. The performance of the MCTS/NN agent is compared to three baseline methods, consisting of standard MCTS which is not guided by the trained neural network, the combination of the IDM and MOBIL model, and only the IDM, which always stays in its original lane and can therefore be viewed as a minimum performance behavior. Unless stated otherwise, both the MCTS/NN and standard MCTS use $n = 2{,}000$ iterations to build the search tree.

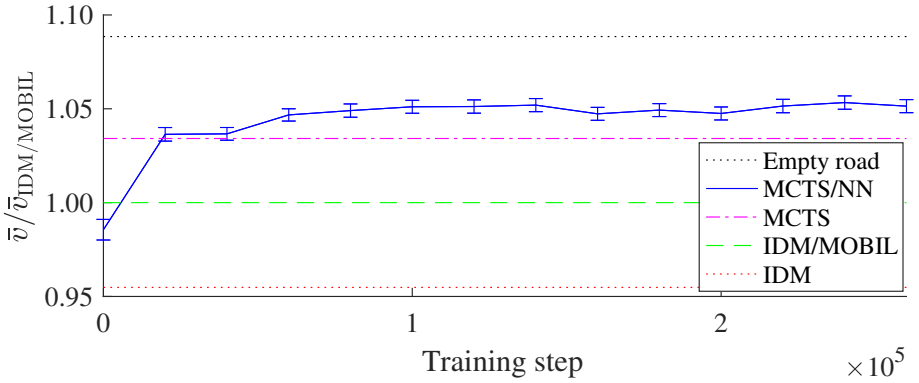


Figure 5.2: *Mean speed $\bar{v}$ during the evaluation episodes for the continuous highway driving scenario, normalized with the mean speed of the IDM/MOBIL agent $\bar{v}_{\mathrm{IDM/MOBIL}}$. The error bars indicate the standard error of the mean for the MCTS/NN agent, i.e., $\sigma_{\mathrm{sample}}/\sqrt{100}$, where σ_{sample} is the standard deviation of the* 100 *evaluation episodes.*

5.3 Results and discussion

The results show that the agents that are obtained by applying the proposed framework to the two different test scenarios outperform the baseline methods. First considering the continuous highway driving scenario, Figure 5.2 shows the average speed $\bar{v}$ of the different agents in the evaluation episodes, normalized with the mean speed of the IDM/MOBIL agent $\bar{v}_{\mathrm{IDM/MOBIL}}$. The IDM, which always maintains its original lane, is naturally slower than the other agents, since it often gets blocked by slower vehicles. Since standard MCTS performs planning, it finds better strategies and can navigate through traffic faster than the IDM/MOBIL agent. The MCTS/NN agent quickly learns to match the performance of the MCTS agent and outperforms it after 60,000 training steps.

The highway exit scenario has a pass or fail outcome and is therefore conceptually different from the continuous highway driving scenario. Figure 5.3 shows the proportion of evaluation episodes where the exit is reached during the training of the MCTS/NN agent. The agent quickly learns how to succeed in most episodes and after 120,000 training steps, which corresponds to around 30 hours of simulated driving, it manages to solve all of them. The standard MCTS agent solves 70% and a modified IDM/MOBIL agent solves 54% of the episodes.

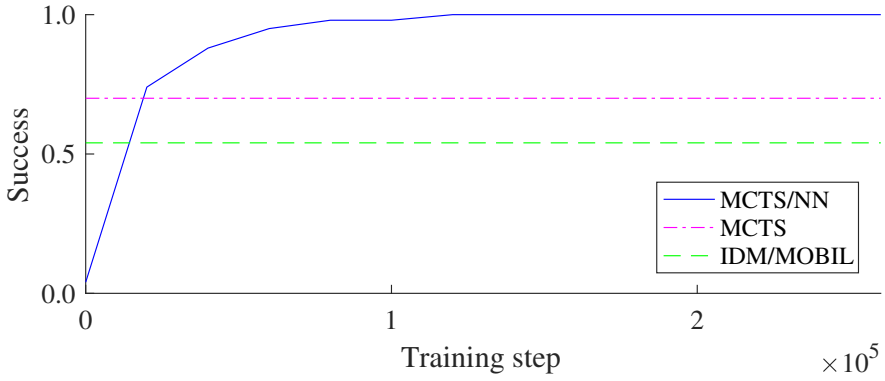
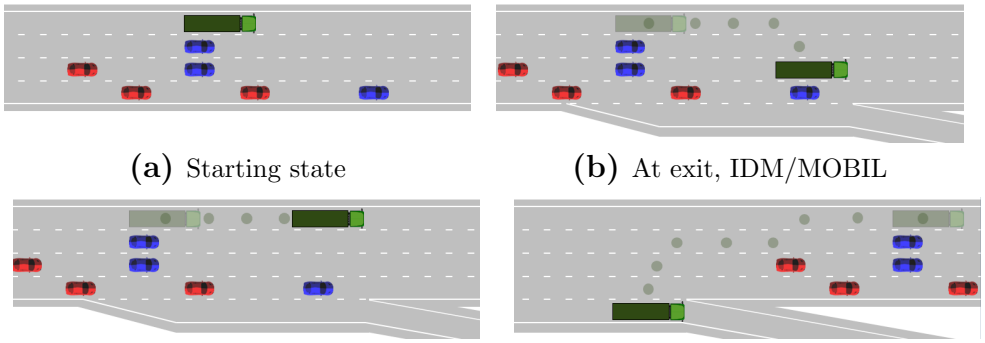


Figure 5.3: *Proportion of successful evaluation episodes, as a function of training steps, for the highway exit scenario.*

A key difference between the compared agents is their planning ability, which is crucial for the outcome in the highway exit scenario. Figure 5.4 shows a simplified situation where it is necessary to plan relatively far into the future. In this example, the ego vehicle starts 300 m from the exit, six other vehicles are placed in the other lanes, and all vehicles start with an initial speed of 21 m/s. The ego vehicle can only reach the exit by first slowing down and then performing multiple lane changes to the right. This strategy is only found by the trained MCTS/NN agents, whereas the standard MCTS agent does not discover that it can reach the exit and therefore remains in its original lane, to avoid receiving negative rewards for changing lanes. The modified IDM/MOBIL agent tries to accelerate and then change lanes to the right, but is blocked by another vehicle and also fails to reach the exit.

The computational load of the MCTS/NN agent is proportional to the number of iterations n that are used to build the search tree. As previously mentioned, $n = 2{,}000$ is used in the presented results. However, the MCTS/NN agent is anytime capable, i.e., it can abort its search after any number of steps, even after just one, which will then return the action given by the neural network. More iterations generally improve the result, up to a limit. The number of necessary iterations depends on how long planning horizon the specific traffic situation requires. More results on how the number of iterations affect the performance are shown and discussed in Paper III.

In contrast to the methods presented in Chapter 4, the decision-making framework of this chapter uses a generative model of the environment. To



(a) Starting state **(b)** At exit, IDM/MOBIL

(c) At exit, MCTS **(d)** At exit, MCTS/NN

Figure 5.4: *Example of when it is necessary to plan relatively far into the future to solve a specific situation. The initial state,* 300 *m from the exit, is shown in (a), and the state at the exit is shown for the three agents in (b), (c), and (d). The dots show the position of the ego vehicle relative to the other vehicles during the maneuver, i.e., in (b) and (c) the ego vehicle accelerates and overtakes the slower vehicles, whereas in (d), the ego vehicle slows down and ends up behind the same vehicles.*

incorporate such domain knowledge in the agent has both advantages and disadvantages regarding, e.g., training time, computational complexity, and generality, which is discussed further in Chapter 7.

Chapter 6

Uncertainty of RL-based agents

RQ 3: How can an RL-based agent provide an uncertainty estimate of its decisions?

Chapter 4 and 5 introduce RL methods for training tactical decision-making agents. As discussed in Chapter 1, an important advantage of learning-based approaches, compared to manually specified systems, is that they could scale to all types of driving situations. However, a drawback of many learning-based approaches is that they provide black-box solutions, which do not indicate how confident the agent is about its decisions, or equivalently, the uncertainty of the decisions.

Uncertainty is commonly divided in epistemic and aleatoric uncertainty [33]. Epistemic uncertainty in a decision-making agent stems from the fact that the agent has not experienced similar situations before, for example, if an agent has been trained for 'normal' driving and then faces an accident or a speeding driver. This type of uncertainty can be reduced by further training of the agent. Contrarily, aleatoric uncertainty arises due to stochasticity in the outcome of an action, for example in a situation where there is an occlusion. Therefore, more training cannot reduce the aleatoric uncertainty. Knowledge of the epistemic uncertainty allows the agent to identify which situations it has not been trained for, which can increase the safety of the decision-making system, and knowledge of the aleatoric uncertainty allows the agent to make risk-aware decisions. Therefore, an estimate of both types of uncertainty is essential for a safety-critical application such as autonomous driving.

Paper IV and V introduce a Bayesian RL approach that provides an estimate of the epistemic uncertainty and a method for classifying a decision as trustworthy or not. The approach is tested in highway and intersection scenarios, respectively. Paper VI extends this work by introducing the Ensemble Quantile Networks method, which provides an estimate of both the aleatoric and the epistemic uncertainty of a trained RL agent. This chapter presents a summary of how both the aleatoric and epistemic uncertainty can be estimated, and displays the main results when the different methods are applied to various highway and intersection scenarios. Implementations of the described methods and simulated experiments are available on GitHub, together with some animated results [22, 23].

6.1 Approach

The following sections describe how the aleatoric and epistemic uncertainty of a trained RL agent can be estimated, which when combined forms the EQN algorithm.

6.1.1 Epistemic uncertainty

Statistical bootstrapping can be used to extend the DQN algorithm by training an ensemble of neural networks on different subsets of the available experiences [48]. The ensemble members then return a distribution over the estimated Q-values. A better Bayesian posterior is obtained by adding different randomized prior functions (RPFs) to each ensemble member [47]. The Q-values of each ensemble member, index by k, is calculated as

$$Q_k(s,a) = f(s,a;\theta_k) + \beta p(s,a;\hat{\theta}_k), \tag{6.1}$$

where f and p are neural networks with identical architecture. The parameters θ_k are updated during the training process, whereas, importantly, the parameters $\hat{\theta}_k$ are fixed at their initial values. A hyperparameter β scales the influence of the fixed prior network. The trainable network is then updated from the TD-error

$$\begin{aligned} L_{\text{RPF}}(\theta_k) = \mathbb{E}_M\Big[&(r_t + \gamma \max_a (f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s_{t+1}, a) \\ &- (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s_t, a_t))^2\Big], \end{aligned} \tag{6.2}$$

Algorithm 2 Ensemble RPF training process

```
for k ← 1 to K
    Initialize θ_k and θ̂_k randomly
    m_k ← {}
t ← 0
while networks not converged
    s_t ← initial random state
    ν ∼ U{1, K}
    while episode not finished
        a_t ← arg max_a Q_ν(s_t, a)
        s_{t+1}, r_t ← StepEnvironment(s_t, a_t)
        for k ← 1 to K
            if p ∼ U(0, 1) < p_add
                m_k ← m_k ∪ {(s_t, a_t, r_t, s_{t+1})}
            M ← sample from m_k
            update θ_k with SGD and loss L_RPF(θ_k)
        t ← t + 1
```

similarly as for the standard DQN method, described in Eq. 4.3.

The training process of the ensemble RPF method is outlined in Algorithm 2. An ensemble of K neural networks is here used, with individual experience replay buffers m_k. For each new episode, a random ensemble member ν is selected and used to take greedy actions throughout the episode, which corresponds to an approximate Thompson sampling approach to the exploration vs. exploration dilemma. Each new experience is then added to the individual replay buffers with probability p_{add}, and the weights are updated with SGD. Finally, during testing, the trained agent follows the policy which maximizes the mean Q-value of the ensemble members.

The ensemble variance of the Q-values gives an estimate of the epistemic uncertainty of the available actions, and a threshold σ_e^2 can be used to classify whether a decision has an acceptable level of uncertainty. There are many uses for an epistemic uncertainty estimate, which is further discussed in Section 6.3. In this thesis, the benefit of this estimate is demonstrated by applying a backup policy $\pi_{\text{backup}}(s)$ if an action has an unacceptable level of

uncertainty, i.e., the trained agent follows the policy

$$\pi_{\sigma_e}(s) = \begin{cases} \arg\max_a \mathbb{E}_k[Q_k(s,a)], & \text{if } \mathrm{Var}_k[Q_k(s,a)] < \sigma_e^2, \\ \pi_{\text{backup}}(s), & \text{otherwise.} \end{cases} \tag{6.3}$$

6.1.2 Aleatoric uncertainty

In contrast to Q-learning, distributional RL aims to learn not only the expected return, but the distribution over returns [5]. This distribution is represented by the random variable

$$Z^\pi(s,a) = \sum_{k=0}^{\infty} \gamma^k R(s_k, a_k), \tag{6.4}$$

given $s_0 = s$, $a_0 = a$, and π, where the mean is the traditional value function, $Q^\pi(s,a) = \mathbb{E}[Z^\pi(s,a)]$. The distribution over returns represents the aleatoric uncertainty of the outcome, which can be used to estimate the risk in different situations and to train an agent in a risk-sensitive manner.

The implicit quantile networks (IQN) approach [12] to distributional RL uses a neural network to implicitly represent the quantile function $F_Z^{-1}(\tau)$ of the random variable Z and then update the weights of the network with quantile regression. For ease of notation, define $Z_\tau := F_Z^{-1}(\tau)$, and note that for $\tau \sim \mathcal{U}(0,1)$ the sample $Z_\tau(s,a) \sim Z(s,a)$. The TD-error for two quantile samples, $\tau, \tau' \sim \mathcal{U}(0,1)$, is

$$\delta_t^{\tau,\tau'} = r_t + \gamma Z_{\tau'}\big(s_{t+1}, \pi^*(s_{t+1}); \theta^-\big) - Z_\tau(s_t, a_t; \theta), \tag{6.5}$$

where $\pi^*(s) = \arg\max_a Q(s,a)$. A sample-based estimate of $\pi^*(s)$ is obtained from K_τ samples of $\tilde{\tau} \sim \mathcal{U}(0,1)$, as

$$\tilde{\pi}(s) = \arg\max_a \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} Z_{\tilde{\tau}_k}(s,a;\theta). \tag{6.6}$$

For a pair of quantiles τ, τ', the quantile Huber regression loss [13], with threshold κ, is calculated as

$$\rho_\kappa(\delta_t^{\tau,\tau'}) = |\tau - \mathbb{I}\{\delta_t^{\tau,\tau'} < 0\}| \frac{\mathcal{L}_\kappa(\delta_t^{\tau,\tau'})}{\kappa}. \tag{6.7}$$

Algorithm 3 IQN training process

Initialize θ randomly
$m \leftarrow \{\}$
$t \leftarrow 0$
while network not converged
    $s_t \leftarrow$ initial random state
    while episode not finished
        if $e \sim \mathcal{U}(0,1) < \epsilon$
            $a_t \leftarrow$ random action
        else
            $\tau_1, \ldots, \tau_{K_\tau} \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0, \alpha)$
            $a_t \leftarrow \arg\max_a \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} Z_{\tau_k}(s_t, a)$
        $s_{t+1}, r_t \leftarrow \textsc{StepEnvironment}(s_t, a_t)$
        $m \leftarrow m \cup \{(s_t, a_t, r_t, s_{t+1})\}$
        $M \leftarrow$ sample from m
        update θ with SGD and loss $L_{\text{IQN}}(\theta)$
        $t \leftarrow t + 1$

Here, $\mathcal{L}_\kappa(\delta_t^{\tau,\tau'})$ is the Huber loss [25], defined as

$$\mathcal{L}_\kappa(\delta_t^{\tau,\tau'}) = \begin{cases} \frac{1}{2}(\delta_t^{\tau,\tau'})^2, & \text{if } |\delta_t^{\tau,\tau'}| \leq \kappa, \\ \kappa(|\delta_t^{\tau,\tau'}| - \frac{1}{2}\kappa), & \text{otherwise,} \end{cases} \tag{6.8}$$

which gives a smooth gradient as $\delta_t^{\tau,\tau'} \to 0$. The full loss function $L_{\text{IQN}}(\theta)$ is obtained from a mini-batch M of sampled experiences, in which the quantiles τ and τ' are sampled N and N' times, respectively, according to

$$L_{\text{IQN}}(\theta) = \mathbb{E}_M \left[\frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_\kappa \left(\delta_t^{\tau_i, \tau_j'} \right) \right]. \tag{6.9}$$

The full training process of the IQN method is outlined in Algorithm 3.

Dabney et al. show that the IQN method can achieve state of the art results on the Atari-57 benchmark and reason about the performance of risk-sensitive training for a few of the Atari games [12]. However, the estimated distribution over returns of the fully trained agent can also be used to quantify the aleatoric uncertainty of a decision. One such uncertainty measure is the variance of the estimated returns for the evenly distributed sample set

$\tau_\sigma = \{i/K_\tau \mid i \in [1, K_\tau]\}$. A threshold σ_{a}^2 can then be defined, such that the agent classifies a decision with a higher variance in returns as uncertain. In this thesis, the benefit of the uncertainty classification is demonstrated by choosing a predefined backup policy $\pi_{\text{backup}}(s)$ if the sample variance is higher than the threshold, i.e., the fully trained agent follows the policy

$$\pi_{\sigma_{\text{a}}}(s) = \begin{cases} \arg\max_a \mathbb{E}_{\tau_\sigma}[Z_\tau(s,a)], & \text{if } \text{Var}_{\tau_\sigma}[Z_\tau(s,a)] < \sigma_{\text{a}}^2, \\ \pi_{\text{backup}}(s), & \text{otherwise.} \end{cases} \tag{6.10}$$

6.1.3 Aleatoric and epistemic uncertainty

In order to obtain a complete uncertainty estimation of both the aleatoric and the epistemic uncertainty, this thesis introduces the Ensemble Quantile Networks algorithm, which combines the properties of the IQN and ensemble RPF methods. An agent that is trained by the EQN method can then take actions that consider both the inherent uncertainty of the outcome and the model uncertainty in each situation.

As the name suggests, the EQN method uses an ensemble of networks, where each ensemble member k individually estimates the distribution over returns as

$$Z_{k,\tau}(s,a) = f_\tau(s,a;\theta_k) + \beta p_\tau(s,a;\hat{\theta}_k). \tag{6.11}$$

Similarly as for the RPF method, f_τ and p_τ are neural networks with identical architecture, θ_k are trainable network parameters, whereas the parameters $\hat{\theta}_k$ are fixed. The TD-error of ensemble member k and two quantile samples, $\tau, \tau' \sim \mathcal{U}(0,1)$, is

$$\delta_{k,t}^{\tau,\tau'} = r_t + \gamma Z_{k,\tau'}(s_{t+1}, \tilde{\pi}_k(s_{t+1})) - Z_{k,\tau}(s_t, a_t), \tag{6.12}$$

where $\tilde{\pi}_k(s) = \arg\max_a \frac{1}{K_\tau} \sum_{j=1}^{K_\tau} Z_{k,\tilde{\tau}_j}(s,a)$ is a sample-based estimate of the optimal policy. Quantile Huber regression is applied to a mini-batch of experiences, which gives the loss function

$$L_{\text{EQN}}(\theta_k) = \mathbb{E}_M \left[\frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_\kappa \left(\delta_{k,t}^{\tau_i,\tau_j'} \right) \right]. \tag{6.13}$$

For each new training episode, the agent follows the policy $\tilde{\pi}_\nu(s)$ of a randomly selected ensemble member ν. The full training process of the EQN agent is outlined in Algorithm 4.

Algorithm 4 EQN training process

for $k \leftarrow 1$ to K
    Initialize θ_k and $\hat{\theta}_k$ randomly
    $m_k \leftarrow \{\}$
$t \leftarrow 0$
while networks not converged
    $s_t \leftarrow$ initial random state
    $\nu \sim \mathcal{U}\{1, K\}$
    while episode not finished
        $\tau_1, \ldots, \tau_{K_\tau} \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0, \alpha)$
        $a_t \leftarrow \arg\max_a \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} Z_{\nu,\tau_k}(s_t, a)$
        $s_{t+1}, r_t \leftarrow \textsc{StepEnvironment}(s_t, a_t)$
        for $k \leftarrow 1$ to K
            if $p \sim \mathcal{U}(0, 1) < p_{\text{add}}$
                $m_k \leftarrow m_k \cup \{(s_t, a_t, r_t, s_{t+1})\}$
            $M \leftarrow$ sample from m_k
            update θ_k with SGD and loss $L_{\text{EQN}}(\theta_k)$
        $t \leftarrow t + 1$

The EQN agent provides an estimate of both the aleatoric and epistemic uncertainty, based on the variance of the returns and the variance of the Q-values. The agent is considered confident about a decision if $\text{Var}_{\tau_\sigma}[\mathbb{E}_k[Z_{k,\tau}(s, a)]] < \sigma_{\text{a}}^2$ and $\text{Var}_k[\mathbb{E}_{\tau_\sigma}[Z_{k,\tau}(s, a)]] < \sigma_{\text{e}}^2$. The trained agent then follows the policy

$$\pi_{\sigma_{\text{a}},\sigma_{\text{e}}}(s) = \begin{cases} \arg\max_a \mathbb{E}_k[\mathbb{E}_{\tau_\sigma}[Z_{k,\tau}(s, a)]], & \text{if confident,} \\ \pi_{\text{backup}}(s), & \text{otherwise.} \end{cases} \tag{6.14}$$

6.2 Simulated experiments

The algorithms for estimating the aleatoric and epistemic uncertainty are tested in different highway and occluded intersection scenarios in this thesis, but could naturally be applied to any type of driving scenario. The highway scenario is similar to the previously described one-way highway scenarios in Section 4.1 and 5.2. Importantly, the training only includes surrounding traffic with a 'normal' highway speed between 15 and 35 m/s.

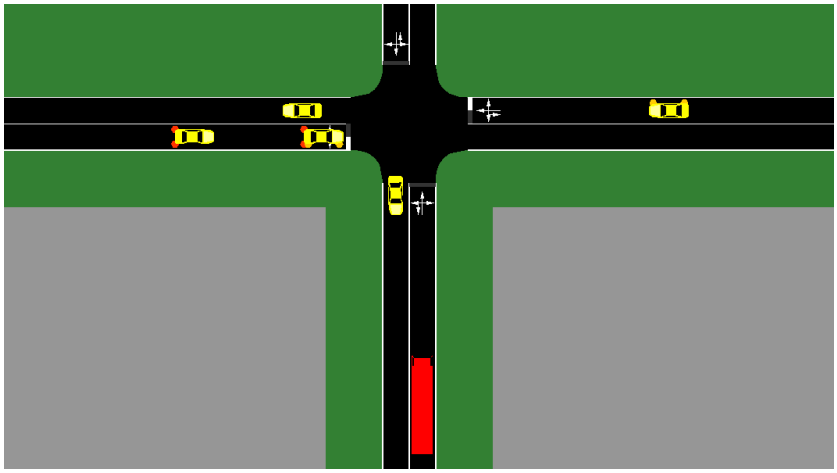


Figure 6.1: *Example of the occluded intersection scenario. The ego vehicle is shown in red, surrounding vehicle in yellow, and the areas that cause occlusions are shown in gray.*

The intersection scenario includes an occlusion, see Figure 6.1, which together with the unknown intentions of the surrounding drivers result in randomness in the outcome from a given state and policy, i.e., an aleatoric uncertainty. The surrounding vehicles are controlled by a modified version of the IDM and during the training process the speed is limited to 15 m/s. The agent observes the position, speed, and heading of the non-occluded vehicles within the sensor range, and can choose between the high-level actions *'stop'*, *'cruise'*, and *'go'*. Slightly simplified, these high-level actions are translated to accelerations through the IDM model by either setting a target that makes the ego vehicle stop at the intersection, maintaining its speed, or accelerating to a set cruising speed. The agent receives a positive reward when the ego vehicle manages to cross the intersection and a negative reward at collisions. If the ego vehicle has not crossed the intersection within $N_{\max} = 100$ steps, the episode is terminated, which is referred to as a timeout.

The backup policy $\pi_{\text{backup}}(s)$ for the highway driving scenario consists of staying in the current lane and full braking. For the intersection scenario, the backup policy selects the action *'stop'* if it is possible to stop before the intersection, considering the maximum braking capability. Otherwise, the backup policy keeps the action that was recommended by the agent. A policy that would always choose the action *'stop'* could end up standing still in the intersection and thereby cause more collisions. In a real-world implementation, naturally more advanced backup policies would be considered [64].

The different agents were trained for three million training steps and evaluated in 1,000 randomly generated test episodes, which were not part of the training process. Each simulation was run with 5 random seeds and the results present the mean, together with the corresponding standard deviation. A permutation invariant neural network architecture is used, similar to the one presented in Section 4.2. For comparison, a DQN agent is also trained for the same scenarios.

6.3 Results and discussion

The results of the experiments show that the IQN method can be used to obtain an estimate of the aleatoric uncertainty and when applying the confidence criterion, situations with high uncertainty are identified, which is used to reduce the number of collisions. The results also illustrate that the ensemble RPF method can provide an estimate of the epistemic uncertainty, which together with the confidence criterion classifies situations as within or without the training distribution. Finally, the results for the EQN method demonstrate that it can estimate both types of uncertainty. This section provides an overview of the main results, whereas more results on different properties of the algorithms, such as risk-sensitive training and the effect of changing hyperparameter values, are available in Paper IV, V, and VI.

An overview of the results for the intersection scenario is shown in Table 6.1. The IQN agent clearly outperforms the standard DQN agent, with a similar crossing time but only half as many collisions. The results also show that the IQN method, combined with the aleatoric uncertainty criterion, can be used to detect situations with a high aleatoric uncertainty. When the bound on the allowed uncertainty is reduced, the number of collisions is reduced while the crossing time increases. Figure 6.2 illustrate this natural trade-off between time efficiency and safety, which is controlled by the parameter σ_{a}.

The RPF agent performs similar to the IQN agent within the training distribution of the intersection scenario, see Table 6.1. The epistemic uncertainty in the situations that are encountered in the test episodes naturally decreases as the training progresses, which is illustrated in Figure 6.3. However, the interesting properties of the RPF agent are illustrated when the agent is exposed to situations that are outside of the training distribution. Figure 6.4 shows the result when the speed of the surrounding vehicles is

Table 6.1: *Intersection scenario, tested within the training distribution*

algorithm	parameter	collisions (%)	crossing time (s)
DQN	-	4.0 ± 0.5	31.7 ± 1.1
IQN	$\sigma_\mathrm{a} = \infty$	1.7 ± 0.3	33.0 ± 1.1
	$\sigma_\mathrm{a} = 4.0$	0.9 ± 0.2	33.5 ± 1.2
	$\sigma_\mathrm{a} = 3.0$	0.5 ± 0.2	34.7 ± 1.3
	$\sigma_\mathrm{a} = 2.0$	0.2 ± 0.1	39.2 ± 1.0
	$\sigma_\mathrm{a} = 1.0$	0.0 ± 0.0	61.2 ± 3.4
RPF	-	1.5 ± 0.3	38.0 ± 1.8
EQN	$\sigma_\mathrm{a} = \infty$	0.9 ± 0.1	32.0 ± 0.2
	$\sigma_\mathrm{a} = 3.0$	0.6 ± 0.2	33.8 ± 0.3
	$\sigma_\mathrm{a} = 2.0$	0.5 ± 0.1	38.4 ± 0.5
	$\sigma_\mathrm{a} = 1.5$	0.3 ± 0.1	47.2 ± 1.2
	$\sigma_\mathrm{a} = 1.0$	0.0 ± 0.0	71.1 ± 1.9
	$\sigma_\mathrm{a} = 1.5$, $\sigma_e = 1.0$	0.0 ± 0.0	48.9 ± 1.6

increased in the intersection scenario. If the allowed epistemic uncertainty is not limited, i.e., setting $\sigma_\mathrm{e} = \infty$, the number of collisions increase significantly. When the threshold is reduced, the number of collisions is reduced to almost zero, at the expense of an increased number of timeouts. Also note that the number of collisions at 15 m/s, which is within the training distribution, is slightly reduced with a tighter bound on the allowed uncertainty. We hypothesize that the explanation for this effect is that some situations that cause collisions are rarely seen during the training process, and therefore the epistemic uncertainty remains high in such situations. A specific example of an out-of-distribution situation is shown for the highway scenario in Figure 6.5, where a vehicle has stopped due to an accident and the agent cannot change lanes due to other surrounding vehicles. As mentioned in Section 6.2, the training was done with speeds between 15 and 35 m/s for the highway scenario. The DQN agent does brake to avoid a collision with the stopped

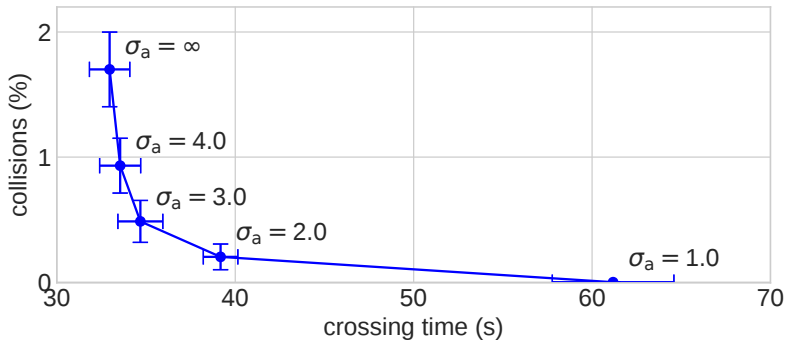


Figure 6.2: *Number of collisions and crossing time for the IQN algorithm for different levels of allowed aleatoric uncertainty, which is achieved by varying the parameter* σ_a.

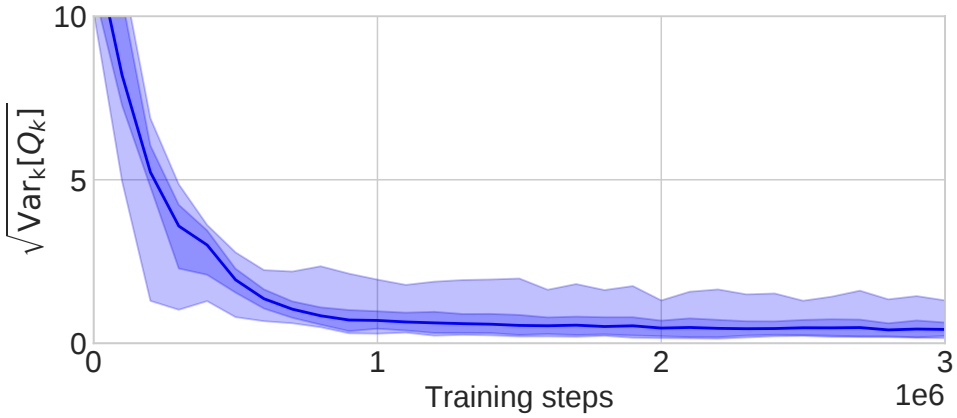
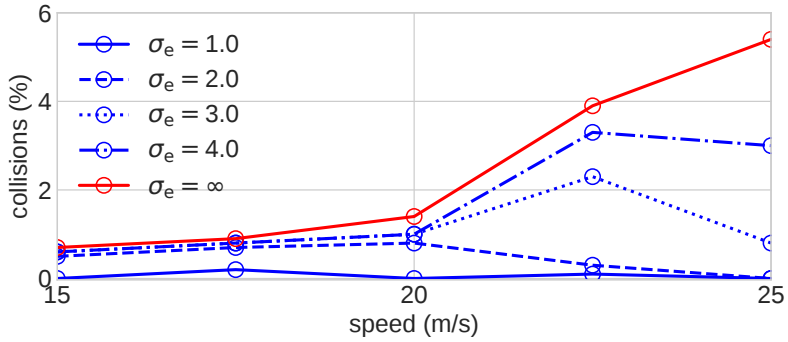


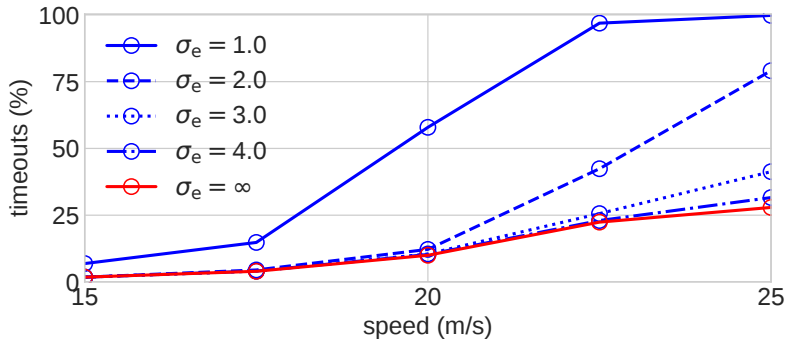
Figure 6.3: *Epistemic uncertainty of the chosen actions for the ensemble RPF agent, with parameters* $\beta = 300$ *and* $K = 10$*, during testing episodes within the training distribution. The solid line shows the mean, while the shaded regions indicate percentile* 10 *to* 90 *and* 1 *to* 99.

vehicle, but too late, which makes a collision unavoidable. However, the RPF agent indicates a high epistemic uncertainty at an early stage and therefore manages to stop in time.

The EQN method can estimate both the aleatoric and epistemic uncertainty simultaneously, and performs similarly to the IQN and RPF methods.



(a) Collisions



(b) Timeouts

Figure 6.4: *Number of collisions and timeouts for the RPF agent ($\beta = 300$, $K = 10$), in situations outside the training distribution. The maximum speed of the crossing vehicles is 15 m/s during the training process, and then the speed is gradually increased in the testing episodes.*

Figure 6.6 shows how the trade-off between the number of collisions and the crossing time within the training distribution is controlled by the parameter σ_a. Outside of the training distribution, a similar trade-off is shown in Figure 6.7 when varying the parameter σ_e. Interestingly, a combination of a moderate limit on both the aleatoric and the epistemic uncertainty removes all the collisions within the training distribution, see Table 6.1. This result illustrates the usefulness of considering the epistemic uncertainty even in situations that are similar to the training scenarios, since rare edge cases can be detected.

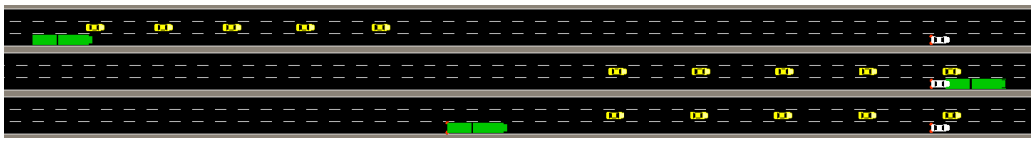


Figure 6.5: *Example of a traffic situation that is outside of the training distribution, where a collision occurs if the confidence of the agent is not considered. The top panel shows the initial state, where the ego vehicle is displayed in green, moving cars in yellow, and a stopped car in white. The two bottom panels show the state for the DQN and ensemble RPF agents after* 12 *s.*

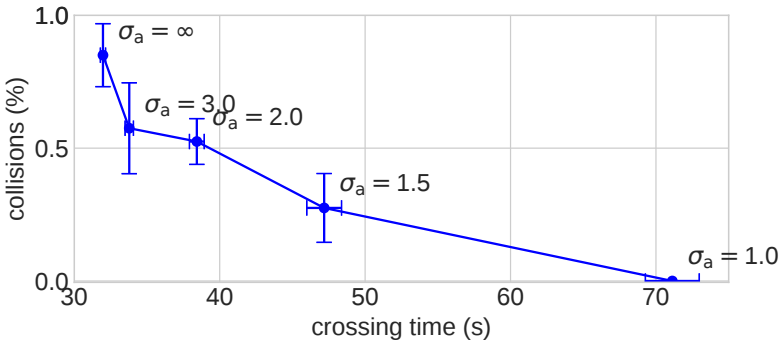
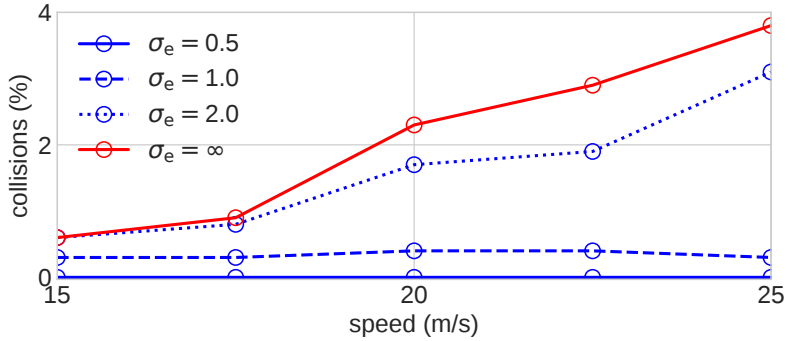
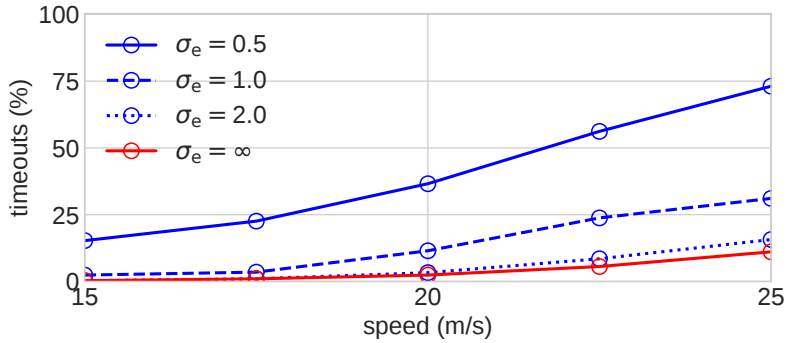


Figure 6.6: *Number of collisions and crossing time for the EQN algorithm for different levels of allowed aleatoric uncertainty, which is achieved by varying the parameter* σ_a.

As the results show, the RPF and EQN methods can increase the safety of a trained agent by estimating the epistemic uncertainty of the decisions and switch to a backup policy if the uncertainty is too high. However, possibly even more importantly, the epistemic uncertainty information could be used to guide the training process to regions of the state space where the agent needs more training. Furthermore, if an agent has been trained in a simulated environment and then is deployed in real traffic, the epistemic uncertainty information can be used to identify situations that the agent has not been trained for, which need to be added to the simulated environment.



(a) Collisions



(b) Timeouts

Figure 6.7: *Number of collisions and timeouts for the EQN agent in situations outside the training distribution. The maximum speed of the crossing vehicles is* 15 *m/s during the training process, and then the speed is gradually increased in the testing episodes.*

Chapter 7

Discussion

Chapter 4, 5, and 6 introduce different methods to create an RL-based decision-making agent for autonomous driving, by addressing the research questions from Section 1.2. This chapter highlights a few important differences and common features of these methods.

7.1 Generality

The results of Chapter 4 and 5 show that the GA, DQN, and MCTS/NN agents outperform the baseline IDM/MOBIL model by taking decisions that allow the ego vehicle to navigate through highway traffic between 5% and 10% faster. However, the main advantage of the presented methods is their generality and ability to handle driving in different environments, which is demonstrated by applying them to different types of highways and intersections. In order to apply the presented methods to a new environment, some domain knowledge is required. First, a high-level state space $\mathcal{S}$ and action space $\mathcal{A}$ need to be defined. Moreover, a reward model R that fulfills the requirements of driving in the new environment also needs to be designed, which is further discussed below. These components are enough to train the DQN-based agents (DQN, IQN, RPF, and EQN), while the GA agent requires more domain knowledge in the form of handcrafted features that the GA can build its rules and actions structure from. Of the presented methods, the MCTS/NN agent requires the most domain knowledge, since it also needs a generative model G of the environment, a belief state estimator, and possibly knowledge on how to prune actions that lead to collisions.

The GA, DQN, and MCTS/NN agents only output a decision in each state, whereas the uncertainty-aware approaches (IQN, RPF, and EQN) provide an additional estimate of the aleatoric or epistemic uncertainties. This information allows the agent to make risk-sensitive decisions and detect situations that it has not been trained for. Therefore, the uncertainty-aware approaches arguably provide more general solutions, since these methods are applicable in scenarios where such information is required.

7.2 Sample and computational complexity

Table 7.1 shows a summary of the number of training samples that were required to obtain the trained agents for the different methods. Although the agents were trained in different highway and intersection scenarios, and the point where an agent is considered fully trained is not well-defined, the significant differences still allow a few qualitative conclusions. The DQN-based agents require orders of magnitudes fewer training samples than the GA agent. This difference is not surprising, since the GA agent only updates its parameters after each generation, which consists of many episodes. The MCTS/NN agent is even more sample-efficient and requires one order of magnitude fewer training samples than the DQN-based agents. As mentioned in Section 5.1, the planning component of the MCTS/NN improves and guides the training process of the neural network. Furthermore, the pruning of actions that lead to collisions also helps to speed up the training. The required 100,000 training samples of the MCTS/NN agent corresponds to less than 30 hours of driving. However, when assessing the required number of simulated driving hours, it is important to note that the training episodes are designed to frequently expose the agent to interesting situations by initializing faster vehicles behind the ego vehicle and slower vehicles in front of the ego vehicle. Real-world highway driving often consists of monotone routine driving, which means that training from real driving will likely require significantly more data.

The computational complexity of the training process depends both on the required number of training samples and the effort of each update. The different agents were trained on a standard desktop computer and Table 7.1 provides an overview of the training time for the different methods. These training times naturally depend on, e.g., the efficiency of the implementations and the hyperparameter settings, which means that the numbers only

Table 7.1: *Required number of training samples and training time for the different agents.*

	Training samples	Training time
GA	400,000,000[1]	72 h
DQN	2,000,000	12 h
IQN	2,000,000	24 h
RPF	2,000,000	72 h
EQN	2,000,000	96 h
MCTS/NN	100,000	120 h

give a rough idea about the relative computational complexity. The DQN and IQN methods require the same number of backward passes through the neural network, which is often the most computationally expensive step, and therefore the difference in training time is relatively small. The RPF and EQN methods train an ensemble of neural networks, which require significantly more resources than the DQN method. However, the training can be parallelized, which reduces the difference compared to the DQN approach. Osband et al. report as little difference as 20% between a similar version to the RPF method and the DQN method, while presumably using an efficient implementation and many computational units [48]. The most computationally expensive algorithm is the the MCTS/NN method, due to the many MCTS iterations that are performed for each decision. However, this process can also be efficiently parallelized.

Besides computational complexity of the training process, the online computational complexity of each decision is also important for an implementation in a real vehicle. The GA agent requires little resources, since it only needs to check the conditions of its evolved rule structure to decide which action to take. The DQN agent needs to make one forward pass through the neural network for each decision, whereas the uncertainty aware approaches require more forward passes. The IQN method needs K_τ passes for the desired set of quantiles (here $K_\tau = 32$) and the RPF method needs one pass for each ensemble member, which sums up to K passes (here normally $K = 10$). The EQN method combines the complexity of the IQN and RPF method,

[1]The GA agent is evaluated and updated after each generation, which consists of multiple episodes. To provide a fair comparison, this number refers to the required number of time steps of $\Delta t = 1$ s, which equals the decision interval of the other methods.

resulting in $K_\tau K$ forward passes. Fortunately, the neural networks are relatively small compared to, e.g., networks that are used in computer vision applications, and these computations can be performed efficiently in parallel. The computational load when using the MCTS/NN agent is higher than for the DQN-based methods, due to the many MCTS iterations. Every iteration needs to traverse through the search tree, use the generative model once to sample a new state from a leaf node, and query the neural network for the prior probabilities and the value of the new leaf node. However, the MCTS/NN agent is anytime capable, i.e., it can return a result after any number of iterations. As described in Section 5.3, the performance will in general improve when more iterations are used, up to a limit, and the number of iterations that are necessary varies for different traffic situations.

7.3 Safety

When training an RL-based decision-making agent, it is important to note that the agent will only be able to solve the type of situations that it encounters during the training process. Therefore, it is crucial that the design of the training environment covers the intended scenarios. Moreover, when using machine learning to create a decision-making agent, it is difficult to guarantee functional safety of the agent's decisions. A common way to address this problem is to use an underlying safety layer, which ensures that the planned trajectory is safe before it is executed by the vehicle control system [71, 68]. Since the uncertainty-aware approaches allow the agent to make risk-sensitive decisions and detect situations outside of the training distribution, these methods could reduce the frequency of the activation of a safety layer, but they cannot independently guarantee safety.

7.4 MDP formulation

As mentioned in Section 1.3, a simple state and action space has been used for the different driving scenarios in this thesis, to provide clear interpretations and analyses of the results. Other studies consider more elaborate and generally applicable choices, and a survey of different state-action representations for autonomous driving is provided by Leurent et al. [39]. Naturally, the design of the reward model has a strong effect of the resulting driving

behavior of the agent. A simple reward model proved to work satisfactory in the scenarios considered here, but additional aspects, such as the effect on the surrounding vehicles, energy efficiency, and comfort could be included. A reward function that mimics human preferences is a compelling approach and could be determined by using inverse reinforcement learning techniques [46]. The reward function could, in theory, also be used to create a risk-sensitive policy by increasing the size of the negative rewards for collisions. However, rewards with different orders of magnitude create numerical problems, which can disrupt the training process [42]. Furthermore, for a complex reward function, it would be non-trivial to balance the different components to achieve the desired result

7.5 Neural network architecture

The neural network architecture that is invariant to permutations of the ordering of surrounding traffic participants, described in Section 4.2.1 and introduced in Paper III, is used for all the presented methods, except for the GA agent, which does not involve a neural network. As described in Section 4.2.1, this network architecture reduces the input space that the agent needs to explore by a factor of $\frac{1}{N_{\max}!}$, where $N_{\max}$ is the maximum number of considered traffic participants, compared to an architecture where the ordering matters. The results for the DQN agent confirm the benefit of this permutation invariant structure compared to a fully connected network. In practice, it would likely not have been possible to train the more complex agents with a fully connected network, although it was not tested in this work. As further discussed in Section 8.2, it is important to increase the sample efficiency of RL-based agents, and to use a permutation invariant network structure helps to achieve this goal.

Chapter 8

Conclusions and future work

As mentioned in Chapter 1, to build a ladder in order to reach the moon will show a steady progress, but this approach will never reach its goal. A more appropriate strategy would instead be to build a rocket. This thesis hypothesizes that a learning-based approach to tactical decision-making is required, as opposed to incrementally improving manually specified systems. A few components to this rocket are supplied by providing answers to the posed research questions. However, the field of using RL for autonomous driving is growing and considerable work remains until the approach can lift off. This chapter provides some concluding remarks on the introduced methods and points out a few interesting future research directions.

8.1 Concluding remarks

This thesis introduces a series of different RL-based methods for creating a tactical decision-making agent and evaluates them in various simulated driving scenarios. Although using RL for autonomous driving is at an early stage, the results are encouraging. With more research and development, RL-based approaches could potentially solve the problem of scaling the decision-making of autonomous vehicles from predefined scenarios to the complexity of the real world and remove the need to manually code solutions for the long tail of edge cases.

Chapter 4 shows that the DQN and GA agents outperform the commonly used baseline IDM/MOBIL model in different highway driving scenarios. Im-

portantly, these two methods provide a general approach that is not tailored for a specific driving scenario, which is further demonstrated by applying a family of DQN methods to different intersection scenarios in Chapter 6. Little domain knowledge is required to adapt these methods to new driving scenarios. However, one drawback that could make it challenging to scale the DQN method to create an agent that solves all types of driving scenarios is the relatively high number of required training samples. More properties of the different algorithms are discussed in Chapter 7.

One way to reduce the sample complexity of a learning-based method is to introduce more domain knowledge, which limits the policy search space. Such an approach is exploited in this work by combining planning and learning, in the form of MCTS and RL. The results show that the MCTS/NN method requires an order of magnitude fewer training samples than the DQN method and can learn strategies that consider long time horizons. Therefore, the MCTS/NN method provides a practical alternative to model-free RL for cases where a model of the intended driving scenario is available. The agent can still learn to improve its behavior, but the policy can be constrained and safety features could be incorporated in the design of the agent. However, since more domain knowledge is exploited, the effort of applying this method to a large number of different driving scenarios is higher than for model-free RL approaches.

In a real-world application of autonomous driving, a black-box agent that only outputs decisions is not sufficient. The agent also needs to provide an estimate of its confidence in the recommended decisions. This information can both be used to improve the performance of the agent itself and to improve the interaction with lower-level decision-making functionality, such as safety assurance layers. This thesis introduces the EQN method, which simultaneously estimates both the aleatoric and the epistemic uncertainty of the agent's decisions. The results demonstrate how the aleatoric uncertainty information can increase the safety of the agent by taking risk into account, and the method provides a natural way to balance the safety-efficiency trade-off. The epistemic uncertainty information can be used to detect situations that are outside the training distribution, in which the agent cannot be expected to make rational decisions. The results show that such information can increase the safety of the agent. However, the main contribution of this work is the method for providing the uncertainty estimate, since this information could be used for other purposes. For example, the exploration process of the agent could be modified to guide the training towards situations that

the agent is not yet confident about. Or, perhaps even more importantly, if an agent is first trained in a simulated world with a limited set of available traffic situations and then deployed in real traffic, the EQN method could be used to identify situations that are different from what the agent experienced during its training. These situations could then automatically be added to the simulated environment to iteratively improve the distribution of scenarios that the agent can handle.

8.2 Future research directions

The methods that are presented in this thesis are all trained and evaluated in simulated environments. While testing the performance of the methods in a larger variety of simulated driving scenarios would be interesting, a more important challenge is to take the step into the real world. Simulated environments provide many benefits, such as low-cost training samples, allowing dangerous exploratory actions, and accelerated testing. However, simulations seldom provide an entirely accurate description of the real world. Therefore, a policy that has been trained in a simulated environment may not be directly transferable to real driving. Simulated environments will likely be used for pretraining policies in the future, Pan et al. provide an early approach [50], but more research is required on how to adapt such a policy to the real world.

One of the motivations for using RL to create a tactical decision-making agent is to avoid the need of manually designing policies. However, the RL methods in this thesis still use several components that are manually specified, such as the reward function, the generative model of the MCTS/NN agent, and the simulation environments. Instead, data-driven approaches could be adopted to obtain these components. As discussed in Section 7.4, inverse RL could be used to learn the reward function [46], driver models could be learned from data [36, 6], or perhaps the whole simulation environment could be based on data.

The methods that are presented in this work require a moderate to large amount of training samples to converge to a suitable policy for the tested driving scenarios, see Table 7.1. If these methods would be trained for a large number of different simulated driving scenarios, or if the training would be performed in the real world, the required number of training samples may be too high. The sample efficiency of model-free RL methods is in general low, and how to improve the efficiency is another interesting future line of research [16].

Bibliography

[1] K. I. Ahmed, *Modeling drivers' acceleration and lane changing behavior*, PhD thesis, Massachusetts Institute of Technology, 1999.

[2] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. V. Covern, and M. Webster, *Odin: Team VictorTango's entry in the DARPA urban challenge*, Journal of Field Robotics, 25 (2008), pp. 467–492.

[3] H. Bai, D. Hsu, and W. S. Lee, *Integrated perception and planning in the continuous space: A POMDP approach*, International Journal of Robotics Research, 33 (2014), pp. 1288–1302.

[4] M. Bansal, A. Krizhevsky, and A. Ogale, *ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst*, in Proceedings of Robotics: Science and Systems, 2019.

[5] M. G. Bellemare, W. Dabney, and R. Munos, *A distributional perspective on reinforcement learning*, in Proceedings of the 34th International Conference on Machine Learning (ICML), 2017, pp. 449–458.

[6] R. P. Bhattacharyya, D. J. Phillips, B. Wulfe, J. Morton, A. Kuefler, and M. J. Kochenderfer, *Multi-agent imitation learning for driving simulation*, in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1534–1539.

[7] M. Bouton, J. Karlsson, A. Nakhaei, K. Fujimura, M. J. Kochenderfer, and J. Tumova, *Reinforcement learning with probabilistic guarantees for autonomous driving*, in Proceedings of the Workshop on Safety, Risk and Uncertainty in Reinforcement Learning, 34th Conference on Uncertainty in Artificial Intelligence (UAI), 2018.

[8] S. Brechtel, T. Gindele, and R. Dillmann, *Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs*, in Proceedings of the 17th IEEE International Conference on Intelligent Transportation Systems (ITSC), 2014, pp. 392–399.

[9] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, *A survey of Monte Carlo tree search methods*, IEEE Transactions on Computational Intelligence and AI in Games, 4 (2012), pp. 1–43.

[10] J. Chen, B. Yuan, and M. Tomizuka, *Model-free deep reinforcement learning for urban autonomous driving*, in Proceedings of the 22nd IEEE Intelligent Transportation Systems Conference (ITSC), 2019, pp. 2765–2771.

[11] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, *Continuous upper confidence trees*, in Proceedings of the International Conference on Learning and Intelligent Optimization, 2011, pp. 433–445.

[12] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, *Implicit quantile networks for distributional reinforcement learning*, in Proceedings of the 35th International Conference on Machine Learning (ICML), 2018, pp. 1096–1105.

[13] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, *Distributional reinforcement learning with quantile regression*, in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018, pp. 2892–2901.

[14] F. Damerow and J. Eggert, *Risk-aversive behavior planning under multiple situations with uncertainty*, in Proceedings of the 18th IEEE

International Conference on Intelligent Transportation Systems (ITSC), 2015, pp. 656–663.

[15] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[16] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, *First return then explore*, Nature, 590 (2021), pp. 580–586.

[17] D. J. Fagnant and K. Kockelman, *Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations*, Transportation Research Part A: Policy and Practice, 77 (2015), pp. 167–181.

[18] D. González, J. Pérez, V. Milanés, and F. Nashashibi, *A review of motion planning techniques for automated vehicles*, IEEE Transactions on Intelligent Transportation Systems, 17 (2016), pp. 1135–1145.

[19] P. Hart, N. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on Systems Science and Cybernetics, 4 (1968), pp. 100–107.

[20] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, *Rainbow: Combining improvements in deep reinforcement learning*, in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018, pp. 3215–3222.

[21] J. Ho and S. Ermon, *Generative adversarial imitation learning*, in Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS), 2016.

[22] C. J. Hoel, *Source code for 'Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation'*, 2020. https://github.com/carljohanhoel/BayesianRLForAutonomousDriving.

[23] C. J. Hoel, *Source code for 'Ensemble quantile networks: Uncertainty-aware reinforcement learning with applications in autonomous driving'*, 2021. https://github.com/carljohanhoel/EnsembleQuantileNetworks.

[24] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[25] P. J. Huber, *Robust estimation of a location parameter*, The Annals of Mathematical Statistics, 35 (1964), pp. 73–101.

[26] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, *Navigating occluded intersections with autonomous vehicles using deep reinforcement learning*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 2034–2039.

[27] J. Janai, F. Güney, A. Behl, and A. Geiger, *Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art*, Now Publishers Inc., 2020.

[28] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, *Anytime motion planning using the RRT**, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 1478–1483.

[29] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, *HG-DAgger: Interactive imitation learning with human experts*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2019, pp. 8077–8083.

[30] 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*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2019, pp. 8248–8254.

[31] A. Kesting, M. Treiber, and D. Helbing, *General lane-changing model MOBIL for car-following models*, Transportation Research Record, 1999 (2007), pp. 86–94.

[32] 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*, IEEE Transactions on Intelligent Transportation Systems, (2021), pp. 1–18.

[33] A. D. Kiureghian and O. Ditlevsen, *Aleatory or epistemic? Does it matter?*, Structural Safety, 31 (2009), pp. 105–112.

[34] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*, MIT Press, 2015.

[35] M. Kuderer, S. Gulati, and W. Burgard, *Learning driving styles for autonomous vehicles from demonstration*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 2641–2646.

[36] A. Kuefler, J. Morton, T. Wheeler, and M. J. Kochenderfer, *Imitating driver behavior with generative adversarial networks*, in Proceedings of the IEEE Intelligent Vehicles Symposium (IV), 2017, pp. 204–211.

[37] S. M. Lavalle, *Rapidly-exploring random trees: A new tool for path planning*, tech. rep., Iowa State University, 1998.

[38] Y. LeCun, Y. Bengio, and G. E. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[39] E. Leurent, Y. Blanco, D. Efimov, and O.-A. Maillard, *A survey of state-action representations for autonomous driving.* hal-01908175, 2018.

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, in Proceedings of the 4th International Conference on Learning Representations (ICLR), 2016.

[41] J. A. Michon, *A critical view of driver behavior models: What do we know, what should we do?*, in Human Behavior and Traffic Safety, L. Evans and R. Schwing, eds., Springer, 1985, pp. 485–524.

[42] V. Mnih et al., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[43] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer,

A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, *Junior: The Stanford entry in the urban challenge*, Journal of Field Robotics, 25 (2008), pp. 569–597.

[44] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, *Tactical decision making for lane changing with deep reinforcement learning*, in Proceedings of the Workshop on Machine Learning for Intelligent Transportation Systems, 32nd Conference on Neural Information Processing Systems (NeurIPS), 2017.

[45] A. Y. Ng, D. Harada, and S. Russell, *Policy invariance under reward transformations: Theory and application to reward shaping*, in In Proceedings of the Sixteenth International Conference on Machine Learning, Morgan Kaufmann, 1999, pp. 278–287.

[46] A. Y. Ng and S. J. Russell, *Algorithms for inverse reinforcement learning*, in Proceedings of the 17th International Conference on Machine Learning (ICML), 2000, pp. 663–670.

[47] I. Osband, J. Aslanides, and A. Cassirer, *Randomized prior functions for deep reinforcement learning*, in Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS), 2018.

[48] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, *Deep exploration via bootstrapped DQN*, in Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS), 2016.

[49] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, *A survey of motion planning and control techniques for self-driving urban vehicles*, IEEE Transactions on Intelligent Vehicles, 1 (2016), pp. 33–55.

[50] X. Pan, Y. You, Z. Wang, and C. Lu, *Virtual to real reinforcement learning for autonomous driving*, in Proceedings of the 28th British Machine Vision Conference (BMVC), 2017, pp. 11.1–11.13.

[51] D. A. Pomerleau, *ALVINN: An autonomous land vehicle in a neural network*, in Proceedings of the 1st Conference on Neural Information Processing Systems (NeurIPS), 1989, pp. 305–313.

[52] S. Ross, G. Gordon, and D. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning*, in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, 2011, pp. 627–635.

[53] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 3 ed., 2010.

[54] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, *Planning for autonomous cars that leverage effects on human actions*, in Proceedings of Robotics: Science and Systems, 2016.

[55] S. Shalev-Shwartz, S. Shammah, and A. Shashua, *Safe, multi-agent, reinforcement learning for autonomous driving.* arXiv:1610.03295, 2016.

[56] S. Sharifzadeh, J. Chiotellis, R. Triebel, and D. Cremers, *Learning to drive using inverse reinforcement learning and deep Q-networks*, in Proceedings of the Workshop on Deep Learning for Action and Interaction, 29th Conference on Neural Information Processing Systems (NeurIPS), 2016.

[57] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*, Science, 362 (2018), pp. 1140–1144.

[58] 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, 550 (2017), pp. 354–359.

[59] S. Singh, *Critical reasons for crashes investigated in the national motor vehicle crash causation survey*, Tech. Rep. DOT HS 812 506, National Highway Traffic Safety Administration, 2018.

[60] E. Sonu, Z. Sunberg, and M. J. Kochenderfer, *Exploiting hierarchy for scalable decision making in autonomous driving*, in Proceed-

ings of the IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 2203–2208.

[61] L. Sun, W. Zhan, C.-Y. Chan, and M. Tomizuka, *Behavior planning of autonomous cars with social perception*, in Proceedings of the IEEE Intelligent Vehicles Symposium (IV), 2019, pp. 207–213.

[62] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, *The value of inferring the internal state of traffic participants for autonomous freeway driving*, in Proceedings of the American Control Conference (ACC), 2017, pp. 3004–3010.

[63] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2 ed., 2018.

[64] L. Svensson, L. Masson, N. Mohan, E. Ward, A. P. Brenden, L. Feng, and M. Törngren, *Safe stop trajectory planning for highly automated vehicles: An optimal control problem formulation*, in Proceedings of the IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 517–522.

[65] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, *Learning negotiating behavior between cars in intersections using deep Q-learning*, in Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 3169–3174.

[66] N. Tran et al., *Global status report on road safety 2018*, tech. rep., World Health Organization, 2018.

[67] M. Treiber, A. Hennecke, and D. Helbing, *Congested traffic states in empirical observations and microscopic simulations*, Physical Review E, 62 (2000), pp. 1805–1824.

[68] M. Törngren, X. Zhang, N. Mohan, M. Becker, L. Svensson, X. Tao, D.-J. Chen, and J. Westman, *Architecting safety supervisors for high levels of automated driving*, in Proceedings of the 21st International Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 1721–1728.

[69] S. Ulbrich and M. Maurer, *Towards tactical lane change behavior planning for automated vehicles*, in Proceedings of the 18th IEEE International Conference on Intelligent Transportation Systems (ITSC), 2015, pp. 989–995.

[70] S. Ulbrich, A. Reschka, J. Rieken, S. Ernst, G. Bagschik, F. Dierkes, M. Nolte, and M. Maurer, *Towards a functional system architecture for automated vehicles*. arXiv:1703.08557, 2017.

[71] S. Underwood, D. Bartz, A. Kade, and M. Crawford, *Truck automation: Testing and trusting the virtual driver*, in Road Vehicle Automation 3, G. Meyer and S. Beiker, eds., Springer, 2016, pp. 91–109.

[72] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, *Autonomous driving in urban environments: Boss and the urban challenge*, Journal of Field Robotics, 25 (2008), pp. 425–466.

[73] P. Wang, H. Li, and C.-Y. Chan, *Continuous control for automated lane change behavior based on deep deterministic policy gradient algorithm*, in Proceedings of the IEEE Intelligent Vehicles Symposium (IV), 2019, pp. 1454–1460.

[74] P. Wang, D. Liu, J. Chen, H. Li, and C. Chan, *Decision making for autonomous driving via augmented adversarial inverse reinforcement learning*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2021.

[75] C. J. C. H. Watkins and P. Dayan, *Q-learning*, Machine Learning, 8 (1992), pp. 279–292.

[76] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, *Optimal trajectory generation for dynamic street scenarios in a Frenét frame*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2010, pp. 987–993.

[77] F. Ye, S. Zhang, P. Wang, and C.-Y. Chan, *A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles.* arXiv:2105.14218, 2021.

[78] Z. Zhu and H. Zhao, *A survey of deep RL and IL for autonomous driving policy learning.* arXiv:2101.01993, 2021.

[79] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, *Planning-based prediction for pedestrians*, in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2009, pp. 3931–3936.

Paper I

An evolutionary approach to general-purpose automated speed and lane change behavior

in

Proceedings of the 16th IEEE International Conference on Machine Learning and Applications, Cancun, Mexico, 2017, pp. 743-748.

An Evolutionary Approach to General-Purpose Automated Speed and Lane Change Behavior

Carl-Johan Hoel*†, Mattias Wahde*, Krister Wolff*
*Chalmers University of Technology, 412 96 Göteborg, Sweden
†Volvo Group Trucks Technology, 405 08 Göteborg, Sweden
Email: {carl-johan.hoel, mattias.wahde, krister.wolff}@chalmers.se

***Abstract*—This paper introduces a method for automatically training a general-purpose driver model, applied to the case of a truck-trailer combination. A genetic algorithm is used to optimize a structure of rules and actions, and their parameters, to achieve the desired driving behavior. The training is carried out in a simulated environment, using a two-stage process. The method is then applied to a highway driving case, where it is shown that it generates a model that matches or surpasses the performance of a commonly used reference model. Furthermore, the generality of the model is demonstrated by applying it to an overtaking situation on a rural road with oncoming traffic.**

I. Introduction

Technology for limited automation of vehicles has been available for some time. Different functions exist to automate specific driving tasks, such as adaptive cruise control, lane keeping systems, automated parallel parking etc. One of the more mature areas of automated driving is highway driving. Much of the complexity from e.g. city driving is here reduced, and during normal operation (excluding special events such as road work etc.) the task is limited to adapt the vehicle speed, follow the intended lane, and carry out lane changes.

Many algorithms for speed control exist; for an overview, see e.g. [1]. One commonly used model for speed control is the *Intelligent driver model* (IDM) [2], which is a simple but robust model that considers the relative distance to, and speed of, the preceding vehicle. In order to decide whether or not a lane change is feasible, different gap acceptance models are often used; see, for example, [3] or [4]. Another algorithm, *Minimize overall braking induced by lane changes* (MOBIL) [5], tries to minimize the longitudinal acceleration of all participants in a traffic situation. A so called politeness factor determines the level of priority of the vehicle under consideration (henceforth referred to as the *ego vehicle*) compared to that of surrounding vehicles. There is also a variety of more advanced algorithms for lane changing, taking into account the risk and utility associated with a lane change [6] or solving the problem as a partially observable Markov decision process [7].

However, most existing methods for automated driving, for example the methods mentioned above, are tailored to handle a specific case, such as highway driving. As soon as a different setting is considered, e.g. driving on a rural road with oncoming traffic instead of a highway, an entirely different method may be required. Moreover, human drivers do not in general drive in a continuously error-minimizing way, as is often assumed in methods based on classical control theory. Instead, human driving is a more complex process involving several cognitive tasks such as neuro-muscular control, monitoring, and decision-making, see [8] and [9].

Therefore, in this paper an alternative method for automated driving is introduced, in which a genetic algorithm (GA) with length-varying chromosomes is used for optimizing a set of rules in simulation, in order to form a complete automated driver model for a given situation, although the number of available lanes, the speed of the vehicles, the presence or absence of oncoming traffic, etc. can differ. The GA is applied in an open-ended manner, optimizing both the structure (i.e. the number of rules, as well as the types of rules used) and the parameters of the set of rules constituting the driver model. For details of GAs in general, see e.g. [10]

The main strength of the proposed method is its ability for generalization: It is shown that this approach is able to generate a driver model for highway driving with a performance that matches the combination of the IDM and the MOBIL model. Moreover, it is shown that a given rule set can, with some tuning, be applied in an entirely different setting, namely (in this case) driving on a rural road with oncoming traffic.

This paper is organized as follows: Sect. II gives an overview of the IDM and the MOBIL model, which are used for comparison, and describes how the simulations were set up. The genetic algorithm is outlined in Sect. III. Next, in Sect. IV, the results are presented, followed by a discussion in Sect. V. Finally the conclusions are given in Sect. VI.

II. Simulation environment

As a baseline for comparison, a reference driver model, combined from the IDM and MOBIL model, was used for longitudinal control and for lane change decisions. Highway traffic was simulated, which included frequent speed changes to accelerate the training process. This section describes the reference model and how the simulations were set up.

A. Reference model

The IDM [2] has often been used for driver modeling, driver assistance systems, and traffic simulations. It is a longitudinal model of the ego vehicle's speed, v, according to

$$\dot{v} = a\left(1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right), \tag{1}$$

$$s^*(v,\Delta v) = s_0 + vT + v\Delta v/(2\sqrt{ab}). \tag{2}$$

Here v is the speed of the ego vehicle, Δv is the speed difference between the ego vehicle and the vehicle in front (approach rate), and s is the distance to the vehicle in front of

the ego vehicle. These quantities act as the inputs to the model, whereas the tuning parameters are the minimum gap distance, s_0, the safe time headway, T, the maximal acceleration, a, the desired deceleration, b, the acceleration exponent δ, and the desired free speed, v_0.

The MOBIL model [5] uses incentives and safety criteria to decide if a lane change should take place. The goal of the model is to minimize the overall negative acceleration of all participants in a traffic situation. In order to estimate the future accelerations of both the ego vehicle and the surrounding vehicles, when choosing to stay in lane or to change lane, the IDM is applied to the different cases.

Two criteria need to be fulfilled to perform a lane change. The first requires that the deceleration of the trailing vehicle in the target lane, $\tilde{a}_n$, must not exceed a safe limit, b_{safe}, i.e. $\tilde{a}_{\text{n}} > -b_{\text{safe}}$. The second criterion states that the incentive to make a lane change should be larger than a specific threshold. The incentive is calculated from the induced accelerations of the surrounding vehicles both when performing, and when not performing, a lane change. This criterion is described by

$$\tilde{a}_{\text{e}} - a_{\text{e}} + p\left((\tilde{a}_{\text{n}} - a_{\text{n}}) + (\tilde{a}_{\text{o}} - a_{\text{o}})\right) > a_{\text{th}}, \qquad (3)$$

where $\tilde{a}_{\text{e}}$, $\tilde{a}_{\text{n}}$ and $\tilde{a}_{\text{o}}$ are the accelerations of the ego vehicle, the trailing vehicle in the target lane, and the trailing vehicle in the current lane, respectively, in the case where the lane change is carried out. Similarly, a_{e}, a_{n} and a_{o} are the accelerations of the ego vehicle, the trailing vehicle in the target lane, and the trailing vehicle in the current lane, respectively, in the case where the lane change does not occur. The politeness factor p controls how much the effect on the surrounding vehicles is considered. Furthermore, the threshold a_{th} decides how advantageous a potential lane change should be in order to be executed. In summary, the model weighs the own advantage (acceleration gain) versus the sum of the disadvantages for other vehicles (acceleration losses).

The same criteria are applied both to the left and right lanes, if they are both available. In cases where the criteria are fulfilled both to the left and right, the most advantageous option, in the form of the highest acceleration gain, is chosen.

In this study, the parameters (given in Table I) from the original papers [2] and [5] were used.

B. Traffic simulation

The main case where the method presented in this paper was applied involved highway driving. More specifically, a straight three-lane highway was used, with U.S. passing rules, i.e. such that it was allowed to overtake a vehicle both on the left side and the right side. Each simulation consisted of one ego vehicle to be controlled and 9 surrounding vehicles, which followed the IDM model longitudinally and stayed in their lanes. In all cases, the ego vehicle consisted of a 16.5 m long truck-semitrailer combination, whereas the surrounding vehicles were normal passenger cars.

Normal highway driving often involves traffic with more or less constant speeds and small accelerations. However, occasionally, vehicles have to brake hard, or even carry out emergency braking to avoid collisions. In order to train a model that can handle situations like these using machine learning, naturally such events must be included in the training simulations. All the vehicles in the simulation were assigned different desired speed trajectories, some examples of which are given in Fig. 1. With the aim of speeding up the training of the model, traffic was simulated with frequent speed changes.

TABLE I
IDM AND MOBIL MODEL PARAMETERS.

Minimum gap distance, s_0	2.0 m
Safe time headway, T	1.6 s
Maximal acceleration, a	0.7 m/s^2
Desired deceleration, b	1.7 m/s^2
Acceleration exponent, δ	4
Politeness factor, p	1
Changing threshold, a_{th}	0.1 m/s^2
Maximum safe deceleration, b_{safe}	4.0 m/s^2

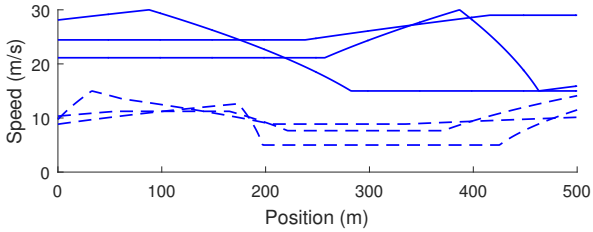


Fig. 1. Example of six different randomly generated speed trajectories, defined for different positions along the highway. The solid lines are fast trajectories, applied to vehicles starting behind the ego vehicle, whereas the dashed lines are slow trajectories, applied to vehicles starting in front of the ego vehicle.

The ego vehicle always started in the middle lane. Then the 9 surrounding vehicles were randomly positioned around it, within a distance d_{long} longitudinally, set to 150 m. All vehicles were separated with a minimum inter-vehicle distance d_{Δ}, set to 20 m. The surrounding vehicles were assigned desired speed trajectories, which were randomly generated by the following procedure: If the vehicle was placed in front of the ego vehicle, an initial speed was randomly chosen in the range $[v^+_{\text{min}}, v^+_{\text{max}}]$, here taken as $[5, 15]$ m/s. If instead it was placed behind the ego vehicle, the initial speed was chosen in the range $[v^-_{\text{min}}, v^-_{\text{max}}]$, here taken as $[15, 30]$ m/s. Then, with a sampling time of 1 s, a choice was made (with equal probability) either to keep the speed constant or to apply a constant acceleration. With equal probability, a positive or negative acceleration was used. The acceleration was drawn from a normal distribution with mean 0 and variance σ m/s^2. In case of positive accelerations, negative values were rejected, and vice versa for negative accelerations. σ was equal to 1 for positive accelerations, and 5 for negative accelerations. The modulus of the acceleration was limited to 2σ. This acceleration was kept constant for a randomly selected duration in the range $[t_{\text{min}}, t_{\text{max}}]$, set to $[2, 20]$ s for positive accelerations and $[0.4, 4]$ s for negative accelerations.

Furthermore, for vehicles initialized in front of the ego vehicle, the speed was limited to the range $[v^+_{\text{min}}, v^+_{\text{max}}]$ (see above), and for vehicles initialized behind the the ego vehicle, to the range $[v^-_{\text{min}}, v^-_{\text{max}}]$. The initial desired ego vehicle speed, $v^{\text{ego}}_{\text{init}}$, and maximum ego vehicle speed, $v^{\text{ego}}_{\text{max}}$, were set to 15 m/s and 20 m/s respectively. Finally, scenarios where any two vehicles were placed too close together with a large speed difference, thus causing an unavoidable collision, were deleted.

An example of an initial traffic situation is shown in Fig. 2.

Fig. 2. Example of an initial traffic situation. The ego vehicle (a truck-trailer combination) is shown in red, in the middle lane. The arrows represent the velocities of the vehicles.

TABLE II
LATERAL CONTROL PARAMETERS.

Distance to near point	5 m
Distance to far point	100 m
Proportional gain far point, k_f	20
Proportional gain near point, k_n	9
Integral gain near point, k_I	10 s^{-1}

C. Vehicle and lateral control model

For each vehicle a lane-following controller was implemented. A two-point visual control model [11] was used,

$$\dot{\delta} = k_f \dot{\theta}_f + k_n \dot{\theta}_n + k_I \theta_n, \tag{4}$$

where $\dot{\delta}$ is the steering angle, and θ_n and θ_f are the perceived angles to a near and far point in the intended lane, respectively. k_f, k_n and k_I are control parameters. The parameter values are given in Table II.

The vehicles were modeled using a simple kinematic model. Regardless of the desired acceleration (see Sect. II-B), the actual acceleration of the vehicles was limited to the range $[a^\mathrm{M}_\mathrm{min}, a^\mathrm{M}_\mathrm{max}]$, here taken as $[-10, 2]$ m/s^2, and the maximum speed was limited to $v^\mathrm{M}_\mathrm{max} = 30$ m/s.

III. GENETIC ALGORITHM

As mentioned in Sect. I, a GA with length-varying chromosomes was used to train a rule-based driver model in simulation. GAs are suitable for optimization problems with non-differentiable objective functions, or even problems that lack a complete mathematical model, where instead a simulation has to be used. Another strength of GAs is their ability to handle complex problems that have irregular fitness landscapes with many local optima, and a varying number of variables [10].

In GAs it is common to use chromosomes with lengths that are preserved during the evolutionary process. However, it is also possible to use a so called non-homologous crossover operator that allows the chromosomes to vary in length during the evolution [12]. Such a crossover operator was implemented in this study, see Sect. III-C for details.

In the simulations, the ego vehicle is controlled by the evolved driver model, whereas the 9 surrounding vehicles are controlled by the standard IDM described in Sect. II-A. The driver model of the ego vehicle consists of a sequence of rules and actions (described below) which are executed with a sampling interval of $\Delta t = 0.1$ s.

A. Chromosome encoding

In the encoding used, a chromosome encodes a set of *instructions*, each represented by 4 genes, $g_1, \ldots, g_4$. The first two numbers in each instruction are integers and represent which rule or action that should be used, and which relative lane that should be considered. The last two parameters are floating-point numbers that have different meanings for different instructions. The detailed interpretation of the numbers is described in Table III. For example, the instruction $[0, 1, 0.2, 0.7]$ would be translated to the instruction (rule): *If there is a vehicle in the left lane, in the interval -60 m to 40 m longitudinally, relative to the ego vehicle, then ...* A braking or accelerating pedal level is linearly mapped to the allowed negative or positive acceleration ranges mentioned in Sect. II-C.

A chromosome is constructed from a variable number of instructions such that, when decoded, it generates a driver model in the form of a list of *rule-action units*. For example, a case with three rule-actions units, could take the form:

- *Rule 1*
  - *Rule 2*
    - ▷ *Action 1*
- *Rule 3*
  - ▷ *Action 2*

▷ *Action 3*

Note that every rule-action unit contains precisely one action, and any number of rules. During evaluation of a chromosome, when deciding which action to take, the first rule is considered. If the conditions are satisfied, the next rule is considered etc. If all rules preceding an action are fulfilled, that action is performed and the evaluation is stopped. This means that at most one action can be executed. If a rule is not fulfilled, the evaluation jumps to the next group of rules preceding the next action. In the example, if, for instance, Rule 1 is not satisfied, Rule 3 would be considered next. If there are no rules associated with an action, as in Action 3 in the example, this action will be used if no preceding action has been carried out. If there is no lane available in a given direction, all rule-action units containing an action involving a lane change in that direction are ignored. Moreover, if a lane is not available, it is considered to contain no vehicles. For actions with relative lane 0 (see Table III), no lane change is performed and only acceleration is considered.

Furthermore, note that the evaluation of the chromosome always takes place every Δt s, even if the chosen action involves a lane change. In that case, the intended lane in the control model, described in Sect. II-C, is immediately switched to the desired lane, and kept there until any future action changes it.

B. Fitness measure

In order to evaluate the performance of an evolved driver model controlling the ego vehicle, the simulation environment described in Sect. II-B was used. Each scenario was initialized randomly and run until a collision occurred or the ego vehicle reached a maximum distance, d_max, in this case 500 m, solving the scenario without collision.

The fitness measure was divided into a positive and a negative part. The positive fitness, ranging from 0 to 1, was

$$f_i^+ = (d/d_\mathrm{max}) \times \min\left(\bar{v}/\bar{v}_\mathrm{ref}, 1\right), \tag{5}$$

where d is the distance driven by the ego vehicle, and d_max is the maximum distance possible. $\bar{v}$ is the average speed of the ego vehicle and $\bar{v}_\mathrm{ref}$ is the average speed when applying the reference model presented in Sect. II-A. As can be seen in the equation, the speed part was limited to the range 0 to

TABLE III
ENCODING OF AN INSTRUCTION, CONSISTING OF 4 GENES

Gene	Value	Interpretation
g_1	0	Rule: If there is a vehicle in lane g_2, in the interval g_3 to g_4 longitudinally, relative to the ego vehicle
	1	Rule: If there is no vehicle in lane g_2, in the interval g_3 to g_4 longitudinally, relative to the ego vehicle
	2	Action: Change to relative lane g_2, brake or accelerate according to g_3, using pedal level g_4
g_2	-1	Right lane
	0	Current lane
	1	Left lane
g_3	$v_3 \in [0, 1]$	if g_1 = 0 or 1, map value v_3 to [-100,100] m if g_1 = 2, g_3 represents braking if $v_3 < 0.5$ and acceleration if $v_3 \geq 0.5$
g_4	$v_4 \in [0, 1]$	if g_1 = 0 or 1, map value v_4 to [-100,100] m if g_1 = 2, g_4 represents pedal level

1, which means that no additional fitness score was given to individuals driving faster than the reference model. This was so, since the model is intended to emphasize safety.

Therefore, when a collision occurred, the distance driven by the ego vehicle was the dominant limiting factor. By contrast, if a model managed to drive without collision, the fitness was determined by the average speed.

Regularization was implemented, in order to avoid overfitting, as a negative fitness contribution depending on the length of the chromosome. Here, up to n instructions were allowed without penalty, but beyond that limit, a negative fitness contribution was computed, somewhat arbitrarily, as

$$f^- = \beta \max\left(0, L/4 - n\right), \tag{6}$$

where L is the length of the chromosome. Since there are 4 genes per instruction, $L/4$ is the number of instructions. Based on preliminary investigations, suitable values for n and β were found to be 20 and 0.2, respectively.

In every GA run, each driver model was evaluated using $i_0 = 10$ random scenarios. Then, whenever all scenarios were solved without collision and the speed part of the fitness was greater than a given threshold (set to 0.85) in 95% of the considered scenarios, an additional scenario was added. The total fitness f was then calculated as the sum of the positive fitness for every scenario minus the overall negative fitness based on the chromosome length, as

$$f = \sum_i f_i^+ - f^-. \tag{7}$$

C. Evolutionary operators and parameters

In each GA run, a population of m individuals was randomly initialized with between 10 and 20 instructions each. The individuals were encoded according to Sect. III-A and were decoded and evaluated as described in Sect. III-B.

When a new generation had been evaluated, a standard tournament selection procedure was applied, with tournament size k and tournament selection parameter p_t. Then crossover was applied with probability p_c to every pair of selected individuals. A non-homologous two-point crossover operator was used, in which the section between the two crossover points was swapped between the two selected individuals, thus allowing the length of the chromosomes of the new individuals to be different from their parents. The crossover points were chosen between the instructions.

TABLE IV
PARAMETERS USED FOR THE EVOLUTIONARY ALGORITHM.

Population size, m	40
Tournament size, k	4
Tournament selection parameter, p_t	0.8
Crossover probability, p_c	0.8
Mutation rate	$1/L$

Following selection and crossover, the individuals were exposed to random mutation. Mutations occurred with a probability computed as the inverse of the chromosome length L, i.e. on average one gene per chromosome was mutated. Note that this means that by mutating the first gene of an instruction, a rule can change into an action and vice versa. In addition to the parametric mutation just described, two additional mutation types were used (also with probability $1/L$). The first one either inserted or deleted one instruction, i.e. one rule or one action. The second one inserted or deleted a rule-action unit. In case of addition, the rule-action unit was taken as a mutated copy of one of the already existing rule-action units in the chromosome.

Finally, elitism was used, meaning that a single copy of the best individual in a given generation was inserted in the next generation without modification. The numerical values of the parameters are summarized in Table IV.

IV. RESULTS

Three different runs (Runs 1 - 3) were carried out with different random seeds of the GA. The resulting fitness variation over the generations is shown in Fig. 3, showing an early increase in fitness as more and more scenarios are solved. However, in all three runs, when eventually a general solution is found, which solves all 500 cases without collision at which point, the fitness jumps to just below 500 (since the average speed of the ego vehicle is generally smaller than that of the reference model in a few scenarios). The final driver model from one of these runs (Run 1) is shown in Table V. The evolved driver model generates a behavior where the vehicle stays in its lane and accelerates if no vehicle is close in front of it. If a vehicle is present, but there is no vehicle in the left lane, it changes lanes to the left lane. If also the right lane is occupied, it stays in its own lane and brakes. Otherwise it changes to the right lane.

It should be noted that some of the rules or rule-action units do not have any effect on the behavior of the generated driver model, i.e. they are never executed (see e.g. the second and third rule in the fourth rule-action unit, and rule-action unit 7 in Table V). For a discussion on why they are kept, see Sect. V. Furthermore, note that rule-action unit 7 and 8 can only be considered when the vehicle is positioned in the right lane, resulting in rule-action unit 6 being ignored.

In order to investigate whether or not an even simpler driver model could be found, an additional run (Run 4) was carried out, with negative fitness contribution applied from the beginning (i.e. $n = 0$ in Eq. 6). For this run, the optimization was initialized with a population consisting of mutated copies of the the final best individual of Run 1. Furthermore, this time, since a feasible structure for solving the problem was already present in the population, the crossover operator was removed. Evolution was therefore only done by selection and

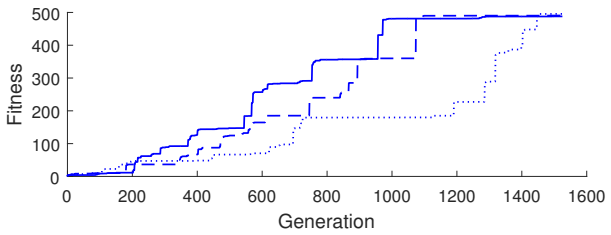


Fig. 3. Fitness variation of the best individual in the population, for three optimization runs (Runs 1 - 3) with different random seeds.

TABLE V
THE RESULTING DRIVER MODEL FROM RUN 1.

- If no vehicle in ego lane is within [-2.0, 21.5] m
  - If no vehicle in right lane is within [28.5, 62.0] m
    - ▷ Keep lane, accelerate with pedal level 0.94
- If no vehicle in left lane is within [-19.5, 80.5] m
  - ▷ Do lane change to the left, accelerate with pedal level 0.65
- If no vehicle in left lane is within [-18.6, 82.3] m
  - ▷ Do lane change to the left, brake with pedal level 0.12
- If vehicle in right lane is within [-92.0, 90.1] m
  - If no vehicle in ego lane is within [-2.2, 37.1] m
    - If no vehicle in ego lane is within [-2.2, 37.1] m
      - ▷ Keep lane, accelerate with pedal level 0.94
- If vehicle in right lane is within [-67.6, 91.3] m
  - If vehicle in right lane is within [-18.1, 81,7] m
    - ▷ Keep lane, brake with pedal level 0.88
- ▷ Do lane change to the right, accelerate with pedal level 0.80
- If no vehicle in ego lane is within [-44.5,85.3] m
  - ▷ Keep lane, brake with pedal level 0.88
- If vehicle in left lane is within [-77.1,46.6] m
  - If no vehicle in ego lane is within [28.6,84.2] m
    - ▷ Keep lane, brake with pedal level 0.88

TABLE VI
THE RESULTING DRIVER MODEL FROM RUN 4, WHICH PENALIZED THE TOTAL NUMBER OF INSTRUCTIONS.

- If no vehicle in ego lane is within [-3.4, 21.5] m
  - If no vehicle in right lane is within [63.3, 99.7] m
    - ▷ Keep lane, accelerate with pedal level 0.94
- If no vehicle in left lane is within [-17.8, 77.2] m
  - ▷ Do lane change to the left, accelerate with pedal level 0.97
- If no vehicle in ego lane is within [-2.2, 38.1] m
  - ▷ Keep lane, accelerate with pedal level 1.00
- If vehicle in right lane is within [-18.1, 40.9] m
  - ▷ Keep lane, brake with pedal level 0.88
- ▷ Do lane change to the right, accelerate with pedal level 0.78
- If vehicle in left lane is within [-75.7, 46.6] m
  - ▷ Keep lane, brake with pedal level 0.88
- ▷ Do lane change to the left, accelerate with pedal level 0.86

mutation. Over 300 generations, the overall fitness improved from 483 to 491. This was partly due to a shortening of the best chromosome, reducing the negative part of the fitness measure, and partly due to further optimization of the parameters of the instructions. The final driver model is shown in Table VI.

The resulting driver model from Run 4 was finally applied to 500 new scenarios, different from the ones over which it was trained. Importantly, the model ran through all these scenarios without collisions. Fig. 4 shows a histogram of the speed ratio $\gamma = \bar{v}/\bar{v}_{\mathrm{ref}}$ over the 500 test scenarios. This figure shows that, even though the average speed of the evolved model generally was close to that of the reference model, there are also some outliers, which are either better or worse (in terms of average speed) than the reference model. This is due to the randomness of the scenarios, where even a reasonable action can lead to a situation where the ego vehicle gets locked in, without a possibility to overtake the surrounding vehicles. Since the

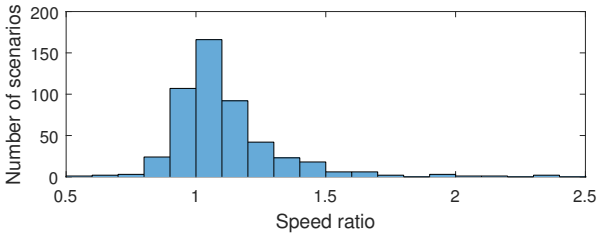


Fig. 4. Histogram of the speed ratio $\gamma = \bar{v}/\bar{v}_{\mathrm{ref}}$, obtained when applying the resulting driver model from Run 4 to 500 new scenarios.

models behave differently, such situations can occur both for the evolved model and the reference model, which explains the outliers. Note that, even though no additional fitness was given for models driving faster than the reference model, the average of γ was 1.11, indicating that the evolved model is slightly more efficient than the reference model.

V. DISCUSSION

The results show that useful driver models, whose performance equals or surpasses that of the reference model in terms of safety (at least over the scenarios considered; see below) and average speed, can be obtained with the proposed method. Reassuringly, the behavior shown by the driver models presented in Tables V and VI is similar to a gap acceptance model for lane changes, as presented in e.g. [3] or [4], where a lane change can be performed if the gap in the traffic is large enough. Moreover, unlike most driver models, the approach presented here is not limited only to a single, specific case (e.g. one-way traffic). In order to illustrate this property, a similar traffic simulation as that described in Sect. II-B was set up, with two lanes but with *oncoming traffic* in the left lane, see Fig. 5. The ego vehicle started in the right lane, with another vehicle in front, following a slow speed profile. Two oncoming vehicles were randomly placed between 200 and 1000 m ahead of the ego vehicle in the oncoming lane. They both followed a fast speed profile. The same evolutionary algorithm as described in Sect. III was applied to this case, using the same type of instructions as before (see Table III). Here, the fitness measure was simply taken as $f_i^+ = (d/d_{\max}) \times (\bar{v}/\bar{v}_{\mathrm{front}})$, where $\bar{v}_{\mathrm{front}}$ was the mean speed of the slow vehicle driving ahead of the ego vehicle. This measure was chosen since, here, there was a clear motivation for the ego vehicle to overtake the slow-moving vehicle in front. With this measure, a driver model that simply follows the slow vehicle will achieve a fitness of (just) below one, whereas if the slow vehicle is overtaken, the fitness value will be larger than 1 (if collisions are avoided, of course). The variation of the fitness from three GA runs for this case is shown in Fig. 6. As in the previous case, all 500 scenarios were solved without collision. Also, no collisions occurred in a re-evaluation involving 500 new scenarios (not used during training). In all cases, the ego vehicle overtook the slower vehicle, resulting in fitness values above 1. This shows that the proposed method is able to solve other cases, with a slight modification of the simulation environment and the fitness measure.

As mentioned in Sect. IV, some duplicate rules and unreachable actions that cannot affect the result, see Table V, were typically present in the obtained models. These non-effective instructions, referred to as introns, often have positive

Fig. 5. Example of a traffic situation for an overtaking scenario with oncoming traffic, showing the situation after 10 seconds of driving. The ego vehicle (a truck-trailer combination) is shown in red, in the bottom lane. The arrows represent the velocities of the vehicles.

effect on the progress of the evolutionary search progress, see e.g. Ch. 7 in [13] for more details. Therefore, introns should not be removed from the chromosomes during the evolution, but instead, if desired, *after* that stage, when the generated final individual will be operating the vehicle.

As described in Sect. IV, the training was carried out in two stages: An initial stage (exemplified by Runs 1-3) aimed at finding a successful driver model, and a second stage (exemplified by Run 4) aimed at simplifying the model. Attempts were also made to carry out the entire training in a single run, with a length penalty already from the first instruction, but in those cases the optimization algorithm typically got stuck in a local optimum, and the corresponding driver model was then unable to complete all scenarios. Thus, the proposed two-stage optimization procedure brought clear benefits.

It is important to note that this method only solves the type of situations to which it is exposed in the simulations. Therefore, it is crucial to set up the simulations correctly, to ensure that they cover the intended case. Furthermore, as with many machine learning methods, it is hard to guarantee functional safety with a learned driving model. One way to deal with this, commonly proposed in literature, see e.g. [14], is to use an underlying safety layer, which verifies that a planned trajectory is safe before forwarding it to the vehicle control system.

For the highway case a test was made in which more complex rules were added, involving the speed of the surrounding vehicles. These would for example have the form *If the relative speed of the vehicle in the left lane is within the range 0 to 5 m/s, then ...* However, adding such rules did not improve the performance. On the contrary, with these rule types added, the optimization quickly got stuck in a local optimum, unable to handle more than a few scenarios.

Thus, in the end, in all cases considered here, the instruction encoding shown in Table III was used. This instruction set would probably be applicable in some other cases as well. However, a more general instruction set would be needed for some cases, e.g. cases involving crossings. One must then be careful to avoid the problems just described, where the optimization gets stuck in a local optimum. Some more techniques to avoid overfitting could possibly help in this regard. Defining more general instructions is, however, a topic for future work. If additional instructions, including more action types, are introduced, allowing more than one action to be executed at every time interval could be considered. This would also make it interesting to study alternative types of chromosome representations. Moreover, investigating further the effects of different choices of the fitness and evolutionary parameters, presented in Sect. III, is also a topic for future work.

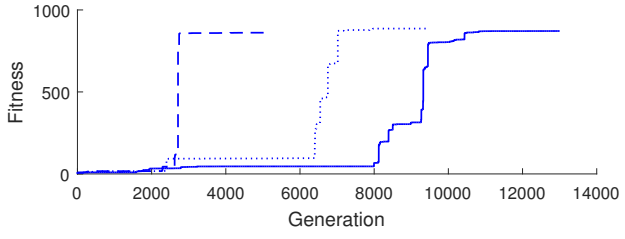


Fig. 6. Fitness variation of the best individual in the population, for three optimization runs with different random seeds.

VI. Conclusion

In conclusion, the main result of this paper is that the proposed method, involving a genetic algorithm with length-varying chromosomes, can automatically produce successful driver models in the form of a sequence of rules and actions. Here, a highway driving case was considered, with a truck-trailer combination as the ego vehicle. The resulting driver models match or surpass the performance of a reference model based on the IDM and the MOBIL model in terms of average speed. Furthermore the generality of the model has been demonstrated by applying it to a second case, in which oncoming traffic was considered. In both cases, the evolved driver model handled all scenarios without collision. Important topics for future work is to extend the model to include even more general instructions, and also to consider additional safety aspects.

Acknowledgment

The authors would like to thank Adj. Prof. Leo Laine for valuable comments on the manuscript. This work was partially supported by the Wallenberg Autonomous Systems and Software Program (WASP), and Vinnova FFI.

References

[1] K. Aghabayk *et al.*, "A state-of-the-art review of car-following models with particular considerations of heavy vehicles," *Transport Reviews*, vol. 35, no. 1, pp. 82–105, 2015.

[2] M. Treiber *et al.*, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, 2000.

[3] P. Gipps, "A model for the structure of lane-changing decisions," *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403 – 414, 1986.

[4] K. I. Ahmed, "Modeling drivers' acceleration and lane changing behavior," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[5] A. Kesting *et al.*, "General lane-changing model mobil for car-following models," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1999, pp. 86–94, 2007.

[6] J. Eggert and F. Damerow, "Complex lane change behavior in the foresighted driver model," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sept. 2015, pp. 1747–1754.

[7] S. Ulbrich and M. Maurer, "Towards tactical lane change behavior planning for automated vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 989–995.

[8] O. Benderius, "Modelling driver steering and neuromuscular behaviour," Ph.D. dissertation, Chalmers University of Technology, 2014.

[9] P. Nilsson *et al.*, "A driver model using optic information for longitudinal and lateral control of a long vehicle combination," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE, 2014, pp. 1456–1461.

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[11] D. D. Salvucci and R. Gray, "A two-point visual control model of steering," *Perception*, vol. 33, no. 10, pp. 1233–1248, 2004.

[12] M. F. Brameier and W. Banzhaf, *Linear genetic programming*. Springer Science & Business Media, 2007.

[13] W. Banzhaf *et al.*, *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[14] S. Underwood *et al.*, *Truck Automation: Testing and Trusting the Virtual Driver*. Springer International Publishing, 2016, pp. 91–109.

Paper II

Automated speed and lane change decision making using deep reinforcement learning

in

Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems, Maui, HI, USA, 2018, pp. 2148-2155.

Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning

Carl-Johan Hoel*†, Krister Wolff*, Leo Laine*†
*Chalmers University of Technology, 412 96 Göteborg, Sweden
†Volvo Group Trucks Technology, 405 08 Göteborg, Sweden
Email: {carl-johan.hoel, krister.wolff, leo.laine}@chalmers.se

***Abstract*—This paper introduces a method, based on deep reinforcement learning, for automatically generating a general purpose decision making function. A Deep Q-Network agent was trained in a simulated environment to handle speed and lane change decisions for a truck-trailer combination. In a highway driving case, it is shown that the method produced an agent that matched or surpassed the performance of a commonly used reference model. To demonstrate the generality of the method, the exact same algorithm was also tested by training it for an overtaking case on a road with oncoming traffic. Furthermore, a novel way of applying a convolutional neural network to high level input that represents interchangeable objects is also introduced.**

I. Introduction

By automating heavy vehicles, there is potential for a significant productivity increase, see e.g. [1]. One of the challenges in developing autonomous vehicles is that they need to make decisions in complex environments, ranging from highway driving to less structured areas inside cities. To predict all possible traffic situations, and code how to handle them, would be a time consuming and error prone work, if at all feasible. Therefore, a method that can learn a suitable behavior from its own experiences would be desirable. Ideally, such a method should be applicable to all possible environments. This paper introduces how a specific machine learning algorithm can be applied to automated driving, here tested on a highway driving case and an overtaking case.

Traditionally, rule based gap acceptance models are common to make lane changing decisions, see for example [2] or [3]. More recent methods often consider the utility of a potential lane change. Either the utility of changing to a specific lane is estimated, see [4] or [5], or the total utility (also called the expected return) over a time horizon is maximized by solving a partially observable Markov decisions process (POMDP), see [6] or [7]. Two commonly used models for speed control and to decide when to change lanes are the Intelligent driver model (IDM) [8] and the Minimize overall braking induced by lane changes (MOBIL) model [9]. The combination of these two models was used as a baseline when evaluating the method presented in this paper.

A common problem with most existing methods for autonomous driving is that they target one specific driving case. For example, the ones mentioned above are designed for highway driving, but if a different case is considered, such as driving on a road with oncoming traffic, a completely different method is required. In an attempt to overcome this issue, we introduced a more general approach in [10]. This method is based on a genetic algorithm, which is used to automatically train a general-purpose driver model that can handle different cases. However, the method still requires some features to be defined manually, in order to adapt its rules and actions to different driving cases.

During the last years, the field of deep learning has made revolutionary progress in many areas, see e.g. [11] or [12]. By combining deep neural networks with reinforcement learning, artificial intelligence has evolved in different domains, from playing Atari games [13], to continuous control [14], reaching a super human performance in the game of Go [15] and beating the best chess computers [16]. Deep reinforcement learning has also successfully been used for some special applications in the field of autonomous driving, see e.g. [17] and [18].

This paper introduces a method based on a Deep Q-Network (DQN) agent [13] that, from training in a simulated environment, automatically generates a decision making function. To the extent of the authors' knowledge, this method has not previously been applied to this problem. The main benefit of the presented method is that it is general, i.e. not limited to a specific driving case. For highway driving, it is shown that it can generate an agent that performs better than the combination of the IDM and MOBIL model. Furthermore, with no tuning, the same method can be applied to a different setting, in this case driving on a road with oncoming traffic. Two important differences compared to our previous approach in [10] is that the method presented in this paper does not need any hand crafted features and that the training is significantly faster. Moreover, this paper introduces a novel way of using a convolutional neural network architecture by applying it to high level sensor data, representing interchangeable objects, which improves and speeds up the learning process.

This paper is organized as follows: The DQN algorithm and how it was implemented is described in Sect. II. Next, Sect. III gives an overview of the IDM and the MOBIL model, and describes how the simulations were set up. In Sect. IV, the results are presented, followed by a discussion in Sect. V. Finally the conclusions are given in Sect. VI.

II. Speed and lane change decision making

In this paper, the task of deciding when to change lanes and to control the speed of the vehicle under consideration (henceforth referred to as the ego vehicle) is viewed as a reinforcement learning problem. A Deep Q-Network (DQN) agent [13] is used to learn the Q-function, which describes

how beneficial different actions are in a given state. The state of the surrounding vehicles and the available lanes are known to the agent, and its objective is to choose which action to take, which for example could be to change lanes, brake or accelerate. The details of the procedure are described in this section.

A. Reinforcement learning

Reinforcement learning is a branch of machine learning, where an agent acts in an environment and tries to learn a policy, π, that maximizes a cumulative reward function. The policy defines which action, a, to take, given a state, s. The state of the environment will then change to a new state, s', and return a reward, r. The reinforcement learning problem is often modeled as a Markov Decision Process (MDP), which is defined as the tuple $\langle S, A, T, R, \gamma \rangle$, where S is the set of states, A is the set of actions, $T : S \times A \to S$ is the state transition probability function, $R : S \times A \times S \to \mathbb{R}$ is the reward function and $\gamma \in [0, 1]$ is a discount factor. An MDP satisfies the Markov property, which means that the probability distribution of the future states depends only on the current state and action, and not on the history of previous states. At every time step, t, the goal of the agent is to maximize the future discounted return, defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{1}$$

where r_{t+k} is the reward given at step $t + k$. See [19] for a comprehensive introduction to reinforcement learning and MDPs.

B. Deep Q-Network

In the reinforcement learning algorithm called Q-learning [20], the agent tries to learn the optimal action value function, $Q^*(s, a)$. This function is defined as the maximum expected return when being in a state, s, taking some action, a, and then following the optimal policy, π^*. This is described by

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]. \tag{2}$$

The optimal action value function follows the Bellman equation, see [20],

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right], \tag{3}$$

which is based on the intuition that if the values of $Q^*(s', a')$ are known, the optimal policy is to select an action, a', that maximizes the expected value of $Q^*(s', a')$.

In the DQN algorithm [13], Q-learning is combined with deep learning. A deep neural network with weights θ is used as a function approximator of the optimal value function, i.e. $Q(s, a; \theta) \approx Q^*(s, a)$. The network is then trained by adjusting its parameters, θ_i, at every iteration, i, to minimize the error in the Bellman equation. This is typically done with stochastic gradient descent, where mini-batches with size M of experiences, described by the tuple $e_t = (s_t, a_t, r_t, s_{t+1})$, are drawn from an experience replay memory. The loss function at iteration i is defined as

$$L_i(\theta_i) = \mathbb{E}_{\mathrm{M}}\Big[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2\Big]. \tag{4}$$

Here, θ_i^- are the network parameters used to calculate the target at iteration i. In order to make the learning process more stable, these parameters are held fixed for a number of iterations and then periodically updated with the latest version of the trained parameters, θ_i. The trade off between exploration and exploitation is handled by following an ϵ-greedy policy. This means that a random action is selected with probability ϵ, and otherwise the action with the highest value is chosen. For further details on the DQN algorithm, see [13].

Q-learning and the DQN algorithm are known to overestimate the action value function under some conditions. A further development is the Double DQN algorithm [21], which aims to decouple the action selection and action evaluation. This is done by updating Eq. 4 to

$$L_i(\theta_i) = \mathbb{E}_{\mathrm{M}}\big[\big(r + \gamma Q(s', \arg\max_{a} Q(s', a; \theta_i); \theta_i^-) - Q(s, a; \theta_i)\big)^2\big]. \tag{5}$$

C. Agent implementation

The Double DQN algorithm, outlined above, was applied to control a vehicle in two test cases, which are further described in Sect. III-B. The details of the implementation of the agent are presented below.

1) MDP formulation: Since the intention of other road users cannot be observed, the speed and lane change decision making problem can be modeled as a partially observable Markov decision process (POMDP) [22]. To address the partial observability, the POMDP can be approximated by an MDP with a k-Markov approximation, where the state consists of the last k observations, $s_t = (o_{t-k+1}, o_{t-k+2}, \ldots, o_t)$ [13]. However, for the method presented in this paper, it proved sufficient to set $k = 1$, i.e. to simply use the last observation.

Two different agents were investigated in this study, called Agent1 and Agent2. They both used the same state input, s, defined as a vector with 27 elements, which contained information on the ego vehicle's speed, existing lanes and states of the 8 surrounding vehicles. Table I shows the configuration of the state (see Sect. III for details on how the traffic environment was simulated).

Agent1 only controlled the lane changing decisions, whereas the speed was automatically controlled by the IDM. This gave a direct comparison to the lane change decisions taken by the MOBIL model, in which the speed also was controlled by the IDM (see Sect. III-A for details). Agent2 controlled both the lane changing decisions and the speed. Here, the speed was changed by choosing between four different acceleration options: full brake (-9 m/s^2), medium brake (-2 m/s^2), maintain speed (0 m/s^2) and accelerate ($+2$ m/s^2). The action spaces of the two agents are given in Table II. When a decision to change lanes was taken, the intended lane of the lateral control model, described in Sect. III-B, was changed. Both agents took decisions at an interval of $\Delta t = 1$ s.

A simple reward function was used. Normally, at every time step, a positive reward was given, based on the distance driven during that interval, Δd, and normalized as $\Delta d / \Delta d_{\max}$. Here, $\Delta d_{\max} = \Delta t v_{\max}^{\mathrm{ego}}$, and $v_{\max}^{\mathrm{ego}}$ was the maximum possible

TABLE I

STATE INPUT VECTOR USED BY THE AGENTS. s_1, s_2 AND s_3 DESCRIBE THE STATE OF THE EGO VEHICLE AND THE AVAILABLE LANES, WHEREAS s_{3i+1}, s_{3i+2} AND s_{3i+3}, FOR i = 1,2,...8, REPRESENT THE STATE OF THE SURROUNDING VEHICLES.

s_1	Normalized ego vehicle speed, $v_{\text{ego}}/v_{\text{ego}}^{\max}$
s_2	$\begin{cases}1, & \text{if there is a lane to the left}\\ 0, & \text{otherwise}\end{cases}$
s_3	$\begin{cases}1, & \text{if there is a lane to the right}\\ 0, & \text{otherwise}\end{cases}$
s_{3i+1}	Normalized relative position of vehicle i, $\Delta s_i/\Delta s_{\max}$
s_{3i+2}	Normalized relative speed of vehicle i, $\Delta v_i/v_{\max}$
s_{3i+3}	$\begin{cases}-1, & \text{if vehicle } i \text{ is two lanes to the right of the ego vehicle}\\ -0.5, & \text{if vehicle } i \text{ is one lane to the right of the ego vehicle}\\ 0, & \text{if vehicle } i \text{ is in the same lane as the ego vehicle}\\ 0.5, & \text{if vehicle } i \text{ is one lane to the left of the ego vehicle}\\ 1, & \text{if vehicle } i \text{ is two lanes to the left of the ego vehicle}\end{cases}$

TABLE II

ACTION SPACES OF THE TWO AGENTS.

	Agent1
a_1	Stay in current lane
a_2	Change lanes to the left
a_3	Change lanes to the right
	Agent2
a_1	Stay in current lane, keep current speed
a_2	Stay in current lane, accelerate with -2 m/s^2
a_3	Stay in current lane, accelerate with -9 m/s^2
a_4	Stay in current lane, accelerate with 2 m/s^2
a_5	Change lanes to the left, keep current speed
a_6	Change lanes to the right, keep current speed

speed of the ego vehicle. This part of the reward function implicitly encouraged lane changes to overtake slower vehicles. However, if a collision occurred, or the ego vehicle drove out of the road (it could choose to change lanes to one that did not exist), a penalizing reward of -10 was given and the episode was terminated. If the ego vehicle ended up in a near collision, defined as being one vehicle length (4.8 m) from another vehicle, a reward of -10 was also given, but the episode was not terminated. Finally, to limit the number of lane changes, a reward of -1 was given when a lane changing action was chosen.

2) Neural network design: Two different neural network architectures were investigated in this study. Both had 27 input neurons, for the state described above. The final output layer had 3 output neurons for Agent1 and 6 output neurons for Agent2, where the value of neuron n_i represented the value function when choosing action a_i, i.e. $Q(s, a_i)$.

The first architecture was a standard fully connected neural network (FCNN), with two hidden layers. Each layer consisted of n_{hidden} neurons, set to 512, and rectified linear units (ReLUs) were used as activation functions [23]. The final output layer used a linear activation function.

The second architecture introduces a new way of applying temporal convolutional neural networks (CNNs). CNNs are inspired by the structure of the visual cortex in animals. By their architecture and weight sharing properties, they create a space and shift invariance, and reduce the number of parameters to be optimized. This has made them successful in

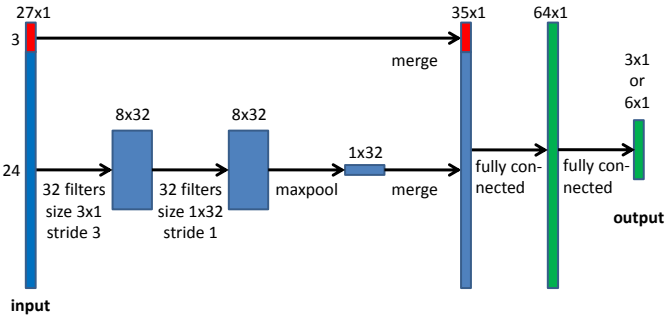


Fig. 1. The second network architecture, which used convolutional neural networks and max pooling to create translational invariance between the input from different surrounding vehicles. See the main text for further explanations.

the field of computer vision, where they have been applied directly to low level input, consisting of pixel values. For further details on CNNs, see e.g. [12].

In this study, a CNN architecture was applied to a high level input, which described the state of identical, interchangeable objects, see Fig. 1. Two convolutional layers were applied to the part of the state vector that represented the relative position, speed and lane of the surrounding vehicles. The first layer had n_{conv1} filters, set to 32, with filter size 3, stride 3 and ReLU activation functions. This structure created an output of 8×32 signals. Since there were 3 neighbouring input neurons that described the properties of each of the 8 surrounding vehicles, by setting the filter size and stride to 3, each row of the output only depended on one vehicle. The second layer had n_{conv2} filters, set to 32, with filter size 1, stride 1 and ReLU activation functions. This further aggregated knowledge about each vehicle in every row of the 8×32 output signal. After the second convolutional layer, a max pooling layer was added. This structure created a translational invariance of the input that described the relative state of the different vehicles, i.e. the result would be the same if e.g. the input describing vehicle 3 and vehicle 4 switched position in the input vector. This translational invariance, in combination with the reduced number of optimizable parameters, simplified and sped up the training of the network. See Sect. V for a further discussion on why a CNN architecture was beneficial in this setting.

The output of the max pooling layer was then concatenated with the rest of the input vector. A fully connected layer with n_{full} units, here set to 64, and ReLu activation functions followed. Finally, the output layer had 3 or 6 neurons, both with linear activation functions.

3) Training details: The network was trained by using the Double DQN algorithm, described in Sect. II-B. During training, the policy followed an ϵ-greedy behavior, where ϵ decreased linearly from ϵ_{start} to ϵ_{end} over $N_{\epsilon-\text{end}}$ iterations. A discount factor, γ, was used for future rewards. The target network was updated every N_{update} iterations by cloning the online parameters, i.e. setting $\theta_i^- = \theta_i$, at the updating step. Learning started after N_{start} iterations and a replay memory of size M_{replay} was used. Mini-batches of training samples with size M_{mini} were uniformly drawn from the replay memory and the network was updated using the RMSProp algorithm [24], with a learning rate of η. In order to improve the stability, error clipping was used by limiting the error term

TABLE III
HYPERPARAMETERS USED TO TRAIN THE DQN AGENTS.

Discount factor, γ	0.99
Learning start iteration, N_{start}	50,000
Replay memory size, M_{replay}	500,000
Initial exploration constant, ϵ_{start}	1
Final exploration constant, ϵ_{end}	0.1
Final exploration iteration, $N_{\epsilon\text{-end}}$	500,000
Learning rate, η	0.00025
Mini-batch size, M_{mini}	32
Target network update frequency, N_{update}	30,000

$r + \gamma Q(s', \arg\max_a Q(s', a; \theta_i); \theta_i^-) - Q(s, a; \theta_i)$ to $[-1, 1]$.

The hyperparameters of the training are summarized in Table III. Due to the computational complexity, a systematic grid search was not performed. Instead, the hyperparameter values were selected from an informal search, based upon the values given in [13] and [21].

The state space, described above, did not provide any information on where in an episode the agent was at a given time step, e.g. if it was in the beginning or close to the end (Sect. III-B describes how an episode was defined). The reason for this choice was that the goal was to train an agent that performed well in highway driving of infinite length. Therefore, the longitudinal position was irrelevant. However, at the end of a successful episode, the future discounted return, R_{end}, was 0. To avoid that the agent learned this, the last experience e_{end} was not stored in the experience replay memory. Thereby, the agent was tricked to believe that the episode continued forever.

III. SIMULATION SETUP

A highway case was used as the main way to test the algorithm outlined above. To evaluate the performance of the agent, a reference model, consisting of the IDM and MOBIL model, was used. This section briefly summarizes the reference model, describes how the simulations were set up and how the performance was measured. Moreover, in order to show the versatility of the proposed method, it was further tested in a secondary overtaking case with oncoming traffic, which is also described here.

A. Reference model

The IDM [8] is widely used in transportation research to model the longitudinal dynamics of a vehicle. With this model, the speed of the ego vehicle, v, varies according to

$$\dot{v} = a\left(1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{d^*(v, \Delta v)}{d}\right)^2\right), \tag{6}$$

$$d^*(v, \Delta v) = d_0 + vT + v\Delta v/(2\sqrt{ab}). \tag{7}$$

The vehicle's speed depends on the distance to the vehicle in front, d, and the speed difference (approach rate), Δv. Table IV shows the parameters that are used to tune the model. The values were taken from the original paper [8].

The MOBIL model [9] makes decisions on when to change lanes by maximizing the acceleration of the vehicle in consideration and the surrounding vehicles. For a lane change to be allowed, the induced acceleration of the following car in the new lane, a_{n}, must fulfill a safety criterion, $a_{\text{n}} > -b_{\text{safe}}$. To predict the acceleration of the ego and surrounding vehicles, the IDM model is used. If the safety criterion is met, MOBIL changes lanes if

$$\tilde{a}_{\text{e}} - a_{\text{e}} + p\left((\tilde{a}_{\text{n}} - a_{\text{n}}) + (\tilde{a}_{\text{o}} - a_{\text{o}})\right) > a_{\text{th}}, \tag{8}$$

where a_{e}, a_{n} and a_{o} are the accelerations of the ego vehicle, the trailing vehicle in the target lane, and the trailing vehicle in the current lane, respectively, assuming that the ego vehicle stays in its lane. Furthermore, $\tilde{a}_{\text{e}}$, $\tilde{a}_{\text{n}}$ and $\tilde{a}_{\text{o}}$ are the corresponding accelerations if the lane change is carried out. The politeness factor, p, controls how the effect on other vehicles is valued. To perform a lane change, the collective acceleration gain must be higher than a threshold, Δa_{th}. If there are lanes available both to the left and to the right, the same criterion is applied to both options. If both criteria are fulfilled, the option with the highest acceleration gain is chosen. The parameter values of the MOBIL model are shown in Table IV. They were taken from the original paper [9], except for the politeness factor, here set to 0. This setting provided a more fair comparison to the DQN agent, since then neither method considered possible acceleration losses of the surrounding vehicles.

TABLE IV
IDM AND MOBIL MODEL PARAMETERS.

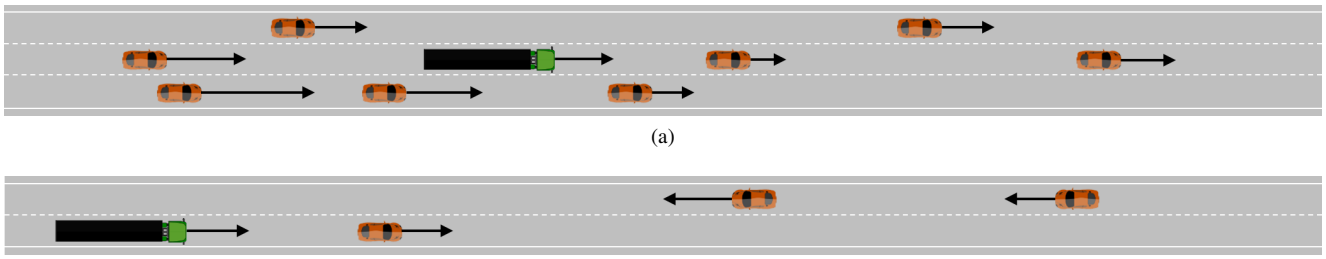
Minimum gap distance, d_0	2 m
Safe time headway, T	1.6 s
Maximal acceleration, a	0.7 m/s^2
Desired deceleration, b	1.7 m/s^2
Acceleration exponent, δ	4
Politeness factor, p	0
Changing threshold, a_{th}	0.1 m/s^2
Maximum safe deceleration, b_{safe}	4 m/s^2

B. Traffic simulation

1) Highway case: A highway case was used as the main way to test the method presented in this paper. This case was similar to the one used in the previous study [10]. For completeness, it is summarized below.

A three-lane highway was used, where the ego vehicle to be controlled was surrounded by 8 other vehicles. The ego vehicle consisted of a 16.5 m long truck-semitrailer combination and the surrounding vehicles were normal 4.8 m long passenger cars. These surrounding vehicles stayed in their initial lanes and followed the IDM model longitudinally. Overtaking was allowed both on the left and the right side of another vehicle. An example of an initial traffic situation is shown in Fig. 2a.

Although normal highway driving mostly consists of traffic with rather constant speeds and small accelerations, occasionally vehicles brake hard, or even at the maximum of their capability to avoid collisions. Drivers can also decide to suddenly increase their speed rapidly. Therefore, in order for the agent to learn to keep a safe inter-vehicle distance, such quick speed changes need to be included in the training process. The surrounding vehicles in the simulations were assigned different desired speed trajectories. To speed up the



(a)

(b)

Fig. 2. (a) Example of an initial traffic situation for the highway case, which was used as the main way to test the algorithm. (b) Example of a traffic situation for a secondary overtaking case with oncoming traffic, showing the situation 10 seconds from the initial state. In both cases, the ego vehicle (truck-trailer combination) is shown in green and black. The arrows represent the velocities of the vehicles.

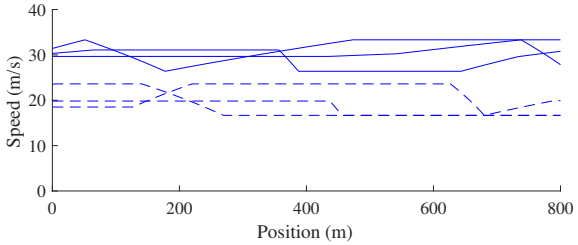


Fig. 3. Example of six different randomly generated speed trajectories, defined for different positions along the highway. The solid lines are fast trajectories, applied to vehicles starting behind the ego vehicle, whereas the dashed lines are slow trajectories, applied to vehicles starting in front of the ego vehicle.

TABLE V
PARAMETERS OF THE SIMULATED HIGHWAY CASE.

Maximum initial vehicle spread, d_{long}	200 m
Minimum initial inter-vehicle distance, d_{Δ}	25 m
Front vehicle minimum speed, v^{+}_{min}	16.7 m/s (60 km/h)
Front vehicle maximum speed, v^{+}_{max}	23.6 m/s (85 km/h)
Rear vehicle minimum speed, v^{-}_{min}	26.4 m/s (95 km/h)
Rear vehicle maximum speed, v^{-}_{max}	33.3 m/s (120 km/h)
Initial ego vehicle speed, $v^{\text{ego}}_{\text{init}}$	25 m/s (90 km/h)
Maximum ego vehicle speed, $v^{\text{ego}}_{\text{max}}$	25 m/s (90 km/h)
Episode length, d_{max}	800 m

training of the agent, these trajectories contained frequent speed changes, which occurred more often than during normal highway driving. Some examples are shown in Fig. 3.

The ego vehicle initially started in the middle lane, surrounded by 8 other vehicles. These were randomly positioned in the lanes, within d_{long} longitudinally and with a minimum inter-vehicle distance d_{Δ}. The initial and maximum ego vehicle speed was $v^{\text{ego}}_{\text{init}}$ and $v^{\text{ego}}_{\text{max}}$ respectively. Vehicles that were positioned in front of the ego vehicle were assigned slower speed trajectories, in the range $[v^{+}_{\text{min}}, v^{+}_{\text{max}}]$, whereas vehicles placed behind the ego vehicle were assigned faster speed trajectories, in the range $[v^{-}_{\text{min}}, v^{-}_{\text{max}}]$. This created traffic situations where the agent needed to make lane changes to overtake slow vehicles, and at the same time consider faster vehicles approaching from behind. Episodes where two vehicles were placed too close together with a large speed difference, thus causing an unavoidable collision, were deleted. Each episode was d_{max} long. The values of the mentioned parameters are presented in Table V. Further details on the setup of the simulations, and how the speed trajectories were generated, are described in [10].

2) Overtaking case: In order to illustrate the generality of the method presented in this paper, a secondary overtaking case, including two-way traffic, was also tested. Fig. 2b shows an example of this case. The ego vehicle started in the right lane, with an initial speed of $v^{\text{ego}}_{\text{init}}$, set to 25 m/s. Another vehicle, which followed a random slow speed profile (defined above), was placed 50 m in front of the ego vehicle. Two oncoming vehicles, also following slow speed profiles, were placed in the left, oncoming lane, at a random distance between 300 and 1100 m in front of the ego vehicle.

3) Vehicle motion and lateral control models: In both the highway and the overtaking case, the motion of the vehicles was simulated by using kinematic models. A lane following two-point visual control model [25] was used to control the vehicles laterally. As mentioned in Sect. II-C, when the agent decided to change lanes, the setpoint of this model was changed to the new desired lane. The same procedure was used if the MOBIL model decided to change lanes. With this control model, a lane change normally took 2 to 3 s, depending on the longitudinal speed. See [10] for further details on the vehicle motion and lateral control models.

C. Performance index

In order to evaluate how the DQN agent performed compared to the reference driver model (presented in Sect. III-A) in a specific episode of the highway case, a performance index, $\tilde{p}$, was defined as

$$\tilde{p} = (d/d_{\text{max}})(\bar{v}/\bar{v}_{\text{ref}}). \tag{9}$$

Here, d is the distance driven by the ego vehicle (limited by a collision or the episode length), d_{max} is the episode length, $\bar{v}$ is the average speed of the ego vehicle and $\bar{v}_{\text{ref}}$ is the average speed when the reference model controlled the ego vehicle through the episode. With this definition, the distance driven by the ego vehicle was the dominant limiting factor when a collision occurred. However, if the agent managed to complete the episode without collisions, the average speed determined the performance index. A value larger than 1 means that the agent performed better than the reference model.

For the overtaking case, the reference model described above cannot be used. Instead, the performance index was simply defines as $\tilde{p}_{\text{o}} = (d/d_{\text{max}})(\bar{v}/\bar{v}_{\text{refIDM}})$. Here, $\bar{v}_{\text{refIDM}}$ was the mean speed of the ego vehicle when it was controlled by the IDM through the same episode, i.e. when it did not overtake the preceding vehicle.

TABLE VI
SUMMARY OF THE RESULTS OF THE DIFFERENT AGENTS FOR THE HIGHWAY CASE AND THE OVERTAKING CASE.

	Highway case		Overtaking case	
	Collision free episodes	Performance index, $\tilde{p}$	Collision free episodes	Performance index, $\tilde{p}_o$
$\text{Agent1}_{\text{CNN}}$	100%	1.01	100%	1.06
$\text{Agent2}_{\text{CNN}}$	100%	1.10	100%	1.11
$\text{Agent1}_{\text{FCNN}}$	98%	0.98	-	-
$\text{Agent2}_{\text{FCNN}}$	86%	0.96	-	-

IV. RESULTS

This section focuses on the results that were obtained for the highway case, described in Sect. III-B, which was the main way of testing the presented method. It also briefly explains and discusses some characteristics of the results, whereas a more general discussion follows in Sect. V. The results regarding the overtaking case are collected in Sect. IV-C.

As described in Sect. II, two agents with different action spaces were investigated. Agent1 only decided when to change lanes, whereas Agent2 decided both the speed and when to change lanes. Furthermore, two different neural network architectures were used. In summary, the four variants were $\text{Agent1}_{\text{FCNN}}$, $\text{Agent1}_{\text{CNN}}$, $\text{Agent2}_{\text{FCNN}}$ and $\text{Agent2}_{\text{CNN}}$.

Five different runs were carried out for the four agent variants, where each run had different random seeds for the DQN and the traffic simulation. The networks were trained for 2 million iterations (3 million for $\text{Agent2}_{\text{FCNN}}$), and at every 50,000 iterations, they were evaluated over 1,000 random episodes. Note that these evaluation episodes were randomly generated, and not presented to the agents during training. During the evaluation runs, the performance index described in Sect. III-C was used to compare the agents' and the reference model's behaviour. The results are shown in Fig. 4, which presents the average proportion, $\hat{p}$, of successfully completed, i.e. collision free, evaluation episodes of the four agent variants, and in Fig. 5, which shows their average performance index, $\tilde{p}$. The final performance of the fully trained agents is summarized in Table VI.

A. Agents using a CNN

In Fig. 4, it can be seen that $\text{Agent1}_{\text{CNN}}$ solved all the episodes already after 100,000 iterations, which is the first evaluation after that the training started at 50,000 iterations. At this point it had learned to always stay in its lane, in order to avoid collisions. Since it often got blocked by slower vehicles, its average performance index was therefore lower than 1 at this point, see Fig. 5. However, after around 600,000 iterations, $\text{Agent1}_{\text{CNN}}$ had learned to carry out lane changes when necessary, and performed similar to the reference model.

Fig. 4 shows that $\text{Agent2}_{\text{CNN}}$ quickly figured out how to change lanes and increase its speed to solve most of the episodes. Its performance index was on par with the reference model (reached 1) early on during the training, at around 250,000 iterations, see Fig. 5. Then, at 400,000 iterations, it solved all the evaluation episodes without collisions. With more training, there were still no collisions, but the performance index increased and stabilized at 1.1.

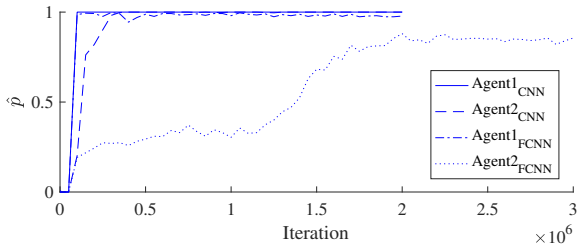


Fig. 4. Proportion of episodes solved without collisions by the different agents during training.

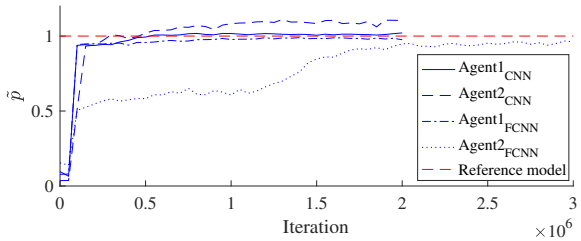


Fig. 5. Performance index of the different agents during training.

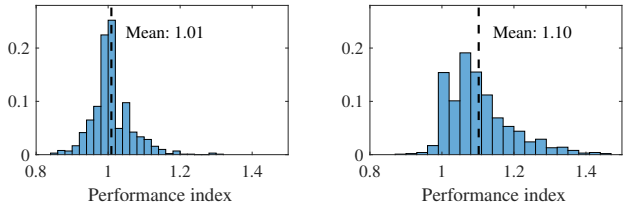


Fig. 6. Histogram of the performance index at the end of the training for $\text{Agent1}_{\text{CNN}}$ (left) and $\text{Agent2}_{\text{CNN}}$ (right).

Fig. 6 shows a histogram of the performance index for 1,000 evaluation episodes, which were run by the final trained version of $\text{Agent1}_{\text{CNN}}$ and $\text{Agent2}_{\text{CNN}}$. Since all the episodes were completed without collisions, the performance index was simply the speed ratio $\bar{v}/\bar{v}_{\text{ref}}$. In the figure, it can be seen that most often there was a small difference between the average speed of the agents and the reference model. There were also some outliers, which were both faster and slower than the reference model. The explanation for these is that the episodes were randomly generated, which meant that even a reasonable action could get the ego vehicle into a situation where it got locked in and could not overtake the surrounding vehicles. Therefore, a small difference in behaviour could lead to such situations for both the trained agents and the reference model, which explains the outliers. Furthermore, the peak at index 1 for $\text{Agent2}_{\text{CNN}}$ is explained by that there were some episodes when the lane in front of the ego vehicle was free from the start. Then both the reference model and the agents drove at the maximum speed through the whole episode.

To further illustrate the properties of the agents, and how they developed during training, the percentage of chosen actions is shown in Fig. 7. For $\text{Agent1}_{\text{CNN}}$, it can be seen that it quickly figured out that changing lanes can lead to collisions, and therefore it chose to stay in its lane almost 100% of the time in the beginning. This explains why it completed all the episodes already from the first evaluation point after its training started. However, as training proceeded, it figured out when it safely could change lanes, and thereby perform

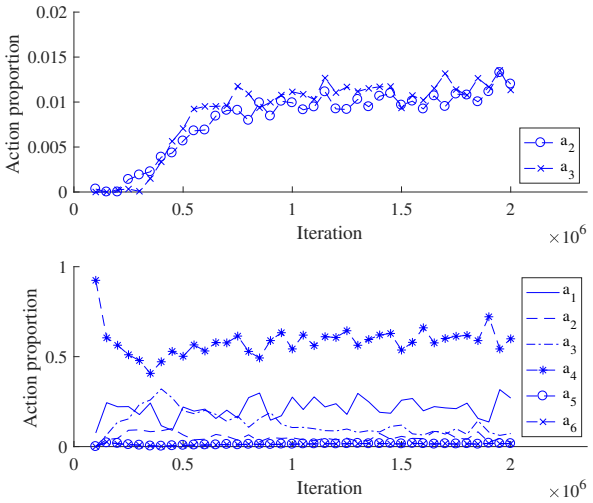


Fig. 7. Top: proportion of actions chosen by $\text{Agent1}_{\text{CNN}}$ during training. Due to the scale difference, a_1, i.e. stay in the current lane, is here left out. Bottom: proportion of actions chosen by $\text{Agent2}_{\text{CNN}}$ during training. Both plots start at 100,000 iterations, since that is the first evaluation point after that training started at 50,000 iterations.

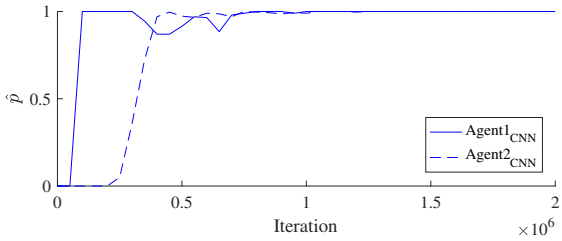


Fig. 8. Proportion of overtaking episodes solved without collisions by the different agents during training.

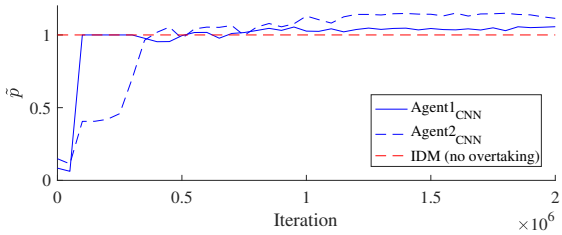


Fig. 9. Performance index of the different agents during training on the overtaking case.

better. At the end of its training, it chose to change lanes around 1% of the time. $\text{Agent2}_{\text{CNN}}$ first learned a short sighted strategy, where it accelerated most of the time to obtain a high immediate reward. This naturally led to many rear end collisions. However, when its training proceeded, it learned to control its speed by braking or idling, and to change lanes when necessary. Reassuringly, both agents learned to change lanes to the left and right equally often.

B. Agents using a FCNN

Both $\text{Agent1}_{\text{FCNN}}$ and $\text{Agent2}_{\text{FCNN}}$ failed to complete all the evaluation episodes without collisions, see Fig. 4 and Table VI. Naturally, $\text{Agent1}_{\text{FCNN}}$ solved a significantly higher fraction of the episodes and performed better than $\text{Agent2}_{\text{FCNN}}$, since it only needed to decide when to change lanes, and not control the speed. In the beginning, it learned to always stay in its lane, and thereby solved all episodes without collisions, but reached a lower performance index than the reference model, see Fig. 5. With more training, it started to change lanes and performed reasonably well, but sometimes caused collisions. $\text{Agent2}_{\text{FCNN}}$ performed significantly worse and collided in 14% of the episodes by the end of its training. A longer training run was carried out for $\text{Agent1}_{\text{FCNN}}$ and $\text{Agent2}_{\text{FCNN}}$, but after 20 million iterations, the results were the same.

C. Overtaking case

In order to demonstrate the generality of the method presented in this paper, the same algorithm was applied to an overtaking situation, described in Sect. III-B. Fig. 8, Fig. 9 and Table VI show the proportion of successfully completed evaluation episodes, $\hat{p}$, and the modified performance index, $\tilde{p}_{\text{o}}$, of $\text{Agent1}_{\text{CNN}}$ and $\text{Agent2}_{\text{CNN}}$. By the end of the training, both agents solved all episodes without collisions. Furthermore, in all the episodes, the ego vehicle overtook the slower vehicle, resulting in performance indexes above 1.

V. Discussion

In Table VI, it can be seen that both Agent1 and Agent2 with the convolutional neural network architecture solved all the episodes without collisions. The performance of $\text{Agent1}_{\text{CNN}}$ was on par with the reference model. Since they both used the IDM to control the speed, this result indicates that the trained agent and the MOBIL model took lane changing decisions with similar quality. However, when adding the possibility for the agent to also control its speed, as in $\text{Agent2}_{\text{CNN}}$, the trained agent had the freedom to find better strategies and could therefore outperform the reference model. This result illustrates that for a better performance, lateral and longitudinal decisions should not be completely separated.

As expected, using a CNN architecture resulted in a significantly better performance than a FCNN architecture, see e.g. Table VI. The reason for this is, as mentioned in Sect. II-C, that the CNN architecture creates a translational invariance of the input that describes the relative state of the different vehicles. This is reasonable, since it is desirable that the agent reacts the same way to other vehicles' behaviour, independently of where they are positioned in the input vector. Furthermore, since CNNs share weights, the complexity of the network is reduced, which in itself speeds up the learning process. This way of using CNNs can be compared to how they previously were introduced and applied to low level input, often on pixels in an image, where they provide a spatial invariance when identifying features, see e.g. [26]. The results of this paper show that it can also be beneficial to apply CNNs to high level input of interchangeable objects, such as the state description shown in Sect. II-C.

As mentioned in Sect. II-C, a simple reward function was used. Naturally, the choice of reward function strongly affects the resulting behaviour. For example, when no penalty was given for a lane change, the agent found solutions where it constantly demanded lane changes in opposite directions, which made the vehicle drive in between two lanes. In this study, a simple reward function worked well, but for other

cases a more careful design may be required. One way to determine a reward function that mimics human preferences is to use inverse reinforcement learning [27].

In a previous paper, [10], we presented a different method, based on a genetic algorithm, that automatically can generate a driving model for similar cases as described here. That method is also general and it was shown that it is applicable to different cases, but it requires some hand crafted features when designing the structure of its rules. However, the method presented in this paper requires no such hand crafted features, and instead uses the measured state, described in Table I, directly as input. Furthermore, the method in [10] achieved a similar performance when it comes to safety and average speed, but the number of necessary training episodes was between one and two orders of magnitude higher than for the method that was investigated in this study. Therefore, the new method is clearly advantageous compared to the previous one.

An important remark is that when training an agent by using the method presented in this paper, the agent will only be able to solve the type of situations that it is exposed to in the simulations. It is therefore important that the design of the simulated traffic environment covers the intended case. Furthermore, when using machine learning to produce a decision making function, it is hard to guarantee functional safety. Therefore, it is common to use an underlying safety layer, which verifies the safety of a planned trajectory before it is executed by the vehicle control system, see e.g. [28].

VI. Conclusion and future work

The main results of this paper show that a Deep Q-Network agent can be trained to make decisions in autonomous driving, without the need of any hand crafted features. In a highway case, the DQN agents performed on par with, or better than, a reference model based on the IDM and MOBIL model. Furthermore, the generality of the method was demonstrated by applying it to a case with oncoming traffic. In both cases, the trained agents handled all episodes without collisions. Another important conclusion is that, for the presented method, applying a CNN to high level input that represents interchangeable objects can both speed up the learning process and increase the performance of the trained agent.

Topics for future work include to further analyze the generality of this method by applying it to other cases, such as crossings and roundabouts, and to systematically investigate the impact of different parameters and network architectures. Moreover, it would be interesting to apply prioritized experience replay [29], which is a method where important experiences are repeated more frequently during the training process. This could potentially improve and speed up the learning process.

Acknowledgment

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP), funded by Knut and Alice Wallenberg Foundation, and partially by Vinnova FFI.

References

[1] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167 – 181, 2015.

[2] P. Gipps, "A model for the structure of lane-changing decisions," *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403 – 414, 1986.

[3] K. I. Ahmed, "Modeling drivers' acceleration and lane changing behavior," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[4] J. Eggert and F. Damerow, "Complex lane change behavior in the foresighted driver model," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1747–1754.

[5] J. Nilsson *et al.*, "If, when, and how to perform lane change maneuvers on highways," *IEEE Intelligent Transportation Systems Magazine*, vol. 8, no. 4, pp. 68–78, 2016.

[6] S. Ulbrich and M. Maurer, "Towards tactical lane change behavior planning for automated vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 989–995.

[7] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, "The value of inferring the internal state of traffic participants for autonomous freeway driving," in *2017 American Control Conference (ACC)*, 2017, pp. 3004–3010.

[8] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, 2000.

[9] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model mobil for car-following models," *Transportation Research Record*, vol. 1999, pp. 86–94, 2007.

[10] C. J. Hoel, M. Wahde, and K. Wolff, "An evolutionary approach to general-purpose automated speed and lane change behavior," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 743–748.

[11] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[12] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[14] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[15] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[16] D. Silver *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[17] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *CoRR*, vol. abs/1610.03295, 2016.

[18] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[19] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1998.

[20] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[21] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 807–814.

[24] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," Coursera: Neural Networks for Machine Learning, 2012.

[25] D. D. Salvucci and R. Gray, "A two-point visual control model of steering," *Perception*, vol. 33, no. 10, pp. 1233–1248, 2004.

[26] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[27] S. Zhifei and E. M. Joo, "A review of inverse reinforcement learning theory and recent advances," in *2012 IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8.

[28] S. Underwood *et al.*, *Truck Automation: Testing and Trusting the Virtual Driver*. Springer International Publishing, 2016, pp. 91–109.

[29] T. Schaul *et al.*, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2015.

Paper III

Combining planning and deep reinforcement learning in tactical decision making for autonomous driving

in

IEEE Transactions on Intelligent Vehicles, vol. 5, no. 2, pp. 294-305, 2020.

Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving

Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J. Kochenderfer

***Abstract*—Tactical decision making for autonomous driving is challenging due to the diversity of environments, the uncertainty in the sensor information, and the complex interaction with other road users. This paper introduces a general framework for tactical decision making, which combines the concepts of planning and learning, in the form of Monte Carlo tree search and deep reinforcement learning. The method is based on the AlphaGo Zero algorithm, which is extended to a domain with a continuous state space where self-play cannot be used. The framework is applied to two different highway driving cases in a simulated environment and it is shown to perform better than a commonly used baseline method. The strength of combining planning and learning is also illustrated by a comparison to using the Monte Carlo tree search or the neural network policy separately.**

***Index Terms*—Autonomous driving, tactical decision making, reinforcement learning, Monte Carlo tree search.**

I. Introduction

AUTONOMOUS vehicles are expected to bring many societal benefits, such as increased productivity, a reduction of accidents, and better energy efficiency [1]. One of the technical challenges for autonomous driving is to be able to make safe and effective decisions in diverse and complex environments, based on uncertain sensor information, while interacting with other traffic participants. A decision making method should therefore be sufficiently general to handle the spectrum of relevant environments and situations. The naive approach of trying to anticipate all possible traffic situations and manually code a suitable behavior for these is infeasible. The topic of this paper is tactical decision making, which considers high level decisions that adapts the behavior of the vehicle to the current traffic situation [2], [3]. For example, these decisions could handle when to change lanes, or whether or not to stop at an intersection.

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems, and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, and partially by Vinnova FFI. *(Corresponding author: Carl-Johan Hoel.)*

C. J. Hoel and L. Laine are with the Department of Vehicle Automation, Volvo Group, 40508 Gothenburg, Sweden, and with the Department of Mechanics and Maritime Sciences, Chalmers University of Technology, 41296 Gothenburg, Sweden (e-mail: {carl-johan.hoel, leo.laine}@volvo.com).

K. Driggs-Campbell is with the Electrical and Computer Engineering Department, University of Illinois at Urbana-Champaign, IL 61801, USA (e-mail: krdc@illinois.edu)

K. Wolff is with the Department of Mechanics and Maritime Sciences, Chalmers University of Technology, 41296 Gothenburg, Sweden (e-mail: krister.wolff@chalmers.se)

M. J. Kochenderfer is with the Aeronautics and Astronautics Department at Stanford University, Stanford, CA 94305, USA (e-mail: mykel@standford.edu).

Rule-based methods, implemented as handcrafted state machines, were successfully used during the DARPA Urban Challenge [4], [5], [6]. However, a drawback with these rule-based approaches is that they lack the ability to generalize to unknown situations, which makes it hard to scale them to the complexity of real-world driving. Another approach is to treat the decision making task as a motion planning problem, which has been applied to highway [7], [8] and intersection scenarios [9]. While successful for some situations, the sequential design of this method, which first predicts the trajectory of the surrounding vehicles and then plan the ego vehicle trajectory accordingly, results in a reactive behavior which does not consider interaction during the trajectory planning.

It is common to model the tactical decision making task as a partially observable Markov decision process (POMDP) [10]. This mathematical framework models uncertainty in the current state, future evolution of the traffic scene, and interactive behavior. The task of finding the optimal policy for a POMDP is difficult, but many approximate methods exist. Offline methods can solve complex situations and precomputes a policy before execution, which for example has been done for an intersection scenario [11], [12]. However, due to the large number of possible real world scenarios, it becomes impossible to precalculate a policy that is generally valid. Online methods compute a policy during execution, which makes them more versatile than offline methods, but the limited computational resources reduces the solution quality. Ulbrich et al. considered a lane changing scenario on a highway, where they introduced a problem-specific high level state space that allowed an exhaustive search to be performed [13]. Another online method for solving a POMDP is Monte Carlo tree search (MCTS) [14], which has been applied to lane changes on a highway [15]. Hybrid approaches between offline and online planning have also been studied [16].

A fundamentally different approach to decision making problems is to learn a suitable behavior from data. Reinforcement learning (RL) [17] has proved successful in various domains, such as playing Atari games [18], in continuous control [19], and reaching a super human performance in the game of Go [20]. In a previous paper, we showed how a Deep Q-Network (DQN) agent could learn to make tactical decisions in two different highway scenarios [21]. A similar approach, but applied to an intersection scenario, was presented by Tram et al. [22]. Expert knowledge can be used to restrict certain actions, which has been shown to speed up the training process for a lane changing scenario [23]. A different approach, which

uses a policy gradient method to learn a desired behavior, has been applied to a merging scenario on a highway [24]. A common drawback with RL methods is that they require many training samples to reach convergence. RL methods may also suffer from the credit assignment problem, which makes it hard to learn long temporal correlations between decisions and the overall performance of the agent [17].

This paper presents a general framework, based on the AlphaGo Zero algorithm [20], that combines the concepts of planning and learning to create a tactical decision making agent for autonomous driving (Sect. III). The planning is done with a variation of Monte Carlo tree search, which constructs a search tree based on random sampling. The difference between standard MCTS and the version used here is that a neural network biases the sampling towards the most relevant parts of the search tree. The neural network is trained by a reinforcement learning algorithm, where the MCTS component both reduces the required number of training samples and aids in finding long temporal correlations. The presented framework is applied to two conceptually different driving cases (Sect. IV), and performs better than a commonly used baseline method (Sect. V). To illustrate the strength of combining planning and learning, it is compared to using the planning or the learned policy separately.

In contrast to the related work, the approach that is introduced in this paper combines the properties of planning and RL. When used online, the planning can be interrupted anytime with a reasonable decision, even after just one iteration, which will then return the learned action. More computational time will improve the result. The proposed approach is general and can be adapted to different driving scenarios. Expert knowledge is used to ensure safety by restricting actions that lead to collisions. The intentions of other vehicles are considered when predicting the future and the algorithm operates on a continuous state space. The AlphaGo Zero algorithm is here extended beyond the zero-sum board game domain of Go, to a domain with a continuous state space, a not directly observable state, and where self-play cannot be used. Further properties of the framework are discussed in Sect. VI.

The main contributions of this paper are:

- The extension of the AlphaGo Zero algorithm, which allows it to be used in the autonomous driving domain.
- The introduction of a general tactical decision making framework for autonomous driving, based on this extended algorithm.
- The performance analysis of the introduced tactical decision making framework, applied to two different test cases.

II. Theoretical Background

This section introduces partially observable Markov decision processes and two solution methods: Monte Carlo tree search, and reinforcement learning.

A. Partially Observable Markov Decision Process

A POMDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{O}$ is the observation space, T is a state transition model, O is an observation model, R is a reward model, and γ is a discount factor. At every time step t, the goal of the agent is to maximize the future discounted return, defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{1}$$

where r_{t+k} is the reward at step $t+k$ [10].

The state transition model $T(s' \mid s, a)$ describes the probability that the system transitions to state s' from state s when action a is taken. The observation model $O(o \mid s, a, s')$ is the probability of observing o in state s', after taking action a in state s. For many real world problems, it is not feasible to represent the probability distributions T and O explicitly. However, some solution approaches only require samples and use a generative model G instead, which generates a new sampled state and observation from a given state and action, i.e., $(s', o) \sim G(s, a)$. The reward model defines the reward of each step as $r = R(s, a, s')$ [10].

Since the agent cannot directly observe the state s of the environment, it can maintain a belief state b, where a probability $b(s)$ is assigned to being in state s. In simple cases, the belief can be exactly calculated using Bayes' rule. However, approximate methods, such as particle filtering, are often used in practice [25].

B. Monte Carlo Tree Search

MCTS can be used to select approximately optimal actions in POMDPs [14]. It constructs a search tree that consists of alternating layers of state nodes and action nodes, in order to estimate the state-action value function $Q(s, a)$, which describes the expected return R_t when taking action a in state s and then following the given policy. A generative model G is used to traverse the tree to the next state, given a state and an action. In the upper confidence tree version, the tree is expanded by selecting the action node that maximizes the upper confidence bound (UCB), defined as

$$UCB(s, a) = Q(s, a) + c_{\text{uct}} \sqrt{\frac{\ln N(s)}{N(s, a)}}, \tag{2}$$

where $Q(s, a)$ is the current estimate of the state-action value function, $N(s, a)$ is the number of times action a has been tried from state s, $N(s) = \sum_{a \in \mathcal{A}} N(s, a)$, and c_{uct} is a parameter that controls the amount of exploration in the tree.

The standard formulation of MCTS cannot handle problems with a continuous state space, since then the same state may never be sampled more than once from G, which would result in a wide tree with just one layer. One way to address this problem is to use progressive widening [26]. With this technique, the number of children of a state-action node is limited to $kN(s, a)^{\alpha}$, where k and α are tuning parameters. When there are less children than the limit, a new state is added by sampling from G. Otherwise, a previous child is randomly chosen. Thereby the number of children gradually grows as $N(s, a)$ increases.

C. Reinforcement Learning

Reinforcement learning is a branch of machine learning, where an agent tries to learn a policy π that maximizes the expected future return $\mathbb{E}(R_t)$ in some environment [17]. The policy defines which action a to take in a given state s. In the RL setting, the state transition model T (or G) of the POMDP may not be known. Instead, the agent gradually learns by taking actions in the environment and observing what happens, i.e., collecting experiences (s, a, s', r).

III. Tactical Decision Making Framework

This paper introduces a framework that combines planning and learning for tactical decision making in the autonomous driving domain. With this approach, a neural network is trained to guide MCTS to the relevant regions of the search tree, and at the same time, MCTS is used to improve the training process of the neural network. This idea is based on the AlphaGo Zero algorithm, originally developed for the game of Go [20]. However, such a zero-sum board game domain has several properties that cannot be used in a more general domain, such as a discrete state, symmetry properties, and the possibility of using self-play. This section shows how the AlphaGo Zero algorithm was generalized to a domain with a continuous state space and where self-play cannot be used.

A. Tree search

A neural network f_θ, with parameters θ, is used to guide the MCTS. The network takes a state s as input, and outputs the estimated value $V(s, \theta)$ of this state and a vector that represents the prior probabilities $\boldsymbol{p}(s, \theta)$ of taking different actions,

$$(\boldsymbol{p}(s, \theta), V(s, \theta)) = f_\theta(s). \tag{3}$$

For m_{act} possible actions, $P(s, a_i, \theta)$ represents the prior probability of taking action a_i in state s, i.e., $\boldsymbol{p}(s, \theta) = (P(s, a_1, \theta), \ldots, P(s, a_{m_{\text{act}}}, \theta))$.

In order to select which action to take from a given state, the SELECTACTION function from Algorithm 1 is used. This function constructs a search tree, where each state-action node stores a set of statistics $\{N(s, a), Q(s, a), C(s, a)\}$, where $N(s, a)$ is the number of visits of the node, $Q(s, a)$ is the estimated state-action value, and $C(s, a)$ it the set of child nodes. To build the search tree, n iterations are done, where each iteration starts in the root node s_0 and continues for time steps $t = 0, 1, \ldots, L$ until it reaches a leaf node s_L at step L. At each step, the algorithm selects the action that maximizes the UCB condition

$$UCB(s, a, \theta) = \frac{Q(s, a)}{Q_{\max}} + c_{\text{puct}} P(s, a, \theta) \frac{\sqrt{\sum_b N(s, b) + 1}}{N(s, a) + 1}. \tag{4}$$

Here, c_{puct} is a parameter that controls the amount of exploration in the tree. In order to keep c_{puct} constant over environments, the Q-values are normalized by $Q_{\max} = r_{\max}/(1-\gamma)$, where $r_{\max}$ is the maximum possible reward in one time step. The reward is also typically normalized, which then means that $r_{\max} = 1$. In order to perform additional exploration during the training phase (not during evaluation), Dirichlet noise is added to the prior probabilities. Therefore, during training, $P(s, a, \theta)$ is replaced with $(1 - \varepsilon)P(s, a, \theta) + \varepsilon\eta$, where $\eta \sim \text{Dir}(\beta)$, and ε and β are parameters that control the amount of noise.

Algorithm 1 Monte Carlo tree search, guided by a neural network policy and value estimate.

function SELECTACTION(s_0, n, θ)
  for $i \in 1 : n$
    SIMULATE(s_0, θ)
  $\pi(a \mid s) \leftarrow \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}$
  $a \leftarrow$ sample from π
  return a, π
function SIMULATE(s, θ)
  if s is terminal
    return 0
  $a \leftarrow \arg\max_a UCB(s, a, \theta)$
  if $|C(s, a)| \leq kN(s, a)^\alpha$
    $s' \sim G(s, a)$
    $r \leftarrow R(s, a, s')$
    $C(s, a) \leftarrow C(s, a) \cup \{(s', r)\}$
    $v \leftarrow \begin{cases} 0, & \text{if } s' \text{ is terminal} \\ V(s', \theta), & \text{otherwise} \end{cases}$
    $q \leftarrow r + \gamma v$
  else
    $(s', r) \leftarrow$ sample uniformly from $C(s, a)$
    $q \leftarrow r + \gamma$SIMULATE(s', θ)
  $N(s, a) \leftarrow N(s, a) + 1$
  $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q

When an action has been chosen, the progressive widening criterion is checked to decide whether a new child node should be expanded. If the number of children is larger than $kN(s, a)^\alpha$, a previous child node is uniformly sampled from the set $C(s, a)$, and the iteration continues down the search tree. However, if $|C(s, a)| \leq kN(s, a)^\alpha$, a new leaf node is created. The state of this leaf node s_L is sampled from the generative model, $s_L \sim G(s_{L-1}, a_{L-1})$, and the transition reward r_{L-1} is given by the reward function $r_{L-1} = R(s_{L-1}, a_{L-1}, s_L)$. The pair (s_L, r_{L-1}) is then added to the set of child nodes $C(s_{L-1}, a_{L-1})$ and the estimated value of the node $V(s_L, \theta)$ is given by the neural network f_θ. All the action nodes $\{a_*\}$ of the leaf state node are initialized such that $N(s_L, a_*) = 0$, $Q(s_L, a_*) = V(s_L, \theta)$, and $C(s_L, a_*) = \emptyset$.

Finally, the visit count $N(s, a)$ and Q-values $Q(s, a)$ of the parent nodes that were visited during the iteration are updated through a backwards pass.

After n iterations, the tree search is completed and an action a is sampled proportionally to the exponentiated visit count of actions of the root node s_0, according to

$$\pi(a \mid s) = \frac{N(s, a)^{1/\tau}}{\sum_b N(s, b)^{1/\tau}}, \tag{5}$$

where τ is a temperature parameter that controls the level of exploration. During the evaluation phase, $\tau \to 0$, which means that the most visited action is greedily chosen.

Algorithm 2 Procedure for generating training data.

function GENERATETRAININGDATA
  while network not converged
    $s_0 \leftarrow$ GENERATERANDOMSTATE
    $i \leftarrow 0$
    while episode not finished
      $a_i, \pi_i \leftarrow$ SELECTACTION(s_i, n, θ)
      $s_{i+1}, r_i \leftarrow$ STEPENVIRONMENT(s_i, a)
      $i \leftarrow i + 1$
    $N_s \leftarrow i$
    $v_{\text{end}} \leftarrow \begin{cases} 0, & \text{if } s_{N_s} \text{ is terminal} \\ V(s_{N_s}, \theta), & \text{otherwise} \end{cases}$
    for $i \in 0 : N_s - 1$
      $z_i \leftarrow \sum_{k=i}^{N_s-1} \gamma^{k-i} r_k + \gamma^{N_s-i} v_{\text{end}}$
      ADDSAMPLETOMEMORY(s_i, π_i, z_i)

B. Training process

Algorithm 2 shows the process for generating training data, for optimizing the neural network parameters. First, experiences from a simulated environment are generated. For each new episode, a random initial state is sampled and then the episode is run until termination, at step N_s, with actions chosen according to the SELECTACTION function of Algorithm 1. Upon termination, the discounted return z_i that was received during the episode is calculated for each step $i = 0, \ldots, N_s - 1$ by summing the rewards r_i of the episode, according to

$$z_i = \sum_{k=i}^{N_s-1} \gamma^{k-i} r_k + \gamma^{N_s-i} v_{\text{end}}. \tag{6}$$

If the final state s_{N_s} is a terminal state, its value is set to zero, i.e., $v_{\text{end}} = 0$. Otherwise, the value is estimated as $v_{\text{end}} = V(s_{N_s}, \theta)$. For each of the states s_i, the received discounted return z_i and the action distribution from the search tree, $\boldsymbol{\pi}_i = (\pi(a_1 \mid s_i), \ldots, \pi(m_{\text{act}} \mid s_i))$, are used as targets for training the neural network. The tuples $(s_i, \boldsymbol{\pi}_i, z_i)$ are therefore added to a memory of experiences.

In parallel to the collection of new training samples, the neural network parameters θ are optimized from the stored samples by using a gradient descent algorithm [27]. A loss function ℓ is calculated as the sum of the mean-squared value error, the cross entropy loss of the policy, and an L_2 weight regularization term,

$$\ell = c_1(z - V(s, \theta))^2 - c_2 \boldsymbol{\pi}^\top \log \boldsymbol{p}(s, \theta) + c_3 \|\theta\|^2, \tag{7}$$

where c_1, c_2, and c_3 are parameters that balance the different parts of the loss.

IV. IMPLEMENTATION

The presented framework for combining planning and reinforcement learning can be applied to autonomous driving. In this study, the properties of the framework were investigated in a simulated environment for two highway driving cases, which are illustrated in Fig. 1. The first case involves navigating in traffic as efficiently as possible, whereas the second case involves exiting on an off-ramp. This section starts with describing the driver and physical modeling of the cases, which is used both as a generative model and for simulating the environment, and is then followed by a description of how the proposed framework was applied, how the simulations were set up, and how the baseline methods were implemented. The design of the test cases was inspired by Sunberg et al. [15].

A. Driver Modeling

The Intelligent Driver Model (IDM) was used to govern the longitudinal motion of each vehicle [28]. The longitudinal acceleration $\dot{v}_{\text{IDM}}$ is determined by

$$\dot{v}_{\text{IDM}} = a\left(1 - \left(\frac{v_x}{v_{\text{set}}}\right)^4 - \left(\frac{d^*(v_x, \Delta v_x)}{d}\right)^2\right), \tag{8}$$

where v_x is the longitudinal speed of the vehicle, and d^* is the desired distance to the vehicle ahead, given by

$$d^*(v_x, \Delta v_x) = d_0 + v_x T_{\text{set}} + v_x \Delta v_x / (2\sqrt{ab}). \tag{9}$$

The acceleration is a function of the vehicle speed v_x, the distance to the vehicle ahead d, and the speed difference (approach rate) Δv_x. The parameters of the model are the set speed v_{set}, the set time gap T_{set}, the minimum distance d_0, the maximum acceleration a, and the desired deceleration b.

Noise was added to the acceleration of the vehicles $\dot{v}_x$, by setting

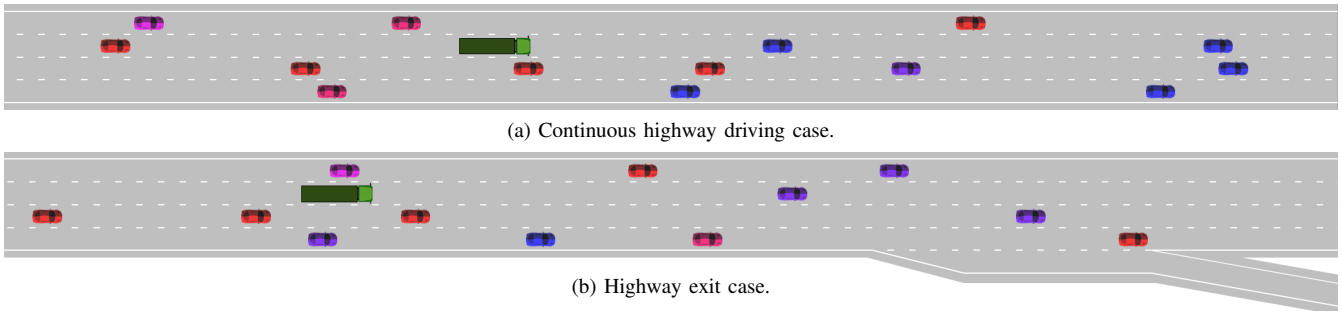
$$\dot{v}_x = \dot{v}_{\text{IDM}} + \frac{\sigma_{\text{vel}}}{\Delta t} w, \tag{10}$$

where w is a normally distributed random variable with unit standard deviation and zero mean, which is independent for each vehicle. The parameter σ_{vel} is the standard deviation of the velocity noise and Δt is the time step of the simulation. No noise was added to the ego vehicle acceleration.

The Minimizing Overall Braking Induced by Lane changes (MOBIL) strategy was used to model the lane changes of the surrounding vehicles [29]. This model makes lane changing decisions with the goal of maximizing the acceleration of all the involved traffic participants at every time step. A lane change is only allowed if the induced acceleration of the following vehicle in the new lane a_{n} fulfills a safety criterion, $a_{\text{n}} > -b_{\text{safe}}$. The IDM is used to predict the accelerations of the neighboring vehicles. Then, a lane change is performed if

$$\tilde{a}_{\text{e}} - a_{\text{e}} + p\left((\tilde{a}_{\text{n}} - a_{\text{n}}) + (\tilde{a}_{\text{o}} - a_{\text{o}})\right) > a_{\text{th}}, \tag{11}$$

where a_{e}, a_{n}, and a_{o} are the accelerations of the ego vehicle, the following vehicle in the target lane, and the following vehicle in the current lane, respectively, if no lane change is performed. Moreover, $\tilde{a}_{\text{e}}$, $\tilde{a}_{\text{n}}$, and $\tilde{a}_{\text{o}}$ are the corresponding accelerations if the ego vehicle changes lane. A politeness factor p controls how much the gains and losses of the surrounding vehicles are valued. The lane change is done if the sum of the weighted acceleration changes is higher than a threshold a_{th}. Finally, if lanes are available both to the left and to the right, the same criterion is applied to both options. If these are both fulfilled, the model chooses the option with the highest acceleration gain.



(a) Continuous highway driving case.

(b) Highway exit case.

Fig. 1. Examples of the two test cases. (a) shows an initial state for the continuous highway driving case, whereas (b) shows the exit case, when the ego vehicle is approaching the exit on the right side of the road. The ego vehicle is the green truck, whereas the color of the surrounding vehicles represent the aggressiveness level of their corresponding driver models, see Sect. IV-E. Red is an aggressive driver, blue is a timid driver, and the different shades of purple represent levels in between.

B. Physical Modeling

Both test cases took place on a straight one-way highway with four lanes. The longitudinal dynamics assumed a constant acceleration, which means that the longitudinal position and speed of a vehicle, x and v_x, were updated according to

$$x' = x + v_x \Delta t + \frac{1}{2}\dot{v}_x \Delta t^2, \tag{12}$$

$$v_x' = v_x + \dot{v}_x \Delta t. \tag{13}$$

Furthermore, the braking acceleration was limited to $b_{\max}$.

The lateral dynamics assumed a constant lateral speed v_y, which means that the lateral position of a vehicle y, was updated according to

$$y' = y + v_y \Delta t. \tag{14}$$

When a lane change was requested, v_y was set to $\pm v_y^{\text{lc}}$, where the sign depends on the intended lane change direction. Otherwise, it was set to 0. Table IV provides the parameter values.

C. POMDP Formulation

This section describes how the decision making problem for the two highway driving cases was formulated as a POMDP, how the state of the system was estimated from observations, and how Algorithm 1 was used to make the decisions. Table IV provides the parameter values.

1) State space, $\mathcal{S}$: The state of the system,

$$s = (s_{\text{term}}, \{(s_i^p, s_i^d)\}_{i \in 0,\dots,N_{\text{veh}}}), \tag{15}$$

consists of the physical state s_i^p and the driver state (driver model parameters) s_i^d of the ego vehicle, and the N_{veh} surrounding vehicles in a traffic scene. There is also a Boolean state s_{term}, which takes the value 1 when a terminal state is reached, and otherwise 0. The physical state consists of the longitudinal and lateral position, and speed, of each vehicle,

$$s_i^p = (x_i, y_i, v_{x,i}, v_{y,i}). \tag{16}$$

The driver state is described by the driver model parameters,

$$s_i^d = (v_{\text{set},i}, T_{\text{set},i}, d_{0,i}, a_i, b_i, p_i, a_{\text{th},i}, b_{\text{safe},i}), \tag{17}$$

which are defined in Sect. IV-A.

TABLE I
ACTION SPACE OF THE AGENT.

a_1	Stay in current lane, keep current ACC setpoint
a_2	Stay in current lane, decrease ACC setpoint
a_3	Stay in current lane, increase ACC setpoint
a_4	Change lanes to the right, keep current ACC setpoint
a_5	Change lanes to the left, keep current ACC setpoint

2) Action space, $\mathcal{A}$: Since this study concerns tactical driving decisions, a high level action space is used. At every time step, the agent can choose between $m_{\text{act}} = 5$ different actions; keep its current driver state a_1, decrease or increase the setpoint of the adaptive cruise controller (ACC) a_2, a_3, and change lanes to the right or to the left a_4, a_5. Table I provides a summary of the available actions. The decision is then forwarded to a lower level operational decision making layer, which handles the detailed execution of the requested maneuver.

In this study, a simplified operational decision making layer is used, where the ACC consists of the IDM. In short, increasing the ACC setpoint corresponds to increasing the requested speed or decreasing the time gap, whereas decreasing the ACC setpoint has the opposite effect. More specifically, when a_3 is chosen and the set speed of the IDM v_{set} is less than the speed desired by a higher level strategic decision making layer v_{des}, then v_{set} is increased by Δv_{set}. However, if $v_{\text{set}} = v_{\text{des}}$, then the set time gap of the IDM T_{set} is decreased with ΔT_{set}. Similarly, if a_2 is chosen and $T_{\text{set}} < T_{\max}$, where $T_{\max}$ is the maximum allowed time gap of the ACC, then T_{set} is increased by ΔT_{set}. However, if $T_{\text{set}} = T_{\max}$, then v_{set} is decreased by Δv_{set}. When action a_4 or a_5 are chosen, the vehicle either starts a lane change, continues a lane change or aborts a lane change, i.e., moves to the right or the left respectively by setting $v_y = \pm v_y^{\text{lc}}$. When a lane change is performed, the set speed is reset to $v_{\text{set}} = v_{\text{des}}$ and the set time gap T_{set} is set to the actual time gap. Decisions were taken at an interval of Δt.

At every time step, the action space is pruned, in order to remove actions that lead to collisions. A lane change action is only allowed if the ego vehicle or the new trailing vehicle need to brake with an acceleration lower than $a_{\min}$ to avoid a collision. Since the IDM itself is also crash-free, the ego vehicle will never cause a collision. Furthermore, a minimum time gap $T_{\min}$ setpoint of the ACC is used. Therefore, if $T_{\text{set}} =$

$T_{\min}$, then a_3 is not considered. Moreover, if a lane change is ongoing, i.e., the vehicle is positioned between two lanes, only actions a_4 and a_5 are considered, i.e., to continue the lane change or change back to the previous lane.

3) Reward model, R: The objective of the agent is to navigate the highway safely and efficiently. Since safety is handled by a crash-free action set, a simple reward function that tries to minimize the time and the number of lane changes is used. Normally, at every time step, a positive reward of $1 - |\frac{v_{\text{ego}} - v_{\text{des}}}{v_{\text{des}}}|$ is given, which penalizes deviations from the desired speed. If a lane change is initiated, a negative reward of c_{lc} is added. Finally, for the case with the highway exit, a reward r_{term} is added when the agent transitions to a terminal state. If the exit is reached at this time, then $r_{\text{term}} = \gamma \frac{1}{1-\gamma}$, which (from a geometric sum) corresponds to that the vehicle would have continued to drive forever and gotten the reward 1 at every subsequent step. If the exit is not reached, then $r_{\text{term}} = 0$.

4) State transition model, T: The state transition model is implicitly defined by a generative model.

5) Generative model, G: The combination of the IDM and MOBIL model, and the physical model are used as a generative model G, where $s' \sim G(s, a)$. The same generative model is used in the tree search of Algorithm 1.

6) Observation space, $\mathcal{O}$: The observations o consists of the physical states of the surrounding vehicles, and the physical and driver state of the ego vehicle, i.e., $o = (s_0^p, s_0^d, \{s_i^p\}_{i \in 1,\dots,N_{\text{veh}}})$. The driver states of the surrounding vehicles are not included in the observation.

7) Observation model, O: A simplified sensor model was used in this study. The physical state of all vehicles that are positioned closer than x_{sensor} of the ego vehicle is assumed to be observed exactly, whereas vehicles further away are not detected at all.

8) Belief state estimator: The driver state of the surrounding vehicles cannot be directly observed, but it can be estimated from the vehicles' physical state history by using a particle filter [25]. A particle $\hat{s}^d$ represents the value of the driver model parameters of the observed surrounding vehicles. At a given time step, a collection of M particles $\{\hat{s}_k^d\}_{k=1}^M$ and their associated weights $\{W_k\}_{k=1}^M$ describe the belief of the driver model parameters. At the next time step, after action a has been taken, the belief is updated by sampling M new particles with a probability that is proportional to the weights. Then, new states are generated by $\hat{s}'_k = G((s^p, s_0^d, \hat{s}_k^d), a)$. Note that there is a component of noise in G, see Sect. IV-A. The new weights are calculated from the new observation, as the approximate conditional probability

$$W'_k = \begin{Bmatrix} \exp\left(-\frac{(v'-\hat{v}')^2}{2\sigma_{\text{vel}}^2}\right) & \text{if } y' = \hat{y}' \\ \gamma_{\text{lane}} \exp\left(-\frac{(v'-\hat{v}')^2}{2\sigma_{\text{vel}}^2}\right) & \text{otherwise} \end{Bmatrix} \propto \Pr\left(\hat{s}_k \mid o\right), \tag{18}$$

where v' and y' are given by the observation, $\hat{v}'$ is taken from $\hat{s}'_k$, and γ_{lane} is a parameter that penalizes false lane changes. Furthermore, Gaussian noise with a standard deviation that is proportional to the sample standard deviation of the current M particles is added to 10% of the new samples, in order to prevent particle deprivation. The design of the particle filter was inspired by Sunberg et al. [15].

TABLE II
INPUT TO THE NEURAL NETWORK ξ.

Ego lane	$\xi_1 = 2y_0/y_{\max} - 1$
Ego vehicle speed	$\xi_2 = 2v_{x,0}/v_{x,0}^{\max} - 1$
Lane change state	$\xi_3 = \text{sgn}\,(v_{y,0})$
Ego vehicle set speed	$\xi_4 = 2v_{\text{set},0}/v_{x,0}^{\max} - 1$
Ego vehicle set time gap	$\xi_5 = \frac{T_{\text{set},0} - (T_{\max}+T_{\min})/2}{(T_{\max}-T_{\min})/2}$
Distance to exit	$\xi_6 = 1 - 2x_0/x_{\text{exit}}$
Terminal state	$\xi_7 = s_{\text{term}}$
Relative long. position of vehicle i	$\xi_{7i+1} = (x_i - x_0)/x_{\text{sensor}}$
Relative lat. position of vehicle i	$\xi_{7i+2} = (y_i - y_0)/y_{\max}$
Relative speed of vehicle i	$\xi_{7i+3} = \frac{v_{x,i} - v_{x,0}}{v_{\text{set}}^{\max} - v_{\text{set}}^{\min}}$
Lane change state of vehicle i	$\xi_{7i+4} = \text{sgn}\,(v_{y,i})$

In Algorithm 1, the estimated most likely state is used as input. The function is called with SELECTACTION(s_0, n, θ), where $s_0 = (s_{\text{term}}, s^p, s_0^d, \hat{s}_{\max}^d)$ consists of the terminal state, the observed physical state, the observed ego driver state, and the particle with the highest weight. This particle represents the most likely driver model state, given by $\hat{s}_{\max}^d = \hat{s}_{\arg\max_k W_k}^d$.

D. Neural Network Architecture and Training Process

A neural network estimates the prior probabilities of taking different actions and the value of the current state. In this implementation, before the state s is passed through the neural network, it is converted to ξ, where all states are normalized, i.e., $\xi_* \in [-1, 1]$, and the positions and velocities of the surrounding vehicles are expressed relative to the ego vehicle. There are $m_{\text{ego}} = 7$ inputs that describe the ego vehicle state and $m_{\text{veh}} = 4$ inputs that describe the state of each surrounding vehicle. The first elements, ξ_1 to ξ_7, describe the state of the ego vehicle, whereas ξ_{7i+1}, ξ_{7i+2}, ξ_{7i+3}, and ξ_{7i+4}, for $i = 1, \dots, N_{\max}$, represent the relative state of the surrounding vehicles. If there are less than $N_{\max}$ vehicles in the sensing range, the remaining inputs are padded with dummy values, $\xi_{7i+1} = -1$ and $\xi_{7i+2} = \xi_{7i+3} = \xi_{7i+4} = 0$, which will not affect the output of the network, see below. Table II describes how ξ is calculated and the values of the normalization parameters are given in Table IV.

In a previous study [21], we introduced a temporal convolutional neural network (CNN) architecture, which simplifies and speeds up the training process by applying CNN layers to the part of the input that consists of interchangeable objects, in this case surrounding vehicles. The input that describes the surrounding vehicles is passed through CNN layers, with a design that results in identical weights for each vehicle, and finally a max pooling layer creates a translational invariance between the vehicles. With this structure, the output will not depend on how the vehicles are ordered in the input vector. It also removes the problem with a fix input vector size, since it can simply be made larger than necessary and padded with dummy values for the extra vehicle slots. The extra input will then be removed by the max pooling layer.

The neural network architecture, shown in Fig. 2, was used in this study and includes the described CNN architecture.

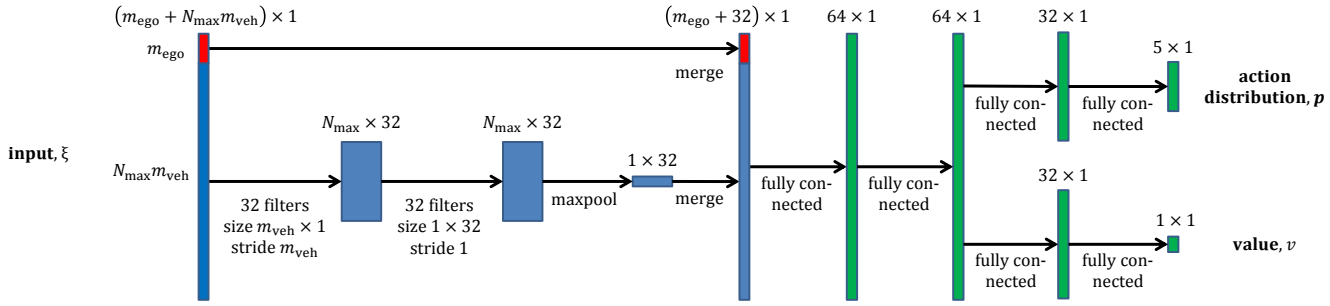


Fig. 2. The figure illustrates the neural network architecture that was used in this study. The convolutional and max pooling layers create a translational invariance between the input from different surrounding vehicles, which makes the ordering and the number of vehicles irrelevant.

TABLE III
PARAMETERS OF THE MCTS AND THE NEURAL NETWORK TRAINING.

MCTS iterations	n	2,000
Exploration parameter	c_{puct}	0.1
Progressive widening linear param.	k	1.0
Progressive widening exponent param.	α	0.3
Temperature parameter,	τ	1.1
Dirichlet noise parameter	β	1.0
Noise proportion parameter	ϵ	0.25
Training start	N_{start}	20,000
Replay memory size	M_{replay}	100,000
Mini-batch size	M_{mini}	32
Loss function weights	(c_1, c_2, c_3)	$(100, 1, 0.0001)$
Learning rate	η	0.01
Momentum	μ	0.9

TABLE IV
VARIOUS SIMULATION PARAMETERS.

Velocity noise standard deviation	σ_{vel}	0.5 m/s
Physical braking limit	$b_{\max}$	8.0 m/s^2
Simulation time step	Δt	0.75 s
Lateral lane change speed	v_y^{lc}	0.67 lanes/s
Minimum set time gap	$T_{\min}$	0.5 s
Maximum set time gap	$T_{\max}$	2.5 s
Step set time gap	ΔT_{set}	1.0 s
Step set speed	Δv_{set}	2.0 m/s
Desired speed of strategic layer	v_{des}	25 m/s
Lane change penalty	c_{lc}	−0.03
Discount factor	γ	0.95
Surrounding sensor range	x_{sensor}	100 m
Maximum number of vehicles	$N_{\max}$	20
Exit lane position	x_{exit}	1,000 m
Initial ego speed	$v_{x,0}$	20 m/s
Number of particles	M	500
Particle filter lane factor	γ_{lane}	0.2
State normalization lane	$y_{\max}$	4 lanes
Maximum ego vehicle speed	$v_{x,0}^{\max}$	25 m/s
Minimum set speed (timid driver)	$v_{\text{set}}^{\min}$	19.4 m/s
Maximum set speed (aggressive driver)	$v_{\text{set}}^{\max}$	30.6 m/s

In short, the input that describe the state of the surrounding vehicles is passed through two convolutional layers, followed by a max pooling layer. This is then concatenated with the input that describes the ego vehicle state and passed through two fully connected layers, before being split up into two parallel heads, which estimate the action distribution $\boldsymbol{p}(s, \theta)$ and the value $V(s, \theta)$ of the input state s. All layers have ReLU activation functions, except for the action head, which has a softmax activation function, and the value head, which has a sigmoid activation function. Finally, the value output is scaled with the factor $1/(1-\gamma)$ (since this is the maximum possible value of a state).

Algorithm 2 was used to collect training samples. When an episode was finished, the n_{new} samples were added to a replay memory of size M_{replay}. Learning started after N_{start} samples had been added. Then, after each finished episode, the network was trained on n_{new} mini-batches, with size M_{mini}, uniformly drawn from the memory. The loss was calculated by using Eq. 7 and the neural network parameters θ were optimized by stochastic gradient descent, with learning rate η and momentum μ. The parameters of Algorithm 1 and the training process are shown in Table III. In order to speed up the training process, the algorithm was parallelized. Twenty workers simultaneously ran simulations to generate training data. They all shared the same neural network and update process. Worker calls to $f_\theta(s)$ were queued and passed to the neural network in batches. The training was performed on a desktop computer, which included a GPU.

E. Episode Implementation

As mentioned above, both the test cases consisted of a straight one-way highway with four lanes. Overtaking was allowed both on the left and the right side of another vehicle. The continuous driving case ended after 200 time steps (with $s_{\text{term}} = 0$), whereas the exit case ended when the ego vehicle reached the exit position longitudinally, i.e., $x_0 \geq x_{\text{exit}}$ (with $s_{\text{term}} = 1$). To successfully reach this exit, the ego vehicle had to be in the rightmost lane at this point. The ego vehicle, a 12.0 m long truck, started in a random lane for the continuous case and in the leftmost lane for the exit case, with an initial velocity of $v_{x,0}$. The allowed speed limit for trucks was 25 m/s, hence $v_{\text{des}} = 25$ m/s. In short, around 20 passenger cars, with a length of 4.8 m, were placed on the road, where slower vehicles were positioned in front of the ego vehicle and faster vehicles were positioned behind. An example of an initial state is shown in Fig. 1a. The details on how the initial states were created are described below.

The surrounding vehicles were controlled by the IDM and MOBIL model. The marginal distribution of the model parameters were uniformly distributed between aggressive and timid driver parameters, which were slightly adapted from Kesting et al. [30] and shown in Table V. The main difference is that the

TABLE V
IDM AND MOBIL MODEL PARAMETERS FOR DIFFERENT DRIVER TYPES.

		Normal	Timid	Aggressive
Desired speed (m/s)	v_{set}	25.0	19.4	30.6
Desired time gap (s)	T_{set}	1.5	2.0	1.0
Minimum gap distance (m)	d_0	2.0	4.0	0.0
Maximal acceleration (m/s^2)	a	1.4	0.8	2.0
Desired deceleration (m/s^2)	b	2.0	1.0	3.0
Politeness factor	p	0.05	0.1	0.0
Changing threshold (m/s^2)	a_{th}	0.1	0.2	0.0
Safe braking (m/s^2)	b_{safe}	2.0	1.0	3.0

politeness factor is here significantly reduced, to create a more challenging task, where slow drivers do not always try to move out of the way. In order to create a new driver model, values were drawn from a Gaussian copula, which had a covariance matrix with 1 along the diagonal and a correlation of $\rho = 0.75$ elsewhere. These values were then scaled and translated to the range between aggressive and timid driver parameters.

In order to create the initial state of the simulation, at first only the ego vehicle was placed on the road and the simulation was run for $n_{\text{init}} = 200$ steps. During this phase, the ego vehicle was controlled by the IDM and it made no lane changes. At every time step, a new vehicle with random parameters was generated. If it was faster than the ego vehicle, it was inserted 300 m behind it, and if it was slower, 300 m in front of it. Furthermore, it was placed in the lane that had the largest clearance to any other vehicle. However, if d^* (see Eq. 9) of the new vehicle, or the vehicle behind it, was less than the actual gap, the vehicle was not inserted. At most $N_{\max}$ vehicle were added. The state after these n_{init} steps was then used as the initial state of the new episode. The ego vehicle driver state was initialized to the same parameters as a 'normal' driver (Table V).

F. Baseline Methods

The performance of the framework that is introduced in this paper is compared to two baseline methods. For both test cases, standard MCTS with progressive widening, described in Sect. II-B, is used as a baseline, with the same POMDP formulation that is described in Sect. IV-C. Furthermore, for a fair comparison, the same parameter values as for the introduced framework is used, described in Table III, and the exploration parameter is set to $c_{\text{uct}} = c_{\text{puct}}$. Rollouts are done using the IDM and MOBIL model, with the 'normal' driver parameters of Table V, and a rollout depth of 20 time steps. After n iterations are performed, the action with the highest visit count is chosen.

For the continuous highway driving case, a second baseline is the IDM and MOBIL model, with the 'normal' driver parameters. A similar model is used as a second baseline for the highway case with an exit. Then, the driver follows the IDM longitudinally and changes lanes to the right as soon as the safety criterion of the MOBIL model is fulfilled.

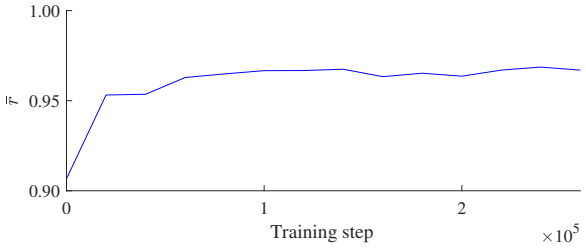


Fig. 3. Average reward per step $\bar{r}$ during the evaluation episodes, as a function of the number of training steps, for the continuous highway driving case.

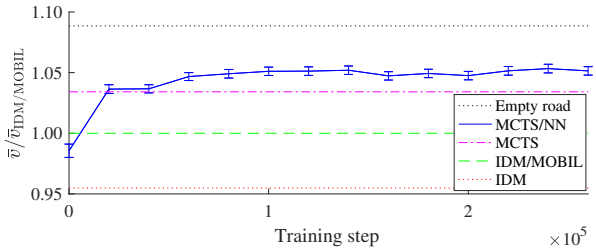


Fig. 4. Mean speed $\bar{v}$ during the evaluation episodes for the continuous highway driving case, normalized with the mean speed of the IDM/MOBIL model $\bar{v}_{\text{IDM/MOBIL}}$. The error bars indicate the standard error of the mean for the MCTS/NN agent, i.e., $\sigma_{\text{sample}}/\sqrt{100}$, where σ_{sample} is the standard deviation of the 100 evaluation episodes.

V. RESULTS

The results show that the agents that were obtained by applying the proposed framework to the continuous highway driving case and the highway exit case outperformed the baseline methods. This section presents further results for the two test cases, together with an explanation and brief discussion on some of the characteristics of the results. A more general discussion follows in Sect. VI. The decision making agent that was created from the presented framework is henceforth called the MCTS/NN agent (where NN refers to neural network), whereas the baseline methods are called the standard MCTS agent and the IDM/MOBIL agent.

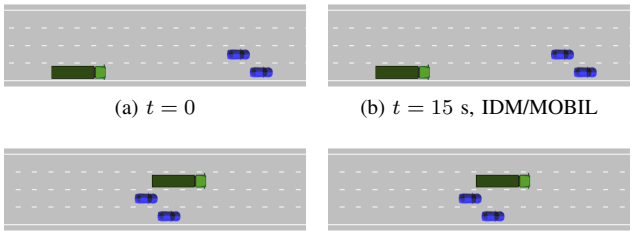
For both test cases, the MCTS/NN agent was trained in a simulated environment (Sect. IV-D). At every 20,000 added training samples, an evaluation phase was run, where the agent was tested on 100 different episodes. These evaluation episodes were randomly generated (Sect. IV-E), but they were identical for all the evaluation phases and for the different agents.

A. Continuous Highway Driving Case

Fig. 3 shows the average reward $\bar{r}$ per step that was received during the evaluation episodes, as a function of the number of added training samples, henceforth called training steps. The maximum possible reward for every step is 1, and it is decreased when the agent deviates from the desired speed and/or make lane changes. The agent performed relatively well even before the training had started, due to its planning component. However, with only 20,000 training steps, the agent learned to perform significantly better. The average received reward then increased slightly with more training, until around 100,000 steps, where the performance settled.

TABLE VI
ACTION DISTRIBUTION FOR THE CONTINUOUS HIGHWAY DRIVING CASE.

	Idle	ACC down	ACC up	Right	Left
IDM/MOBIL	95.2%	-	-	2.3%	2.5%
MCTS	61.8%	12.4%	13.8%	6.0%	6.0%
MCTS/NN	90.4%	1.1%	3.9%	2.1%	2.5%



(a) $t = 0$ (b) $t = 15$ s, IDM/MOBIL

(c) $t = 15$ s, MCTS (d) $t = 15$ s, MCTS/NN

Fig. 5. Example of a situation where planning is necessary. The initial state is shown in (a) and the state after 15 s is shown for the three agents in (b), (c), and (d). The green truck is the ego vehicle.

A comparison of the average speed $\bar{v}$ and the action distribution during the evaluation episodes was made for the MCTS/NN agent and the baseline methods. Fig. 4 shows how $\bar{v}$ varies during the training of the agent, normalized with the mean speed of the IDM/MOBIL model $\bar{v}_{\mathrm{IDM/MOBIL}}$. For reference, the figure displays the average speed when applying the IDM, which always stays in its original lane during the whole episode, and can therefore be considered as a minimum performance. The ideal mean speed (25 m/s) for when the road is empty is also indicated, which is naturally not achievable in these episodes due to the surrounding traffic. The figure shows that the standard MCTS agent outperformed the IDM/MOBIL agent, and that the MCTS/NN agent quickly learned to match the performance of the MCTS agent and then surpassed it after around 60,000 training steps. Table VI shows the action distribution for the baseline methods and for the MCTS/NN agent after 250,000 training steps. The trained MCTS/NN agent and the IDM/MOBIL agent performed lane changes at about the same rate, whereas the standard MCTS agent made significantly more lane changes. It also changed its ACC state more than the trained MCTS/NN agent.

A key difference between the compared methods is highlighted in Fig. 5, which shows a situation where planning is required. The ego vehicle is here placed behind two slow vehicles in adjacent lanes, with timid driver parameters. The best behavior for the ego vehicle is to make two lane changes to the left, in order to overtake the slow vehicles. Both the standard MCTS agent and the trained MCTS/NN agents solved this situation, whereas the IDM/MOBIL agent got stuck in the original lane.

Finally, the effect of the number of MCTS iterations n on the performance of the trained agent was investigated, which is shown in Fig. 6. To execute just one iteration corresponds to using the policy that was learned by the neural network, which performed better than the IDM/MOBIL agent. At around 30 iterations the MCTS/NN agent surpassed the standard MCTS agent, which used 2,000 iterations, and at $n = 1{,}000$ the performance settled.

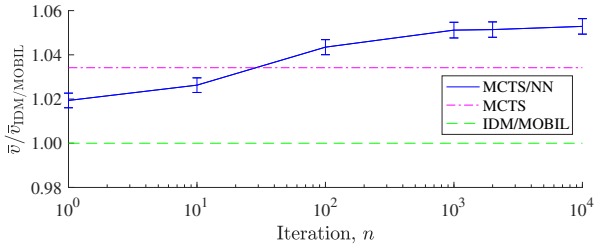


Fig. 6. Normalized mean speed as a function of the number of MCTS iterations n for the trained MCTS/NN agent, in the continuous highway driving case. The error bars indicate the standard error of the mean.

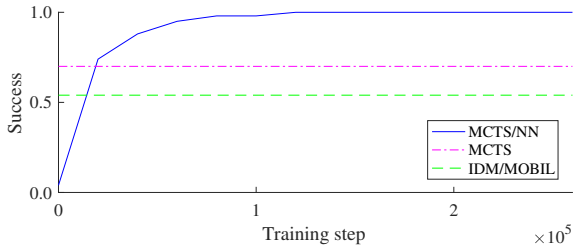


Fig. 7. Proportion of successful evaluation episodes, as a function of training steps, for the highway exit case.

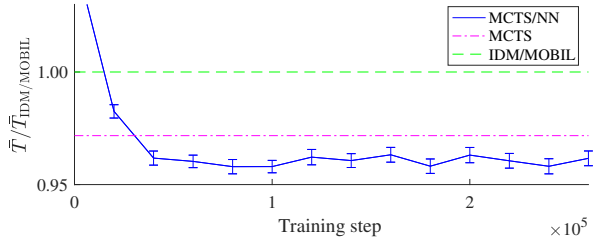
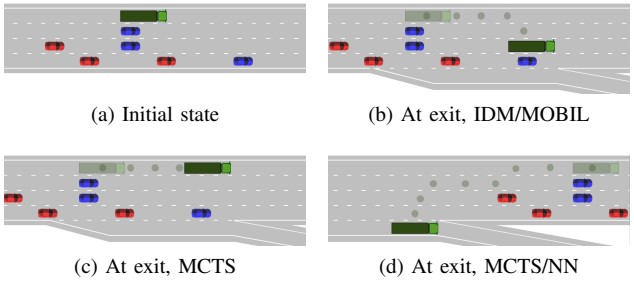


Fig. 8. The average time $\bar{T}$ it took to reach the exit during the evaluation episodes, normalized with the time of the IDM/MOBIL agent $\bar{T}_{\mathrm{IDM/MOBIL}}$, as a function of training steps. The error bars indicate the standard error of the mean.

B. Highway Exit Case

The highway exit case is conceptually different from the continuous driving case, since it has a pass/fail outcome. In this case, the most important objective is to reach the exit, whereas a secondary objective is to do so in a time efficient way. Fig. 7 shows the proportion of evaluation episodes where the exit was reached, as a function of training steps for the MCTS/NN agent. The agent quickly learned how to reach the exit in most episodes, and after around 120,000 training steps, it solved all of them. The success rate of the baseline methods was 70% for the standard MCTS agent and 54% for the modified IDM/MOBIL agent. The time it took to reach the exit for the different methods is shown in Fig. 8. Only the episodes where all methods succeeded to reach the exit are included in the comparison.

Similarly to the continuous highway driving case, there are situations for the highway exit case where it is necessary to plan over a long time horizon. One such situation is shown in Fig. 9. There, the ego vehicle starts in the leftmost lane, 300 m from the exit, and six vehicles are positioned in the other lanes. Three of the vehicles have timid driver parameters and



(a) Initial state (b) At exit, IDM/MOBIL

(c) At exit, MCTS (d) At exit, MCTS/NN

Fig. 9. Example of when it is necessary to plan relatively far into the future to solve a specific situation. The initial state is shown in (a) and the state at the exit is shown for the three agents in (b), (c), and (d). The dots show the position of the ego vehicle relative to the other vehicles during the maneuver, i.e., in (b) and (c) the ego vehicle accelerates and overtakes the slower vehicles, whereas in (d), the ego vehicle slows down and ends up behind the same vehicles.

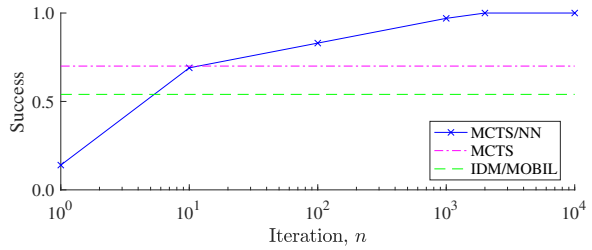


Fig. 10. Proportion of successful episodes as a function of the number of MCTS iterations n for the trained MCTS/NN agent, in the highway exit case.

the other three have aggressive driver parameters, except for the set speed, which is $v_{\text{set}} = 21$ m/s for all of them. All vehicles also start with an initial speed of 21 m/s. The single way the ego vehicle can reach the exit in this situation is to first slow down and then make several lane changes to the right, which was only discovered by the trained MCTS/NN agent. The standard MCTS agent never found the way to the exit in its tree search and therefore stayed in its original lane, to not get negative rewards from changing lanes. The IDM/MOBIL agent accelerated to 25 m/s and changed lanes to right as far as it could, without reaching the exit.

The effect of the number of MCTS iterations n is shown in Fig. 10. When using the learned policy from the neural network, i.e., one iteration, the MCTS/NN agent only succeeded in 14% of the evaluation episodes. With ten iterations, the success rate matched the standard MCTS agent, which used 2,000 iterations. Then, when the MCTS/NN agent also used 2,000 iterations, it managed to plan far enough to solve all the evaluation episodes.

In order to illustrate the behavior of the trained MCTS/NN agent, Fig. 11 shows the learned value and the action that was taken for different states when approaching the exit, with no other vehicles present. The ego vehicle had an initial speed of 25 m/s, $v_{\text{set}} = 25$ m/s, and $T_{\text{set}} = 2.5$ s. For states longitudinally far away from the exit, the agent learned that the value is 20, which corresponds to expecting a reward of 1 for all future steps (since the geometric sum of Eq. 1 equals 20 for $\gamma = 0.95$). As expected, the learned value decreases for all the lanes, except for the rightmost lane, for states close to the exit. Far from the exit, the agent always chooses action a_1, i.e., to stay in the current lane and keep its current speed, whereas closer to the exit, the agent changes lanes to the right, to bring it to the rightmost lane.

VI. Discussion

The results show that the agents that were obtained by applying the proposed framework to the two test cases outperform the baseline methods, and that the advantage is more significant for the highway exit case, especially in the number of solved test episodes (Fig. 7). The reason for that the difference is larger in the exit case is that it is a more complex case, where the effect of the policy is more decisive. A suboptimal action in the continuous highway driving case just means a small time loss, whereas a mistake in the exit case can result in that the exit is not reached. Moreover, in general, the exit case requires a longer planning horizon than the continuous case, which is exemplified in Fig. 9. Such a situation cannot be resolved by the IDM/MOBIL agent, since it does not perform any planning. The standard MCTS agent also fails, since in the example situation, the MCTS does not reach deep enough in the search tree to figure out how the ego vehicle could reach the exit However, the prior action probabilities and the value estimate of different states, which the MCTS/NN agent obtained from the training, allows it to focus the tree search to the most promising regions. Thus, for the same number of MCTS iterations, it can search deeper in the tree and perform planning over a longer time horizon than the standard MCTS. In the example situation, the MCTS/NN agent is therefore able to figure out which actions that are needed in order to reach the exit.

The MCTS/NN agent is anytime capable, i.e., it can abort its search after any number of iterations, even after just one, which will then return the action given by the neural network. More iterations will in general improve the result, up to a limit, where the performance settles. In the cases considered in this study, full performance was reached at around $n = 1{,}000$, see Fig. 6 and 10. The number of searches that are necessary depends on the complexity of the environment and the specific traffic situation, which will require different planning depths, as was discussed above.

As mentioned in Sect. IV-C, a simple reward model was used, which promotes driving close to the desired speed and penalizes lane changes. This model proved to work well in this study, but a more careful design may be required for other cases. Additional aspects, such as fuel efficiency or the influence on the surrounding traffic, could also be included. A reward model that mimics human preferences could be determined by using inverse reinforcement learning [31].

In a previous paper [21], we introduced a different method, where a DQN agent learned to make tactical decisions for a case that was similar to the continuous highway driving case described here. That method required around one order of magnitude more training samples to reach a similar performance as the MCTS/NN agent, which indicates the value of letting a planning component guide the learning process, from a sample efficiency perspective. However, each training sample is more computationally expensive to obtain for the

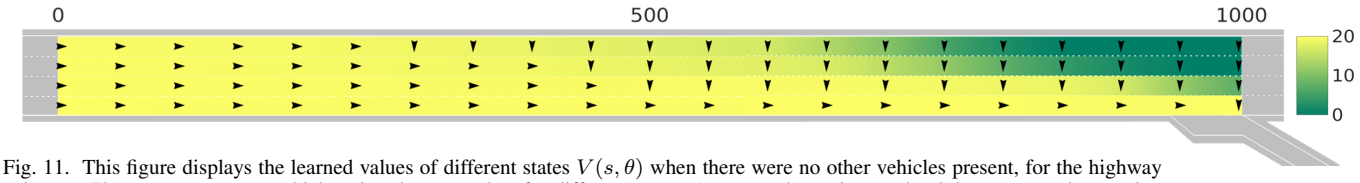


Fig. 11. This figure displays the learned values of different states $V(s, \theta)$ when there were no other vehicles present, for the highway exit case. The arrows represent which action that was taken for different states. An arrow that points to the right corresponds to action a_1, whereas downwards corresponds to a_4 (Table I). Note that the axes do not have the same scale.

method presented here, due to the many MCTS iterations that are done for each decision. If the training is carried out in a simulated environment where training samples are cheap, the advantage of the sample efficiency of the MCTS/NN agent can be argued, but if the training samples are obtained from real world driving where each sample is expensive, sample efficiency is important. For the two test cases in this study, the MCTS agent required around 100,000 training samples, which corresponds to around 20 hours of driving.

The generality of the proposed decision making framework was demonstrated by applying it to two different cases, that are conceptually different. In the continuous highway driving case, the only goal is to navigate in traffic as efficiently as possible, whereas in the exit case, there is a terminal state with a pass/fail outcome, i.e., if the exit was reached or not. In order to apply the framework to other cases, the following components need to be defined: the state space $\mathcal{S}$, the action space $\mathcal{A}$, the reward model R, the generative model G, and the belief state estimator. However, the tree search and training process of Algorithm 1 and 2 would be identical for all cases.

When training a decision making agent by using the framework presented in this paper, or any other machine learning technique, it is important to note that the agent will only be able to solve the type of situations it encounters during the training process. Therefore, it is crucial to set up the training episodes so that they cover the intended case. Moreover, when using machine learning to create a decision making agent, it is difficult to guarantee functional safety of the agent's decisions. A common way to avoid this problem is to use an underlying safety layer, which ensures that the planned trajectory is safe before it is executed by the vehicle control system [32].

In this paper, the AlphaGo Zero algorithm was extended to a domain with a continuous state space, a not directly observable state, and where self-play cannot be used. A generative model replaced the self-play component, progressive widening was added to deal with the continuous state space, and a state estimation component was added to handle the unknown state. Furthermore, the CNN architecture of AlphaGo Zero, which due to Go's grid-like structure can be used to extract important features, was replaced by a CNN architecture that was applied to features of the surrounding vehicles. One technical difference to the AlphaGo Zero algorithm is the UCB condition in Eq. 4, which determines which action to expand in the tree search. The numerator is here changed from AlphaGo Zero's $\sqrt{\sum_b N(s,b)}$ to $\sqrt{\sum_b N(s,b)+1}$, which means that when the tree search reaches a leaf node, it will choose to expand the action that is recommended by the neural network policy, i.e., $a = \arg\max_a P(s, a, \theta)$, instead of a random action. This proved to be beneficial in this study, but more investigations are necessary to determine if it is beneficial in general. Two other small technical differences are that the Q-value in the UCB condition is normalized, in order to keep c_{puct} constant for different environments, and that the Q-value of a leaf node is initialized to the value that is estimated by the neural network $V(s, \theta)$, which for this domain is a better estimate than setting it to zero, as in AlphaGo Zero.

VII. Conclusions

The results of this paper show that the presented framework, which combines planning and learning, can be used to create a tactical decision making agent for autonomous driving. For two conceptually different highway driving cases, the resulting agent performed better than individually using planning, in the form of MCTS, or learning, in the form of the trained neural network. The agent also outperformed a baseline method, based on the IDM and MOBIL model. The presented framework is flexible and can easily be adapted to other driving environments. It is also sample efficient and requires one order of magnitude less training samples than a DQN agent that was applied to a similar case [21].

Future work includes to further investigate how the tactical decision making problem can be formulated as a POMDP in the most efficient way, to validate the presented framework on more test cases, to investigate the effect of different parameters choices, and to implement and test the framework in a real vehicle.

References

[1] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[2] J. A. Michon, "A critical view of driver behavior models: What do we know, what should we do?," in *Human Behavior and Traffic Safety* (L. Evans and S. R.C., eds.), pp. 485–524, Boston, MA: Springer US, 1985.

[3] R. Díaz de León Torres, M. Molina, and P. Campoy, "Survey of Bayesian networks applications on unmanned intelligent autonomous vehicles," *CoRR*, vol. abs/1901.05517, 2019.

[4] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, 2008.

[5] M. Montemerlo *et al.*, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[6] A. Bacha *et al.*, "Odin: Team VictorTango's entry in the DARPA urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 467–492, 2008.

[7] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *IEEE International Conference on Robotics and Automation*, pp. 987–993, 2010.

[8] P. Nilsson, L. Laine, N. van Duijkeren, and B. Jacobson, "Automated highway lane changes of long vehicle combinations: A specific comparison between driver model based control and non-linear model predictive control," in *International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, pp. 1–8, 2015.

[9] F. Damerow and J. Eggert, "Risk-aversive behavior planning under multiple situations with uncertainty," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 656–663, 2015.

[10] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[11] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 392–399, 2014.

[12] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A POMDP approach," *Int. J. Rob. Res.*, vol. 33, no. 9, pp. 1288–1302, 2014.

[13] S. Ulbrich and M. Maurer, "Towards tactical lane change behavior planning for automated vehicles," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 989–995, 2015.

[14] C. Browne *et al.*, "A survey of Monte Carlo tree search methods.," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[15] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, "The value of inferring the internal state of traffic participants for autonomous freeway driving," in *American Control Conference (ACC)*, pp. 3004–3010, 2017.

[16] E. Sonu, Z. Sunberg, and M. J. Kochenderfer, "Exploiting hierarchy for scalable decision making in autonomous driving," in *IEEE Intelligent Vehicles Symposium (IV)*, pp. 2203–2208, 2018.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, second ed., 2018.

[18] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[20] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[21] C. J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2148–2155, 2018.

[22] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep Q-learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3169–3174, 2018.

[23] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," in *NIPS Workshop on Machine Learning for Intelligent Transportation Systems*, 2017.

[24] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *CoRR*, vol. abs/1610.03295, 2016.

[25] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[26] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *Learning and Intelligent Optimization*, pp. 433–445, 2011.

[27] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[28] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, 2000.

[29] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model MOBIL for car-following models," *Transportation Research Record*, vol. 1999, pp. 86–94, 2007.

[30] A. Kesting, M. Treiber, and D. Helbing, "Agents for traffic simulation," in *Multi-Agent Systems: Simulation and Applications* (A. M. Uhrmacher and D. Weyns, eds.), pp. 325–356, CRC Press, 2009.

[31] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *International Conference on Machine Learning*, pp. 663–670, 2000.

[32] S. Underwood, D. Bartz, A. Kade, and M. Crawford, "Truck automation: Testing and trusting the virtual driver," in *Road Vehicle Automation 3* (G. Meyer and S. Beiker, eds.), pp. 91–109, Springer, 2016.

Carl-Johan Hoel Carl-Johan Hoel received the B.S. and M.S degrees in physics from Chalmers University of Technology, Gothenburg, Sweden, in 2011. He is currently working towards the Ph.D. degree at Chalmers and Volvo Group, Gothenburg, Sweden. His research interests include reinforcement learning methods to create a general tactical decision making agent for autonomous driving.

Katherine Driggs-Campbell Katie Driggs-Campbell received her BSE in Electrical Engineering from Arizona State University and her MS and PhD in Electrical Engineering and Computer Science from the University of California, Berkeley. She is currently an Assistant Professor in the ECE Department at the University of Illinois at Urbana-Champaign. Her research focuses on exploring and uncovering structure in complex human-robot systems to create more intelligent, interactive autonomy. She draws from the fields of optimization, learning & AI, and control theory, applied to human robot interaction and autonomous vehicles.

Krister Wolff Krister Wolff received the M.S. degree in physics from Gothenburg University, Gothenburg, Sweden, and the Ph.D. degree from Chalmers University of Technology, Gothenburg, Sweden. He is currently an Associate Professor of adaptive systems, and he is also the Vice head of Department at Mechanics and maritime sciences, Chalmers. His research is within the application of AI in different domains, such as autonomous robots and self-driving vehicles, using machine learning and bio-inspired computational methods as the main approaches.

Leo Laine Leo Laine received the Ph.D. degree from Chalmers University of Technology, Gothenburg, Sweden, within Vehicle Motion management. Since 2007, he has been with the Volvo Group Trucks Technology in the Vehicle Automation department. Since 2013, he has also been an Adjunct Professor in vehicle dynamics with Chalmers Vehicle Engineering and Autonomous Systems. Since 2013, he is specialist within complete vehicle control. Since 2017, he is technical advisor within Vehicle Automation department.

Mykel J. Kochenderfer Mykel J. Kochenderfer received the B.S. and M.S. degrees in computer science from Stanford University, Stanford, CA, USA, and the Ph.D. degree from The University of Edinburgh, Edinburgh, U.K. He is currently an Assistant Professor of aeronautics and astronautics with Stanford University. He is also the Director of the Stanford Intelligent Systems Laboratory, conducting research on advanced algorithms and analytical methods for the design of robust decision-making systems.

Paper IV

Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation

in

Proceedings of the 31st IEEE Intelligent Vehicles Symposium, Virtual conference, 2020, pp. 1563-1569.

Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation

Carl-Johan Hoel*†, Krister Wolff*, and Leo Laine*†

***Abstract*— Reinforcement learning (RL) can be used to create a tactical decision-making agent for autonomous driving. However, previous approaches only output decisions and do not provide information about the agent's confidence in the recommended actions. This paper investigates how a Bayesian RL technique, based on an ensemble of neural networks with additional randomized prior functions (RPF), can be used to estimate the uncertainty of decisions in autonomous driving. A method for classifying whether or not an action should be considered safe is also introduced. The performance of the ensemble RPF method is evaluated by training an agent on a highway driving scenario. It is shown that the trained agent can estimate the uncertainty of its decisions and indicate an unacceptable level when the agent faces a situation that is far from the training distribution. Furthermore, within the training distribution, the ensemble RPF agent outperforms a standard Deep Q-Network agent. In this study, the estimated uncertainty is used to choose safe actions in unknown situations. However, the uncertainty information could also be used to identify situations that should be added to the training process.**

I. Introduction

Autonomous driving has the potential to benefit society in many ways, such as increase the productivity and improve the energy efficiency of autonomous vehicles, and to reduce the number of accidents [1]. The decision-making task of an autonomous vehicle is challenging, since the system must handle a diverse set of environments, interact with other traffic participants, and consider uncertainty in the sensor information. To manually predict all situations that can occur and code a suitable behavior is infeasible. Therefore, it is compelling to consider methods that are based on machine learning to train a decision-making agent. A desired property of such an agent is that it should not just output a recommended decision, but also estimate the uncertainty of the given decision. This paper investigates a way to create a tactical[1] decision-making agent that is also aware of its limitations, for autonomous driving.

Traditional decision-making methods, which are based on predefined rules and implemented as hand-crafted state machines, were successful during the DARPA Urban Challenge [4], [5], [6]. Other classical methods treat the decision-making task as a motion planning problem [7], [8], [9]. Although these methods are successful in many cases, one drawback is that they are designed for specific driving situations, which makes it hard to scale them to the complexity of real-world driving.

Reinforcement learning (RL) techniques have been successful in various domains during the last decade [10], [11], [12]. Compared to non-learning based methods, RL approaches are general and could potentially scale to all driving situations. RL has previously been applied to decision-making for autonomous driving in simulated environments, for example Deep Q-Network (DQN) approaches for highway driving [13], [14] and intersections [15], [16], policy gradient techniques for merging situations [17], or combining Monte Carlo tree search and RL [18]. A few studies have also trained decision-making agents in a simulated environment and then deployed them in a real vehicle [19], [20], or for a limited case, trained an agent directly in a real vehicle [21].

The agents that were trained by RL in previous works can naturally only be expected to output rational decisions in situations that are close to the training distribution. However, a fundamental problem with these methods is that no matter what situation the agents are facing, they will always output a decision, with no information on the uncertainty of the decision or if the agent has experienced anything similar during its training. If, for example, an agent that was trained for a one-way highway scenario would be deployed in a scenario with oncoming traffic, it would still output a decision, without any warning. A more subtle difference could be if the agent has been trained for nominal highway driving, and suddenly faces a speeding driver or an accident that creates standstill traffic. The importance of estimating the uncertainty of decisions in autonomous driving is further emphasized by McAllister et al. [22].

A common way to model uncertainty is through Bayesian probability theory [23]. Bayesian deep learning has previously been used in the autonomous driving field for, e.g., image segmentation [24] and end-to-end learning [25]. Early work on applying Bayesian approaches to RL, for balancing the exploration vs. exploitation dilemma, was introduced by Dearden et al. [26]. More recent studies have extended this approach to deep RL, by using an ensemble of neural networks with randomized prior functions [27].

In contrast to the related work, this paper investigates an RL method for tactical decision-making in autonomous driving that can estimate the uncertainty of its decision, based

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP), funded by Knut and Alice Wallenberg Foundation, and partially by Vinnova FFI.

*Carl-Johan Hoel, Krister Wolff, and Leo Laine are with Chalmers University of Technology, Gothenburg, Sweden {carl-johan.hoel, krister.wolff, leo.laine}@chalmers.se

†Carl-Johan Hoel and Leo Laine are with Volvo Group, Gothenburg, Sweden {carl-johan.hoel, leo.laine}@volvo.com

[1]The decision-making task of an autonomous vehicle is commonly divided into strategic, tactical, and operational decision-making, also called navigation, guidance and stabilization [2], [3]. In short, tactical decisions refer to high level, often discrete, decisions, such as when to change lanes on a highway.

on the work by Osband et al. [27] (Sect. II). A criterion for when the agent is considered confident enough about its decisions is introduced (Sect. II-C). A decision-making agent is trained in a one-way highway driving scenario (Sect. III), and the results show that it outperforms both a common heuristic method and a standard DQN method (Sect. IV-A). This study shows that the presented method can estimate the uncertainty of the recommended actions, and that this information can be used to choose less risky actions in unknown situations (Sect. IV-B). Another potential use for the uncertainty estimation is to identify situations that should be added to the training process. Further properties of the presented method are discussed in Sect. V.

The main contributions of this paper are:

1) A novel application of an RL method for tactical decision-making in autonomous driving that can estimate the uncertainty of its decisions (Sect. III).
2) A criterion that determines if the trained agent is confident enough about a particular decision (Sect. II-C).
3) A performance analysis of the introduced approach in different highway driving scenarios, compared to a commonly used heuristic method and a standard DQN approach (Sect. IV).

II. Approach

This section gives a brief introduction to RL, and a description of how the uncertainty of a recommended action can be estimated. The details on how this approach was applied to autonomous driving follows in Sect. III.

A. Reinforcement learning

Reinforcement learning is a subfield of machine learning, where an agent interacts with some environment to learn a policy $\pi(s)$ that maximizes the future expected return [28]. The policy defines which action a to take in each state s. When an action is taken, the environment transitions to a new state s' and the agent receives a reward r. The reinforcement learning problem can be modeled as a Markov Decision Process (MDP), which is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is a state transition model, R is a reward model, and γ is a discount factor. At every time step t, the goal of the agent is to choose an action a that maximizes the discounted return,

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{1}$$

In Q-learning [29], the agent tries to learn the optimal action-value function $Q^*(s, a)$, which is defined as

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]. \tag{2}$$

The DQN algorithm uses a neural network with weights θ to approximate the optimal action-value function as $Q(s, a; \theta) \approx Q^*(s, a)$ [10]. Since the action-value function follows the Bellman equation, the weights can be optimized by minimizing the loss function

$$L(\theta) = \mathbb{E}_M\Big[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2\Big]. \tag{3}$$

Algorithm 1 Ensemble RPF training process

for $k \leftarrow 1$ to K
  Initialize θ_k and $\hat{\theta}_k$ randomly
  $m_k \leftarrow \{\}$
$i \leftarrow 0$
while networks not converged
  $s_i \leftarrow$ initial random state
  $k \sim \mathcal{U}\{1, K\}$
  while episode not finished
    $a_i \leftarrow \arg\max_a Q_k(s_i, a)$
    $s_{i+1}, r_i \leftarrow$ StepEnvironment(s_i, a_i)
    for $k \leftarrow 1$ to K
      if $p \sim \mathcal{U}(0, 1) < p_{\text{add}}$
        $m_k \leftarrow m_k \cup \{(s_i, a_i, r_i, s_{i+1})\}$
      $M \leftarrow$ sample from m_k
      update θ_k with SGD and loss $L(\theta_k)$
    $i \leftarrow i + 1$

The loss is calculated for a mini-batch M, and a target network θ^- is updated regularly.

B. Bayesian reinforcement learning

The DQN algorithm returns a maximum likelihood estimate of the Q-values, but gives no information about the uncertainty of the estimation. The risk of an action could be represented as the variance of the return when taking that action [30]. One line of RL research focuses on obtaining an estimate of the uncertainty by statistical bootstrap [31], [32]. An ensemble of models is then trained on different subsets of the available data and the distribution that is given by the ensemble is used to approximate the uncertainty. A better Bayesian posterior is obtained if a randomized prior function (RPF) is added to each ensemble member [27]. Then, each individual ensemble member, here indexed by k, estimates the Q-values as the sum

$$Q_k(s, a) = f(s, a; \theta_k) + \beta p(s, a; \hat{\theta}_k), \tag{4}$$

where f and p are neural networks, with parameters θ_k that can be trained, and parameters $\hat{\theta}_k$ that are kept fixed. The factor β balances the importance of the prior function. When adding the prior, the loss function of Eq. 3 is changed to

$$L(\theta_k) = \mathbb{E}_M\Big[(r + \gamma \max_{a'}(f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s', a') - (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s, a))^2\Big]. \tag{5}$$

The full ensemble RPF method, which was used in this study, is outlined in Algorithm 1. An ensemble of K trainable neural networks and K fixed prior networks are first initialized randomly. A replay memory is divided into K parallel buffers m_k, for the individual ensemble members (although in practice, this can be implemented in a memory efficient way that uses negligible more memory than a single replay memory). To handle exploration, a random ensemble member is chosen for each training episode. Actions are then taken by greedily maximizing the Q-value of the chosen ensemble member, which corresponds to a form of approximate

Fig. 1: Example of an initial state of the highway driving scenario. The ego vehicle is shown in green, whereas the color of the surrounding vehicles represent their current speed. Yellow corresponds to 15 m/s, red to 35 m/s, and the different shades of orange represent speeds in between.

Thompson sampling. The new experience (s_i, a_i, r_i, s_{i+1}) is then added to each ensemble buffer with probability p_{add}. Finally, a minibatch M of experiences is sampled from each ensemble buffer and the trainable network parameters of the corresponding ensemble member are updated by stochastic gradient descent (SGD), using the loss function in Eq. 5.

C. Uncertainty threshold

The coefficient of variation[2] $c_{\text{v}}(s, a)$ of the Q-values that are given by the neural networks can be used to estimate the agent's uncertainty of taking different actions from a given state. In this study, a hard uncertainty threshold $c_{\text{v}}^{\text{safe}}$ is used to classify if the agent is confident enough of its decision, but a progressive scale could also be used, which is further discussed in Sect. V. When $c_{\text{v}}(s, a) > c_{\text{v}}^{\text{safe}}$, action a is considered unsafe in state s, which indicates that (s, a) is far from the training distribution. The value of the parameter $c_{\text{v}}^{\text{safe}}$ can be determined by observing the performance of the agent and the variation in $c_{\text{v}}(s, a)$ for the chosen action during testing episodes within the training distribution, see Sect IV-A for further details.

When the training process is completed and the trained agent is deployed (i.e., not during the training process), the agent chooses actions by maximizing the mean Q-value of the K neural networks, under the condition $c_{\text{v}}(s, a) < c_{\text{v}}^{\text{safe}}$, i.e.,

$$\begin{aligned} &\arg\max_a \frac{1}{K}\sum_{k=1}^{K} Q_k(s, a), \\ &\text{s.t.} \quad c_{\text{v}}(s, a) < c_{\text{v}}^{\text{safe}}. \end{aligned} \tag{6}$$

If no action fulfills the criteria, a fallback action a_{safe} is used.

III. Implementation

A one-way highway driving scenario (Fig. 1) was used to train and test the presented ensemble RPF algorithm. This section describes how the scenarios were set up, how the decision-making problem was formulated as an MDP, how the neural networks were designed, and how the training process was set up. The code that was used to implement the algorithm and experiments is available on GitHub [33].

A. Simulation setup

The Simulation of Urban Mobility (SUMO) traffic simulator was used for the experiments in this study [34]. A one-way highway with three lanes was simulated, where the controlled vehicle consisted of a 16 m long truck-trailer combination, with a maximum speed of $v_{\text{max}}^{\text{ego}} = 25$ m/s. In the beginning of each episode, 25 passenger cars were inserted into the simulation, with a random desired speed in the range $[v_{\text{min}}, v_{\text{max}}] = [15, 35]$ m/s. In order to create interesting traffic situations, slower vehicles were positioned in front of the ego vehicle, and faster vehicles were placed behind the ego vehicle. Each episode was terminated after $N = 100$ time steps, or earlier if a collision occurred or the ego vehicle drove off the road. The simulation times step was set to $\Delta t = 1$ s. Fig. 1 gives an example of the initial state of an episode.

The passenger vehicles were controlled by the standard SUMO driver model, which consists of an adaptive cruise controller for the longitudinal motion [35], and a lane change model that makes tactical decisions to overtake slower vehicles [36]. In the scenarios considered here, no strategical decisions were necessary, so the strategical part of the lane changing model was turned off. Furthermore, in order to make the traffic situations more demanding, the cooperation level of the lane changing model was set to 0. Overtaking was allowed both on the left and right side of another vehicle, and each lane change took $t_{\text{lc}} = 4$ s to complete.

B. MDP formulation[3]

The decision-making problem was formulated according to the following Markov decision process.

1) State space, $\mathcal{S}$: The state of the system,

$$s = (\{x_i, y_i, v_{x,i}, v_{y,i}\}_{i \in 0,\dots,N_{\text{veh}}}), \tag{7}$$

consists of the positions (x_i, y_i) and speeds $(v_{x,i}, v_{y,i})$ of each vehicle in a traffic scene, where index 0 refers to the ego vehicle. The agent that controls the ego vehicle can observe the state of all surrounding vehicles within the distance $x_{\text{sensor}} = 200$ m.

2) Action space, $\mathcal{A}$: At every time step, the agent can choose between any combination of three longitudinal actions and three lateral actions, which consist of setting the acceleration to $\{-1, 0, 1\}$ m/s^2 and {*'stay in lane'*, *'change left'*, *'change right'*}. The final possible action is to brake hard by setting the longitudinal acceleration to -4 m/s^2. In total, this results in 10 different possible actions. Once a lane change is initiated, it cannot be aborted. The fallback action a_{safe} is set to *'stay in lane'* laterally and -4 m/s longitudinally.

[2]The coefficient of variation is also known as the relative standard deviations, which is defined as the ratio of the standard deviation to the mean.

[3]Technically, the problem is a Partially Observable Markov Decision Process (POMDP) [37], since the ego vehicle cannot observe the internal state of the driver models of the surrounding vehicles. However, the POMDP can be approximated as an MDP with a k-Markov approximation, where the state consists of the last k observations [10]. For this study, it proved sufficient to use only the last observation.

TABLE I: Input ξ to the neural network.

Ego lane	$\xi_1 = 2y_0/y_{\max} - 1$
Ego vehicle speed	$\xi_2 = 2v_{x,0}/v_{\max}^{\text{ego}} - 1$
Lane change state	$\xi_3 = \text{sgn}\,(v_{y,0})$
Relative long. position of vehicle i	$\xi_{4i+1} = (x_i - x_0)/x_{\text{sensor}}$
Relative lat. position of vehicle i	$\xi_{4i+2} = (y_i - y_0)/y_{\max}$
Relative speed of vehicle i	$\xi_{4i+3} = \frac{v_{x,i} - v_{x,0}}{v_{\max} - v_{\min}}$
Lane change state of vehicle i	$\xi_{4i+4} = \text{sgn}\,(v_{y,i})$

3) Reward model, R: The objective of the agent is to navigate through traffic in a safe and time efficient way. A simple reward model is used to achieve this goal. At every time step, the agent receives a positive reward of $1 - \frac{v_{\max} - v_0}{v_{\max}}$, which encourages a time efficient policy that, for example, overtakes slow vehicles. However, if a collision occurs, or the ego vehicle drives off the road (by changing lanes outside of the road), a negative reward of $r_{\text{col}} = -10$ is added and the episode is terminated. Furthermore, if the behavior of the ego vehicle causes another vehicle to emergency brake, defined by a deceleration with a magnitude greater than $a_{\text{e}} = -4.5$ m/s^2, or if the ego vehicle drives closer to another vehicle than a minimum time gap of $t_{\text{gap}} = 2.5$ s, a negative reward of $r_{\text{near}} = -10$ is added, but the episode is not terminated. Furthermore, in order to discourage unnecessary lane changes, a negative reward of $r_{\text{lc}} = -1$ is added when a lane change is initiated.

4) State transition model, T: The state transition probabilities are implicitly defined by the generative simulation model, and not known to the agent.

C. Neural network architecture

A neural network estimates the Q-values of the different actions in a given state. The state s is transformed to the normalized state vector ξ before it is passed to the network, where all elements $\xi_* \in [-1, 1]$. The positions and speeds of the surrounding vehicles are expressed as relative to the ego vehicle. Further details on how ξ is calculated are given in Table I.

In a previous paper [14], we introduced a one-dimensional convolutional neural network (CNN) architecture, which makes the training faster and gives better results than a standard fully connected architecture. By applying CNN layers and a max pooling layer to the part of the input that describes the surrounding vehicles, the output of the network becomes independent of the ordering of the surrounding vehicles in the input vector, and the architecture allows a varying input vector size.

The neural network architecture that was used in this study is shown in Fig. 2. Both convolutional layers have 32 filters. The size and stride of the first convolutional layer is set to four, which equals the number of state inputs of each surrounding vehicle, whereas the size and stride of the second layer is set to one. The two fully connected (FC) layers have 64 units each. Rectified linear units (ReLUs) are used as activation functions for all layers, except the last, which has a linear activation function. The architecture also includes a dueling structure that separates the state value and action advantage estimation [38].

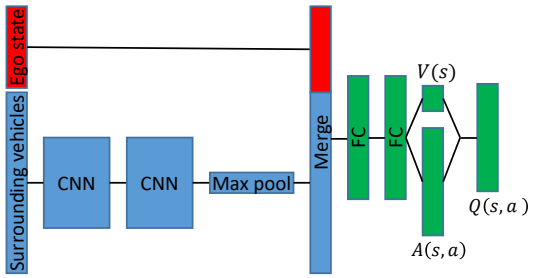


Fig. 2: The neural network architecture that was used in this study.

TABLE II: Hyperparameters of Algorithm 1 and baseline DQN.

Number of ensemble members, K	10
Prior scale factor, β	50
Experience adding probability, p_{add}	0.5
Discount factor, γ	0.99
Learning start iteration, N_{start}	50,000
Replay memory size, M_{replay}	500,000
Learning rate, η	0.0005
Mini-batch size, M_{mini}	32
Target network update frequency, N_{update}	20,000
Huber loss threshold, δ	10
Initial exploration constant, ϵ_{start}	1
Final exploration constant, ϵ_{end}	0.05
Final exploration iteration, $N_{\epsilon\text{-end}}$	1,000,000

D. Training process

Algorithm 1 was used to train the ensemble of neural networks, with the exception that the loss function of Double DQN was used, which slightly changes the maximization term of Eq. 3 to $\gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta_i^-)$ [39]. Adam [40] was used to update the parameters θ_k of the K ensemble members, and the update step was parallelized in order to speed up the process. For episodes without collisions, the last experience was not added to the replay memory, in order to trick the agent that the episodes continue forever [14]. Table II displays the hyperparameters of Algorithm 1 and the training process. Due to the computational complexity, the hyperparameter values were selected from an informal search and not a systematic grid search.

E. Baseline methods

The Double DQN algorithm was used as a baseline method (henceforth simply referred to as the DQN method). To make a fair comparison, the same hyperparameters as for Algorithm 1 were used, with additional hyperparameters for an annealed exploration schedule, given in Table II. During the testing episodes, the action with the highest Q-value was greedily chosen. The standard SUMO driver model, which is further described in Sect. III-A, was used as a second baseline method.

IV. Results

The results show that the ensemble RPF method outperforms the SUMO driver model and performs more consistently than the baseline DQN method when tested on similar scenarios as the agents were trained on. However, when the trained agents were tested on scenarios outside of the training distributions, the ensemble RPF method both indicates a high uncertainty and chooses safe actions, whereas the DQN agent causes collisions. This section presents more details on the results, together with a brief analysis and discussion on some of the characteristics of the results, whereas a more general discussion follows in Sect. V.

Both the ensemble RPF and the DQN agents were trained in a simulated environment (Sect. III). At every $50{,}000$ added training sample, henceforth called training step, the agents were evaluated on 100 different test episodes. These test episodes were randomly generated in the same way as the training episodes, but not present during the training. The test episodes were also kept identical for all the test phases and agents. The safety criterion described in Sect II-C was not active in the test episodes (since the uncertainty c_v varies during the training process), but used when the fully trained agent was exposed to unseen scenarios, see Sect. IV-B.

A. Within training distribution

Fig. 3 shows the proportion of collision free test episodes as a function of training steps for the ensemble RPF and DQN agents. For 10 random seeds, the figure also shows the mean and the standard deviation of the return during the test episodes, normalized by the return of the SUMO driver. Both the ensemble RPF and DQN agents quickly learn to brake and thereby solve all test episodes without collisions, although the DQN agent experience occasional collisions later on during the training process. With more training, both methods also learn to overtake slow vehicles and outperform the SUMO driver model. The ensemble RPF agent receives a slightly higher return and has a more stable performance compared to the DQN agent. The small variation in final performance between random seeds of the ensemble RPF agent is likely due to that a close to optimal policy has been found.

To gain insight in the uncertainty estimation during the training process, and to illustrate how to set the uncertainty threshold parameter $c_\mathrm{v}^\mathrm{safe}$ (Sect. II-C), Fig. 4 shows the coefficient of variation c_v for the chosen action during the test episodes as a function of training steps. Note that the figure shows the uncertainty of the chosen action, whereas the uncertainty for other actions could be higher. After around four million training steps, the coefficient of variation settles at around 0.01, with a small spread in values, which motivates setting $c_\mathrm{v}^\mathrm{safe} = 0.02$.

B. Outside training distribution

In order to illustrate the ability of the ensemble RPF agent to detect unseen situations, the agent that was obtained after five million training steps was deployed in scenarios that were not included in the training episodes. In various situations that involve an oncoming vehicle, the uncertainty

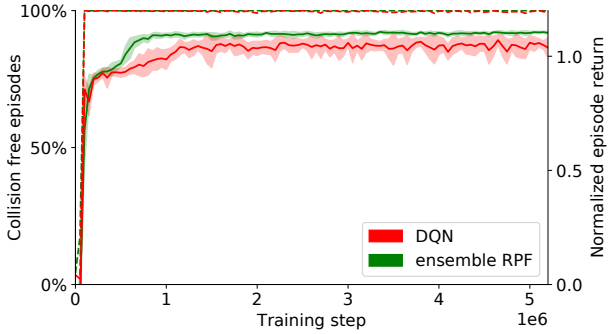


Fig. 3: Proportion of collision free test episodes (dashed), and mean normalized return over training steps for the ensemble RPF and DQN agents (solid). The shaded areas show the standard deviation for 10 random seeds.

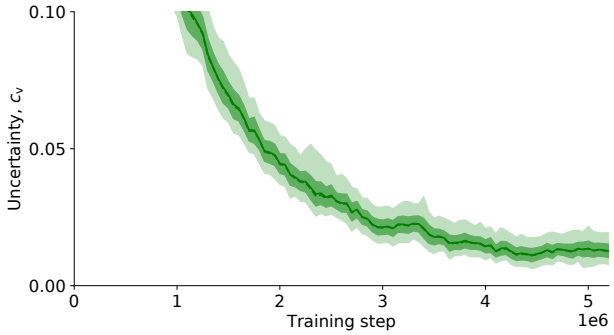
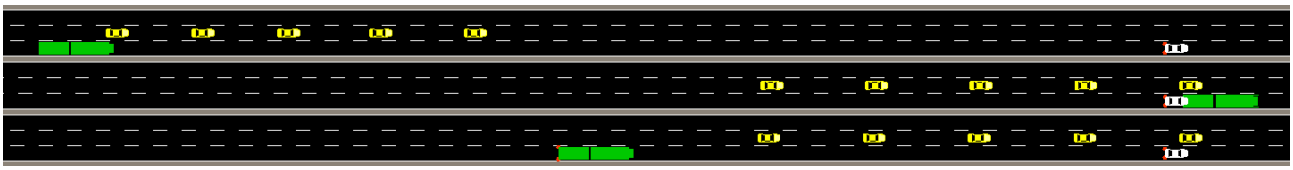


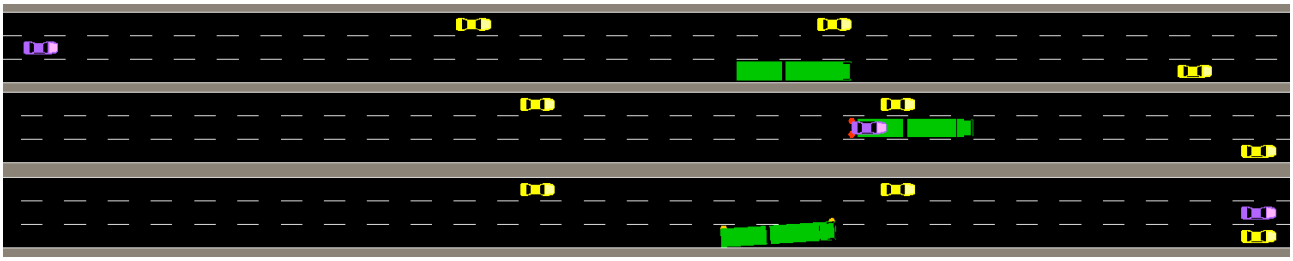
Fig. 4: Mean uncertainty, represented by the coefficient of variation c_v, of the chosen action during the test episodes. The dark shaded area represent the standard deviation and the bright shaded area represent percentiles 1 to 99.

estimate was consistently high, $c_\mathrm{v} > 0.2$. This level is one order of magnitude larger than the uncertainty threshold $c_\mathrm{v}^\mathrm{safe}$, and therefore clearly indicates that such situations are outside of the training distribution.

For a deployed agent that has been trained in one-way highway traffic, an arguably more representative situation that the agent could be exposed to involves an accident, which has caused a vehicle to stop on the highway, see Fig. 5a. As mentioned in Sect. III-A, the surrounding vehicles were simulated with a random speed in the range $[15, 35]$ m/s during the training, hence a vehicle that stands still is outside of the training distribution. The ego vehicle starts with a speed of 25 m/s and is placed in the rightmost lane, with the stopped vehicle 300 m in front of it. There are several slower vehicles in the center lane, which makes changing lanes impossible. The DQN agent does brake when it approaches the stopped vehicle, but since such a situation was not present in the training episodes, the agent does not brake early enough to avoid a collision. The ensemble RPF agent also outputs the highest Q-value for maintaining its current speed when the ego vehicle is far from the stopped vehicle, and would have collided if that action had been chosen. However, as soon as the stopped vehicle is within the ego vehicle's sensor range x_sensor, the uncertainty $c_\mathrm{v} > c_\mathrm{v}^\mathrm{safe}$, see Fig. 6. Since this uncertainty level indicates that the situation is



(a) Accident situation with a stopped vehicle, shown in white.



(b) Situation with a speeding vehicle, shown in purple.

Fig. 5: Two situations that are outside of the training distribution and cause collisions if the confidence of the agent is not considered. The top panel of each figure shows the initial state, whereas the two bottom panels show the state for the DQN and ensemble RPF agents, after 12 s in (a) and 7 s in (b).

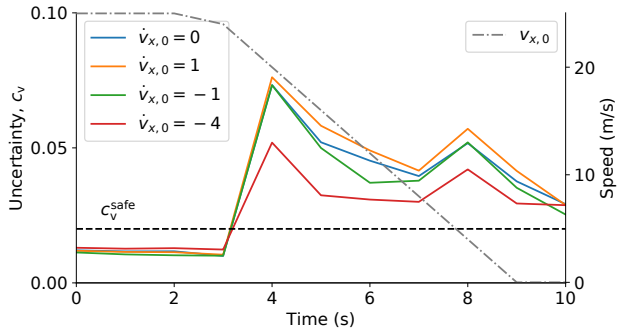


Fig. 6: Uncertainty c_v for the *'stay in lane'* action with different accelerations, and ego vehicle speed $v_{x,0}$, during the case with a stopped vehicle. The uncertainty for the other possible actions are orders of magnitude larger. Due to the sensor range limitation, the stopped vehicle is not observed until $t = 4$ s.

outside of the training distributions, the agent chooses to brake hard (Sect. II-C) early enough to avoid a collision.

The trained agent was also tested in a situation that involves a speeding vehicle, see Fig. 5b, where a vehicle that drives at 55 m/s is approaching the ego vehicle from behind, in the neighboring lane. A slow vehicle is positioned in front of the ego vehicle, which creates an incentive to overtake it on the left. The DQN agent does change lanes to the left, but then causes a collisions since the speeding vehicle cannot brake fast enough. The ensemble RPF agent also estimates the highest Q-values for the lane changing actions, but estimates $c_\text{v} > 0.025$ for changing lanes, which is larger than the uncertainty threshold c_v^safe, until the speeding vehicle has passed. However, $c_v \approx 0.015$ for staying in the lane and braking with -1 m/s^2, which the agent then does, according to the policy that was described in Sect. II-C. When the speeding vehicle has passed, the ego vehicle changes lanes, since c_v has then decreased to below the threshold.

Videos of the presented scenarios and additional situations, together with the code that was used to obtain the results, are available on GitHub [33].

V. Discussion

The results show that the ensemble RPF algorithm can be used to train an agent that is aware of the uncertainty of its decisions. The ensemble RPF agents outperforms both the DQN agent and the heuristic SUMO driver model within the training distribution (Fig. 3). In addition, the ensemble RPF agent can also indicate its uncertainty when the agent is exposed to situations that are far from the training distribution (Fig. 5). In this study, the uncertainty information was used to choose safe actions, by prohibiting actions with a level of uncertainty that exceeds a defined threshold. However, to formally guarantee functional safety with a learning-based method is challenging and likely requires an underlying safety layer in a real application [41]. While the presented approach could reduce the activation frequency of such a safety layer, a possibly even more important application could be to improve the learning process. The uncertainty information could be used to guide the training to situations that the agent is currently not confident about, which could improve the sample efficiency and broaden the distribution of situations that the agent can handle. Furthermore, if an agent is trained in a simulated environment and later deployed in real traffic, the uncertainty information could be used to detect situations that need to be added to the simulated world.

Since a simple safety function was used in this study, a hard uncertainty threshold c_v^safe was used to determine if the agent is confident enough to take a particular action. If a more advanced safety function would receive the information from the agent, it could be beneficial to instead output a continuous confidence measure. One option is to define such a confidence measure as $1 - \frac{c_\text{v}(s,a) - c_\text{v}^\text{min}}{c_\text{v}^\text{safe} - c_\text{v}^\text{min}}$, where c_v^min is a parameter that defines the minimum uncertainty. A value of 1 would then indicate maximum confidence, and values below 0 would be considered unsafe.

The main disadvantage of using the ensemble RPF method compared to DQN is the higher computational cost, since K neural networks need to be trained instead of one. However, the design of the algorithm allows an efficient parallelization of the training process, which in practice reduces the effect. Both agents were trained on a desktop computer, where the DQN agent required 36 hours and the ensemble RPF agent required 72 hours to complete five million training steps. Osband et al. reports that the difference can be further reduced to 20% in wall-time with their implementation [32].

VI. Conclusion

The advantage of using Bayesian RL compared to standard RL for tactical decision-making in autonomous driving has been demonstrated in this paper. The ensemble RPF method learns to make more efficient decisions and it has a more stable performance compared to the DQN method within the training distribution. Outside of the training distribution, the ensemble RPF method is aware of the high uncertainty and can fall back to taking safe actions, in order to avoid collisions. However, since the DQN agent does not possess the uncertainty information, collisions occur in unknown situations.

A possibly even more important aspect when having information on what the agent knows and does not know is that the training can be adapted accordingly. To investigate this further is a topic for future work. The performance of the proposed method will also be further evaluated in different types of traffic situations in a future paper.

References

[1] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transp. Res. Part A: Policy and Pract.*, vol. 77, pp. 167–181, 2015.

[2] J. A. Michon, "A critical view of driver behavior models: What do we know, what should we do?" in *Human Behav. and Traffic Saf.*, L. Evans and S. R.C., Eds. Boston, MA: Springer US, 1985, pp. 485–524.

[3] S. Ulbrich *et al.*, "Towards a functional system architecture for automated vehicles," *CoRR*, vol. abs/1703.08557, 2017.

[4] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, 2008.

[5] M. Montemerlo *et al.*, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[6] A. Bacha *et al.*, "Odin: Team VictorTango's entry in the DARPA urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 467–492, 2008.

[7] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *IEEE Int. Conf. on Robot. and Automat.*, 2010, pp. 987–993.

[8] P. Nilsson, L. Laine, N. van Duijkeren, and B. Jacobson, "Automated highway lane changes of long vehicle combinations: A specific comparison between driver model based control and non-linear model predictive control," in *Int. Symp. on Innov. in Intell Syst. and Appl. (INISTA)*, 2015, pp. 1–8.

[9] F. Damerow and J. Eggert, "Risk-aversive behavior planning under multiple situations with uncertainty," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2015, pp. 656–663.

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[12] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[13] P. Wang, C. Chan, and A. d. L. Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *IEEE Int. Veh. Symp. (IV)*, 2018, pp. 1379–1384.

[14] C. J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2018, pp. 2148–2155.

[15] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep Q-learning," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2018, pp. 3169–3174.

[16] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *IEEE Int. Conf. on Robot. and Automat. (ICRA)*, 2018, pp. 2034–2039.

[17] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *CoRR*, vol. abs/1610.03295, 2016.

[18] C. J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," *IEEE Trans. on Intell. Veh.*, 2019.

[19] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017.

[20] M. Bansal, A. Krizhevsky, and A. S. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," *Robot: Sci. & Syst. (RSS)*, 2019.

[21] A. Kendall *et al.*, "Learning to drive in a day," in *IEEE Int. Conf. on Robot. and Automat. (ICRA)*, 2019, pp. 8248–8254.

[22] R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning," in *Proc. of the 26th Int. Joint Conf. on Artif. Intell.*, 2017, pp. 4745–4753.

[23] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[24] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017, pp. 57.1–57.12.

[25] R. Michelmore, M. Kwiatkowska, and Y. Gal, "Evaluating uncertainty quantification in end-to-end autonomous driving control," *CoRR*, vol. abs/1811.06817, 2018.

[26] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," in *Proc. of the 15th Nat/10th Conf. on Artif. Intell./Innov. Appl. of Artif. Intell.*, 1998, p. 761–768.

[27] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," in *Adv. in Neural Inf. Process. Syst. 31*, 2018, pp. 8617–8629.

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[29] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992.

[30] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. of Mach. Learn. Res.*, vol. 16, no. 42, pp. 1437–1480, 2015.

[31] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*. Soc. for Ind. and Appl. Math., 1982.

[32] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Adv. in Neural Inf. Process. Syst. 29*, 2016, pp. 4026–4034.

[33] C. J. Hoel, "Source code for 'Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation'," 2020. [Online]. Available: https://github.com/carljohanhoel/BayesianRLForAutonomousDriving

[34] P. A. Lopez *et al.*, "Microscopic traffic simulation using SUMO," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2018, pp. 2575–2582.

[35] S. Krauss, P. Wagner, and C. Gawron, "Metastable states in a microscopic model of traffic flow," *Phys. Rev. E*, vol. 55, pp. 5597–5602, May 1997.

[36] J. Erdmann, "Lane-changing model in SUMO," *Proc. of the SUMO2014 Model. Mob. with Open Data*, vol. 24, pp. 77–88, 2014.

[37] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.

[38] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. of the 33rd Int. Conf. on Mach. Learn.*, vol. 48, 2016, pp. 1995–2003.

[39] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. of the 39th AAAI Conf. on Artif. Intell.*, 2016, pp. 2094–2100.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Int. Conf. on Learn. Repr.*, 12 2014.

[41] S. Underwood, D. Bartz, A. Kade, and M. Crawford, "Truck automation: Testing and trusting the virtual driver," in *Road Veh. Automat. 3*, G. Meyer and S. Beiker, Eds. Springer, 2016, pp. 91–109.

Paper V

Reinforcement Learning with Uncertainty Estimation for Tactical Decision-Making in Intersections

in

Proceedings of the 23rd IEEE International Conference on Intelligent Transportation Systems, Virtual conference, 2020, pp. 1-7.

Reinforcement Learning with Uncertainty Estimation for Tactical Decision-Making in Intersections

Carl-Johan Hoel[*,1,3,4], Tommy Tram[*,2,3,4] and Jonas Sjöberg[3]

Abstract— **This paper investigates how a Bayesian reinforcement learning method can be used to create a tactical decision-making agent for autonomous driving in an intersection scenario, where the agent can estimate the confidence of its decisions. An ensemble of neural networks, with additional randomized prior functions (RPF), are trained by using a bootstrapped experience replay memory. The coefficient of variation in the estimated Q-values of the ensemble members is used to approximate the uncertainty, and a criterion that determines if the agent is sufficiently confident to make a particular decision is introduced. The performance of the ensemble RPF method is evaluated in an intersection scenario and compared to a standard Deep Q-Network method, which does not estimate the uncertainty. It is shown that the trained ensemble RPF agent can detect cases with high uncertainty, both in situations that are far from the training distribution, and in situations that seldom occur within the training distribution. This work demonstrates one possible application of such a confidence estimate, by using this information to choose safe actions in unknown situations, which removes all collisions from within the training distribution, and most collisions outside of the distribution.**

I. Introduction

To make safe, efficient, and comfortable decisions in intersections is one of the challenges of autonomous driving. A decision-making agent needs to handle a diverse set of intersection types and layouts, interact with other traffic participants, and consider uncertainty in sensor information. The fact that around 40% of all traffic accidents during manual driving occur in intersections indicates that decision-making in intersections is a complex task [1]. To manually predict all situations that can occur and tailor a suitable behavior is not feasible. Therefore, a data-driven approach that can learn to make decisions from experience is a compelling approach. A desired property of such a machine learning approach is that it should also be able to indicate how confident the resulting agent is about a particular decision.

Reinforcement learning (RL) provides a general approach to solve decision-making problems [2], and could potentially scale to all types of driving situations. Promising results have been achieved in simulation by applying a Deep Q-Network (DQN) agent to intersection scenarios [3], [4], and highway driving [5], [6], or a policy gradient method to a lane merging situation [7]. Some studies have trained an RL agent in a simulated environment and then deployed the agent in a real vehicle [8], [9], and for a limited case, trained the agent directly in a real vehicle [10].

Generally, a fundamental problem with the RL methods in previous work is that the trained agents do not provide any confidence measure of their decisions. For example, if an agent that was trained for a highway driving scenario would be exposed to an intersection situation, it would still output a decision, although it would likely not be a good one. A less extreme example involves an agent that has been trained in an intersection scenario with nominal traffic, and then faces a speeding driver. McAllister et al. further discuss the importance of estimating the uncertainty of decisions in autonomous driving [11].

A common way of estimating uncertainty is through Bayesian probability theory [12]. Bayesian deep learning has previously been used to estimate uncertainty in autonomous driving for image segmentation [10] and end-to-end learning [13]. Dearden et al. introduced Bayesian approaches to RL that balances the trade off between exploration and exploitation [14]. In recent work, this approach has been extended to deep RL, by using an ensemble of neural networks [15]. However, these studies focus on creating an efficient exploration method for RL, and do not provide a confidence measure for the agents' decisions.

This paper investigates an RL method that can estimate the uncertainty of the resulting agent's decisions, applied to decision-making in an intersection scenario. The RL method uses an ensemble of neural networks with randomized prior functions that are trained on a bootstrapped experience replay memory, which gives a distribution of estimated Q-values (Sect. II). The distribution of Q-values is then used to estimate the uncertainty of the recommended action, and a criterion that determines the confidence level of the agent's decision is introduced (Sect. II-C). The method is used to train a decision-making agent in different intersection scenarios (Sect. III), in which the results show that the introduced method outperforms a DQN agent within the training distribution. The results also show that the ensemble method can detect situations that were not present in the training process, and thereby choose safe fallback actions in such situations (Sect. IV). Further characteristics of the introduced method is discussed in Sect. V. This work is an extension to a recent paper, where we introduced the mentioned method, but applied to a highway driving scenario [16].

[*] Both authors contributed equally to this work.

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems, and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, and partially by Vinnova FFI.

[1] Affiliated with Volvo Group, Gothenburg, Sweden.

[2] Affiliated with Zenuity AB, Gothenburg, Sweden.

[3] Affiliated with Chalmers University of Technology, Gothenburg, Sweden. {carl-johan.hoel,tommy.tram,jonas.sjoberg}@chalmers.se

[4] Affiliated with AI Innovation of Sweden, Gothenburg, Sweden.

II. Approach

This section gives a brief introduction to RL, describes how the uncertainty of an action can be estimated by an ensemble method, and introduces a measure of confidence for different actions. Further details on how this approach was applied to driving in an intersection scenario follows in Sect. III.

A. Reinforcement learning

Reinforcement learning is a branch of machine learning, where an agents explores an environment and tries to learn a policy $\pi(s)$ that maximizes the future expected return, based on the agent's experiences [2]. The policy determines which action a to take in a given state s. The state of the environment will then transitions to a new state s' and the agent receives a reward r. A Markov Decision Process (MDP) is often used to model the reinforcement learning problem. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is a state transition model, R is a reward model, and γ is a discount factor. At each time step t, the agent tries to maximize the future discounted return

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{1}$$

In a value-based branch of RL called Q-learning [17], the objective of the agent is to learn the optimal state-action value function $Q^*(s, a)$. This function is defined as the expected return when the agent takes action a from state s and then follow the optimal policy π^*, i.e.,

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]. \tag{2}$$

The Q-function can be estimated by a neural network with weights θ, i.e., $Q(s, a) \approx Q(s, a; \theta)$. The weights are optimized by minimizing the loss function

$$L(\theta) = \mathbb{E}_M\left[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2\right], \tag{3}$$

which is derived from the Bellman equation. The loss is obtained from a mini-batch M of training samples, and θ^- represents the weights of a target network that is updated regularly. More details on the DQN algorithm are presented by Mnih et al. [18].

B. Bayesian reinforcement learning

One limitation of the DQN algorithm is that only the maximum likelihood estimate of the Q-values is returned. The risk of taking a particular action can be approximated as the variance in the estimated Q-value [19]. One approach to obtain a variance estimation is through statistical bootstrapping [20], which has been applied to the DQN algorithm [21]. The basic idea is to train an ensemble of neural network on different subsets of the available replay memory. The ensemble will then provide a distribution of Q-values, which can be used to estimate the variance. Osband et al. extended the ensemble method by adding a randomized prior function (RPF) to each ensemble member, which gives a better Bayesian posterior [15]. The Q-values of each ensemble member k is then calculated as the sum of two neural networks, f and p, with equal architecture, i.e.,

$$Q_k(s, a) = f(s, a; \theta_k) + \beta p(s, a; \hat{\theta}_k). \tag{4}$$

Here, the weights θ_k of network f are trainable, and the weights $\hat{\theta}_k$ of the prior network p are fixed to the randomly initialized values. A parameter β scales the importance of the networks. With the two networks, the loss function in Eq. 3 becomes

$$L(\theta_k) = \mathbb{E}_M\Big[(r + \gamma \max_{a'}(f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s', a') - (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s, a))^2\Big]. \tag{5}$$

Algorithm 1 Ensemble RPF training process

```
for k ← 1 to K
    Initialize θ_k and θ̂_k randomly
    m_k ← {}
i ← 0
while networks not converged
    s_i ← initial random state
    ν ~ U{1, K}
    while episode not finished
        a_i ← arg max_a Q_ν(s_i, a)
        s_{i+1}, r_i ← StepEnvironment(s_i, a_i)
        for k ← 1 to K
            if p ~ U(0, 1) < p_add
                m_k ← m_k ∪ {(s_i, a_i, r_i, s_{i+1})}
        M ← sample mini-batch from m_k
        update θ_k with SGD and loss L(θ_k)
        i ← i + 1
```

Algorithm 1 outlines the complete ensemble RPF method, which was used in this work. An ensemble of K trainable and prior neural networks are first initialized randomly. Each ensemble member is also assigned a separate experience replay memory buffer m_k (although in a practical implementation, the replay memory can be designed in such a way that it uses negligible more memory than a shared buffer). For each new training episode, a uniformly sampled ensemble member, $\nu \sim \mathcal{U}\{1, K\}$, is used to greedily select the action with the highest Q-value. This procedure handles the exploration vs. exploitation trade-off and corresponds to a form of approximate Thompson sampling. Each new experience $e = (s_i, a_i, r_i, s_{i+1})$ is then added to the separate replay buffers m_k with probability p_{add}. Finally, the trainable weights of each ensemble member are updated by uniformly sample a mini-batch M of experiences and using stochastic gradient descent (SGD) to backpropagate the loss of Eq. 5.

C. Confidence criterion

The agent's uncertainty in choosing different actions can be defined as the coefficient of variation[1] $c_{\text{v}}(s, a)$ of the

[1]Ratio of the standard deviation to the mean.

Q-values of the ensemble members. In previous work, we introduced a confidence criterion that disqualifies actions with $c_{\mathrm{v}}(s,a) > c_{\mathrm{v}}^{\mathrm{safe}}$, where c_{safe} is a hard threshold [16]. The value of the threshold should be set so that (s,a) combinations that are contained in the training distribution are accepted, and those which are not will be rejected. This value can be determined by observing values of c_{v} in testing episodes within the training distribution, see Sect. IV-A for further details.

When the agent is fully trained (i.e., not during the training phase), the policy chooses actions by maximizing the mean of the Q-values of the ensemble members, with the restriction $c_{\mathrm{v}}(s,a) < c_{\mathrm{v}}^{\mathrm{safe}}$, i.e.,

$$\begin{aligned} &\arg\max_a \frac{1}{K}\sum_{k=1}^{K} Q_k(s,a), \\ &\text{s.t.} \quad c_{\mathrm{v}}(s,a) < c_{\mathrm{v}}^{\mathrm{safe}}. \end{aligned} \tag{6}$$

In a situation where no possible action fulfills the confidence criterion, a fallback action a_{safe} is chosen.

III. Implementation

The ensemble RPF method, which can obtain an uncertainty estimation of different actions, is tested on different intersection scenarios. In this work, the uncertainty information is used to reject unsafe actions and reduce the number of collisions. This section describes how the simulation of the scenarios is set up, how the decision-making problem is formulated as an MDP, the architecture of the neural networks, and the details on how the training is performed.

A. Simulation setup

The simulated environment consists of different intersection scenarios, and is based on previous work [22]. For completeness, an overview is presented here. Each episode starts by randomly selecting a single or bi-directional intersection, shown in Fig. 1, and placing the ego vehicle to the left with a random distance $p_{\mathrm{e}}^{\mathrm{c},j}$ to the intersection and a speed of 10 m/s. A random number N of other vehicles are positioned along the top and bottom roads with a random distance $p_o^{\mathrm{c},j}$ to the intersection, and a random desired speed v_{d}^{j}. The other vehicles follow the Intelligent Driver Model (IDM) [23], with a set time gap of $t_{\mathrm{d}}^{j} = 1$ s. One quarter of the vehicles stop at the intersection and three quarters continue through the intersection, regardless of the behavior of the ego vehicle. When a vehicle has passed the intersection and reached the end of the road, it is moved back to the other side of the intersection, which creates a constant traffic flow. The simulator is updated at 25 Hz, and decisions are taken at 4 Hz. The goal of the ego vehicle is to reach a position that is located 10 m to the right of the last crossing point.

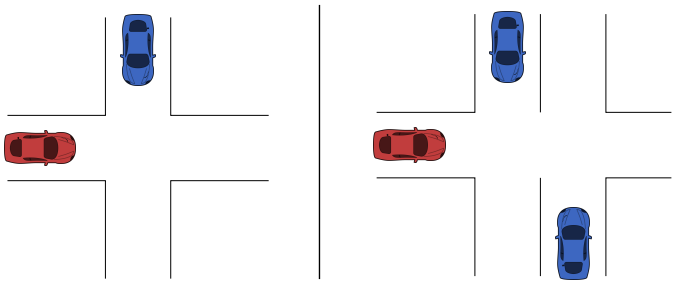


Fig. 1: The two intersection scenarios considered in this work; single directional to the left and bidirectional to the right. The agent controls the red car.

TABLE I: Parameters for simulator

Number of other vehicles, N	$\{1,2,3,4\}$
Starting position ego, $p_{\mathrm{e}}^{\mathrm{c},j}$	$[50,60]$ m
Starting position target, $p_{\mathrm{o}}^{\mathrm{c},j}$	$[10,55]$ m
Desired velocity, v_{d}^{j}	$[8,12]$ m/s

B. MDP formulation[2]

The following Markov decision process is used to model the decision-making problem.

1) State space, $\mathcal{S}$: The design of the state of the system,

$$s = (p_{\mathrm{e}}^{\mathrm{g}}, v_{\mathrm{e}}, a_{\mathrm{e}}, \{p_{\mathrm{e}}^{\mathrm{s},j}, p_{\mathrm{e}}^{\mathrm{c},j}, p_{\mathrm{o}}^{\mathrm{s},j}, p_{\mathrm{o}}^{\mathrm{c},j}, v_{\mathrm{o}}^{j}, a_{\mathrm{o}}^{j}\}_{j\in 0,\ldots,N}), \tag{7}$$

allows the description of intersections with different layouts [4]. The state, illustrated in Fig. 2, consists of the distance from the ego vehicle to the goal $p_{\mathrm{e}}^{\mathrm{g}}$, the velocity and acceleration of the ego vehicle, v_{e}, a_{e}, and the other vehicles, v_{o}^{j}, a_{o}^{j}, where j denotes the index of the other vehicles. Furthermore, $p_{\mathrm{e}}^{\mathrm{s},j}$ and $p_{\mathrm{e}}^{\mathrm{c},j}$ are the distances from the ego vehicle to the start of the intersection and crossing point, relative to target vehicle j respectively. The distances $p_{\mathrm{o}}^{\mathrm{s},j}$ and $p_{\mathrm{o}}^{\mathrm{c},j}$ are the distance from the other vehicles to the start of the intersection and the crossing point.

2) Action space, $\mathcal{A}$: The action space consists of six tactical decisions: {*'take way'*, *'give way'*, *'follow car* $\{1,\ldots,4\}$*'*}, which set the target of the IDM controller. The *'take way'* action treats the situation as an empty road, whereas the *'give way'* action sets a target distance of $p_e^{s,j}$ and a target speed of 0 m/s. The *'follow car j'* actions sets the target distance to $p_{\mathrm{e}}^{\mathrm{c},j} - p_{\mathrm{o}}^{\mathrm{c},j}$ and target speed to v_{o}^{j}. In cases where $p_{\mathrm{o}}^{\mathrm{c},j} > p_{\mathrm{e}}^{\mathrm{c},j}$, the target distance is set to a value that corresponds to timegap 0.5 s. The output of the IDM model is further limited by a maximum jerk $j_{\max} = 5$ m/s^3 and maximum acceleration $a_{\max} = 5$ m/s^2. If less than four vehicles are present, the actions that correspond to choosing an absent vehicle are pruned by using Q-masking [25].

[2]The full state is not directly observable, since the intentions of the surrounding vehicles are not known to the agent. Therefore, the problem is a Partially Observable Markov Decision Process (POMDP) [24]. However, by using a k-Markov approximation, where the state consists of the k last observations, the POMDP can be approximated as an MDP [18]. For the scenarios that were considered in this work, it proved sufficient to simply use the last observation.

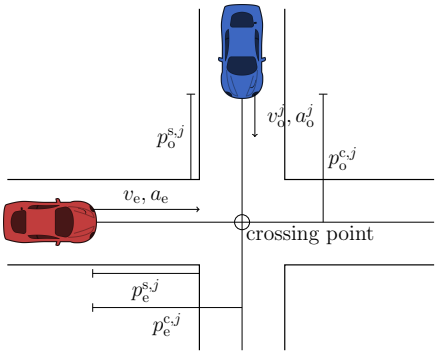


Fig. 2: The state space definitions for a single crossing scenario, where subscript e and o denotes ego and other vehicle, respectively.

3) Reward model, R: The objective of the agent is to reach the goal on the other side of the intersection, without colliding with other vehicles and for comfort reasons, with as little jerk j_t as possible. Therefore, the reward at each time step r_t is defined as

$$r_t = \begin{cases} 1 & \text{at reaching the goal,} \\ -1 & \text{at a collision,} \\ -\left(\frac{j_t}{j_{\max}}\right)^2 \frac{\Delta\tau}{\tau_{\max}} & \text{at non-terminating steps.} \end{cases}$$

The non-terminating reward is scaled with the maximum time of an episode, $\tau_{\max}$, and the step time $\Delta\tau = 0.04$ s, to ensure $\sum_{t=0}^{t=\tau_{\max}} \in [-1, 0]$. Further details about the reward function can be found in previous research [22].

4) Transition model, T: The state transition probabilities are not known to the agent. However, the true transition model is defined by the simulation model, described in Sect. III-A.

C. Fallback action

As mentioned in Sect. II-C, a fallback action a_{safe} is used when $c_{\text{v}} > c_{\text{v}}^{\text{safe}}$ for all available actions. This fallback action is set to *'give way'*, with the difference that no jerk limitation is applied and with a higher acceleration limit $a_{\max} = 10$ m/s^2.

D. Network architecture

In previous studies, we have showed that a network architecture that applies the same weights to the input that describes the surrounding vehicles results in a better performance and speeds up the training process [6], [4]. Such an architecture can be constructed by applying a one-dimensional convolutional neural network (CNN) structure to the surrounding vehicles' input. The network architecture that is used in this work is shown in Fig. 3. The first convolutional layer has 32 filters, with size and stride set to six, which equals the number of state inputs of each surrounding vehicle, and the second convolutional layers has 16 filter, with size and stride set to one. The fully connected (FC) layer that is connected to the ego vehicle input has 16 units, and the joint fully connected layer has

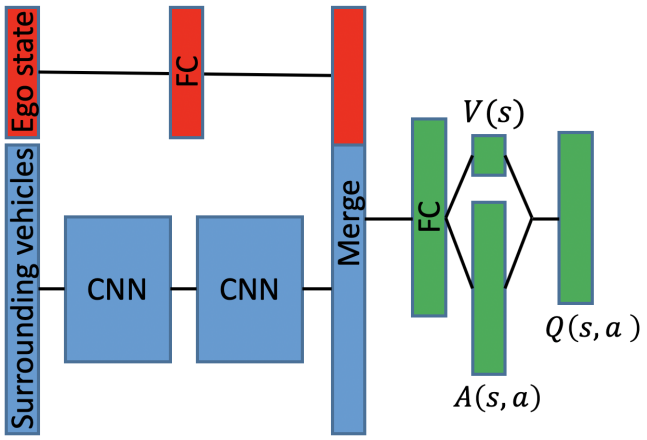


Fig. 3: The neural network architecture that was used in this work.

64 units. All layers use rectified linear units (ReLUs) as activation functions, except for the last layer, which has a linear activation function. The final dueling structure of the network separates the estimation of the state value $V(s)$ and the action advantage $A(s,a)$ [26]. The input vector is normalized to the range $[-1, 1]$. The input vector contains slots for four surrounding vehicles, and if less vehicles are present in the traffic scene, the empty input is set to -1.

E. Training process

Algorithm 1 is used to train the agent. The loss function of Double DQN is applied, which subtly modifies the maximization operation of Eq. 3 to $\gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta_i^-)$ [27]. The Adam optimizer is used to update the weights [28], and K parallel workers are used for the backpropagation step. The hyperparameters of the training process are shown in Table II, and the values were selected by an informal search, due to the computational complexity.

If the current policy of the agent decides to stop the ego vehicle, an episode could continue forever. Therefore, a timeout time is set to $\tau_{\max} = 20$ s, at which the episode terminates. The last experience of such an episode is not added to the replay memory. This trick prevents the agent to learn that an episode can end due to a timeout, and makes it seem like an episode can continue forever, which is important, since the terminating state due to the time limit is not part of the MDP [6].

F. Baseline method

The Double DQN method, hereafter simply referred to as the DQN method, is used as a baseline. For a fair comparison, the same hyperparameters as for the ensemble RPF method is used, with the addition of an annealing ϵ-greedy exploration schedule, which is shown in Table II. During test episodes, a greedy policy is used.

TABLE II: Hyperparameters of Algorithm 1 and baseline DQN.

Number of ensemble members, K	10
Prior scale factor, β	1
Experience adding probability, p_{add}	0.5
Discount factor, γ	0.99
Learning start iteration, N_{start}	50,000
Replay memory size, M_{replay}	500,000
Learning rate, η	0.0005
Mini-batch size, M_{mini}	32
Target network update frequency, N_{update}	20,000
Huber loss threshold, δ	10
Initial exploration constant, ϵ_{start}	1
Final exploration constant, ϵ_{end}	0.05
Final exploration iteration, $N_{\epsilon\text{-end}}$	1,000,000

IV. Results

The results show that the ensemble RPF method outperforms the DQN method, both in terms of training speed and final performance, when the resulting agents are tested on scenarios that are similar to the training scenarios. When the fully trained ensemble RPF agent is exposed to situations that are outside of the training distribution, the agent indicates a high uncertainty and chooses safe actions, whereas the DQN agent collides with other vehicles. More details on the characteristics of the results are presented and briefly discussed in this section, whereas a more general discussion follows in Sect. V.

The ensemble RPF and DQN agents were trained in the simulated environment that was described in Sect. III. After every 50,000 training steps, the performance of the agents were evaluated on 100 random test episodes. These test episodes were randomly generated in the same way as the training episodes, but kept fixed for all the evaluation phases.

A. Within training distribution

The average return and the average proportion of episodes where the ego vehicle reached the goal, as a function of number of training steps, is shown in Fig. 4, for the test episodes. The figure also shows the standard deviation for 5 random seeds, which generates different sets of initial parameters of the networks and different training episodes, whereas the test episodes are kept fixed. The results show that the ensemble RPF method both learns faster, yields a higher return, and causes less collisions than the DQN method.

Fig. 5 shows how the coefficient of variation c_{v} of the chosen action varies during the testing episodes. Note that the uncertainty of actions that are not chosen can be higher, which is often the case. After around one million training steps, the average value of c_{v} settles at around 0.04, with a 99 percentile value of 0.15, which motivates the choice of setting $c_{\text{v}}^{\text{safe}} = 0.2$.

As shown in Fig. 4, occasional collisions still occur during the test episodes when deploying the fully trained ensemble RPF agent. The reasons for these collisions are further discussed in Sect. V. In one particular example of a collision, the agent fails to brake early enough and ends up in an impossible situation, where it collides with another vehicle in the intersection. However, the estimated uncertainty increases significantly during the time before the collision, when the incorrect actions are taken, see Fig. 6. When applying the confidence criterion (Sect. II-C), the agent instead brakes early enough, and can thereby avoid the collision. The confidence criterion was also applied to all the test episodes, which removed all collisions.

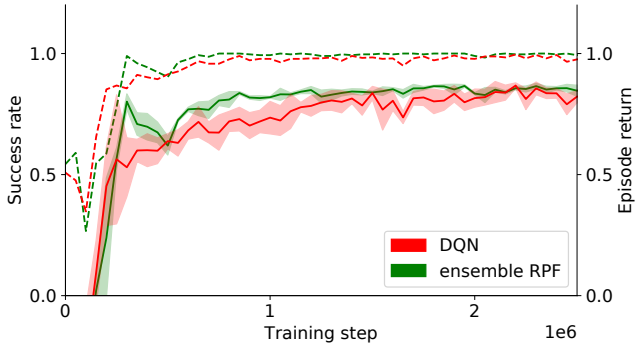


Fig. 4: Proportion of test episodes where the ego vehicle reached its goal (dashed), and episode return (solid), over training steps for the ensemble RPF and DQN methods. The shaded areas show the standard deviation for 5 random seeds.

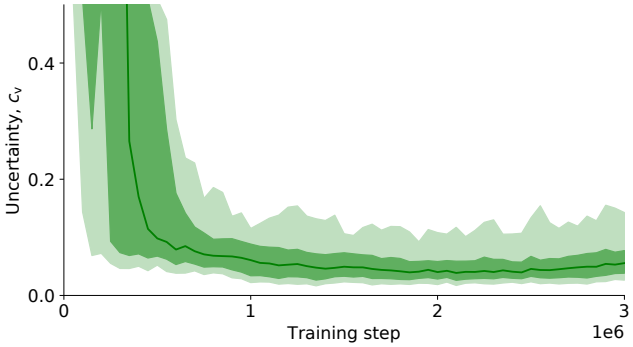


Fig. 5: Mean coefficient of variation c_{v} for the chosen action during the test episodes. The dark shaded area shows percentiles 10 to 90, and the bright shaded area shows percentiles 1 to 99.

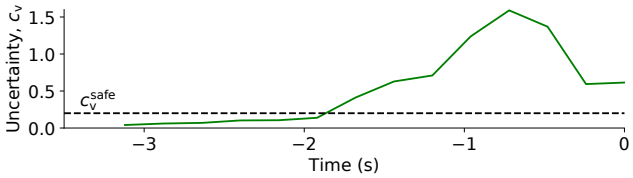
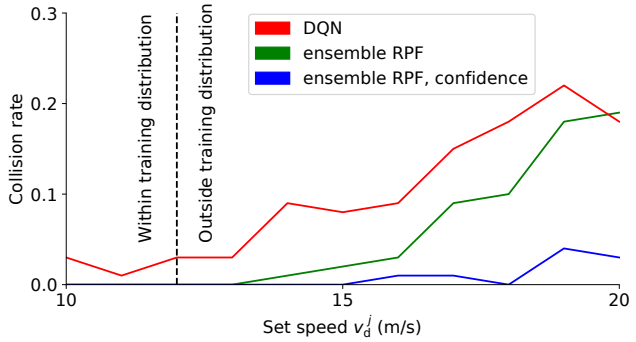
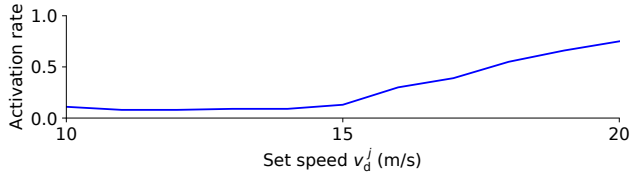


Fig. 6: Uncertainty c_{v} during the time steps before one of the collisions in the test episodes, within the training distribution. The collision occurs at $t = 0$ s.

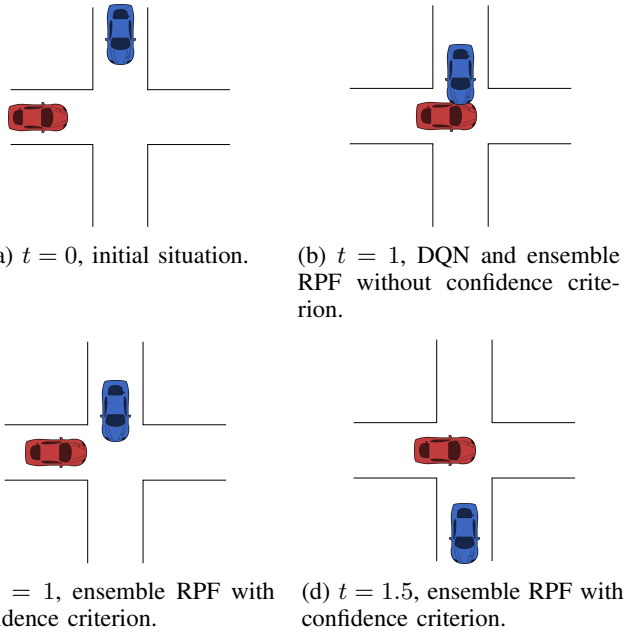


(a) Proportion of collisions.



(b) Proportion of episodes where a_{safe} was used at least once.

Fig. 7: Performance of the ensemble RPF agent, with and without the confidence criterion, and the DQN agent, in test episodes with different set speeds v_{d}^j for the surrounding vehicles.



(a) $t = 0$, initial situation.

(b) $t = 1$, DQN and ensemble RPF without confidence criterion.

(c) $t = 1$, ensemble RPF with confidence criterion.

(d) $t = 1.5$, ensemble RPF with confidence criterion.

Fig. 8: Example of a situation outside of the training distribution, where there would be a collision if the confidence criterion is not used. The vehicle at the top is here approaching the crossing at 20 m/s.

B. Outside training distribution

The ensemble RPF agent that was obtained after three million training steps was tested in scenarios outside of the training distribution, in order to evaluate the agent's ability to detect unseen situations. The same testing scenarios as for within the distribution was used, with the exception that the speed of the surrounding vehicles was set to a single deterministic value, which was varied during different runs in the range $v_{\mathrm{d}}^j = [10, 20]$ m/s. The proportion of collisions as a function of set speed of the surrounding vehicles is shown in Fig. 7, together with the proportion of episodes where the confidence criterion was violated at least once. The figure shows that when the confidence criterion is used, most of the collisions can be avoided. Furthermore, the violations of the criterion increase when the speed of the surrounding vehicles increase, i.e., the scenarios move further from the training distribution.

An example of a situation that causes a collision is shown in Fig. 8, where an approaching vehicle drives with a speed of 20 m/s. The Q-values of both the trained ensemble RPF and DQN agents indicate that the agents expect to make it over the crossing before the other vehicle. However, since the approaching vehicle drives faster than what the agents have seen during the training, a collision occurs. When the confidence criterion is applied, the uncertainty rises to $c_{\mathrm{v}} > c_{\mathrm{v}}^{\mathrm{safe}}$ for all actions when the ego vehicle approaches the critical region, where it has to brake in order to be able to stop, and a collision is avoided by choosing action a_{safe}.

V. Discussion

The results show that the ensemble RPF method can indicate an elevated uncertainty for situations that the agent has been insufficiently trained for, both within and outside of the training distribution. In previous work by the authors of this paper, we observed similar results when using the ensemble RPF method to estimate uncertainty outside of the training distribution in a highway driving scenario [16]. In contrast, this paper shows that, in some cases, the ensemble RPF method can even detect situations with high uncertainty within the training distribution. Such situations include rare events that seldom or never occur during the training process, which makes it hard for the agent to provide an accurate estimate of the Q-values for the corresponding states. Since these states are seldom used to update the neural networks of the ensemble, the weights of the trainable networks will not adapt to the respective prior networks, and the uncertainty measure c_{v} will remain high for these rare events. This information is useful to detect edge cases within the training set and indicate when the decision of the trained agent is not fully reliable.

In this work, the estimated uncertainty is used to choose a safe fallback action if the uncertainty exceeds a threshold value. For the cases that are considered here, this confidence criterion removes all collisions within the training distribution, and almost all collisions when the speed of the surrounding vehicles is increased to levels outside of the training distribution. However, to guarantee safety by using a learning-based method is challenging, and an underlying safety layer is often used [29]. The presented method could decrease the number of activations of such a safety layer,

but possibly more importantly, the uncertainty measure could also be used to guide the training process to focus on situations that the current agent needs to explore further. Moreover, if an agent is trained in simulation and then deployed in real traffic, the uncertainty estimation of the agent could detect situations that should be added to the simulated world, in order to better match real-world driving.

The results show that the ensemble RPF method performs better and more stable than a standard DQN method within the training distribution. The main disadvantage is the increased computational complexity, since K neural networks need to be trained. This disadvantage is somewhat mitigated in practice, since the design of the algorithm allows an efficient parallelization. Furthermore, the tuning complexity of the ensemble RPF and DQN methods are similar. Hyperparameters for the number of ensemble members K and prior scale factor β are introduced, but the parameters that control the exploration of DQN are removed.

VI. Conclusion

The results of this paper demonstrates the usefulness of using a Bayesian RL technique for tactical-decision making in an intersection scenario. The ensemble RPF method can be used to estimate the confidence of the recommended actions, and the results show that the trained agent indicates high uncertainty for situations that are outside of the training distribution. Importantly, the method also indicates high uncertainty for rare events within the training distribution. In this work, the confidence information was used to choose a safe action in situations with high uncertainty, which removed all collisions from within the training distribution, and most of the collisions in situations outside of the training distribution.

The uncertainty information could also be used to identify situations that are not known to the agent, and guide the training process accordingly. To investigate this further is a topic for future work. Another subject for future work involves how to set the parameter value c_v^{safe} in a more systematic way, and how to automatically update the value during training.

References

[1] "Traffic safety facts," National Highway Traffic Safety Administration, Tech. Rep. DOT HS 812 261, 2014.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[3] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *IEEE Int. Conf. on Robot. and Automat. (ICRA)*, 2018, pp. 2034–2039.

[4] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep Q-learning," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2018, pp. 3169–3174.

[5] P. Wang, C. Chan, and A. d. L. Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *IEEE Int. Veh. Symp. (IV)*, 2018, pp. 1379–1384.

[6] C. J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2018, pp. 2148–2155.

[7] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *CoRR*, vol. abs/1610.03295, 2016.

[8] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017.

[9] M. Bansal, A. Krizhevsky, and A. S. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," *Robot: Sci. & Syst. (RSS)*, 2019.

[10] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017, pp. 57.1–57.12.

[11] R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning," in *Proc. of the 26th Int. Joint Conf. on Artif. Intell.*, 2017, pp. 4745–4753.

[12] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[13] R. Michelmore, M. Kwiatkowska, and Y. Gal, "Evaluating uncertainty quantification in end-to-end autonomous driving control," *CoRR*, vol. abs/1811.06817, 2018.

[14] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," in *Proc. of the 15th Nat/10th Conf. on Artif. Intell./Innov. Appl. of Artif. Intell.*, 1998, p. 761–768.

[15] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," in *Adv. in Neural Inf. Process. Syst. 31*, 2018, pp. 8617–8629.

[16] C. J. Hoel, K. Wolff, and L. Laine, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," submitted to IEEE Intell. Veh. Symp. (IV) 2020.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992.

[18] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. of Mach. Learn. Res.*, vol. 16, no. 42, pp. 1437–1480, 2015.

[20] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*. Soc. for Ind. and Appl. Math., 1982.

[21] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Adv. in Neural Inf. Process. Syst. 29*, 2016, pp. 4026–4034.

[22] T. Tram, I. Batkovic, M. Ali, and J. Sjöberg, "Learning when to drive in intersections by combining reinforcement learning and model predictive control," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, Oct 2019, pp. 3263–3268.

[23] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, 2000.

[24] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.

[25] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical Decision Making for Lane Changing with Deep Reinforcement Learning," *Neural Information Processing Systems (NIPS)*, 2017.

[26] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. of the 33rd Int. Conf. on Mach. Learn.*, vol. 48, 2016, pp. 1995–2003.

[27] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. of the 39th AAAI Conf. on Artif. Intell.*, 2016, pp. 2094–2100.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Int. Conf. on Learn. Repr.*, 12 2014.

[29] S. Underwood, D. Bartz, A. Kade, and M. Crawford, "Truck automation: Testing and trusting the virtual driver," in *Road Veh. Automat. 3*, G. Meyer and S. Beiker, Eds. Springer, 2016, pp. 91–109.

Paper VI

Ensemble Quantile Networks: Uncertainty-Aware Reinforcement Learning with Applications in Autonomous Driving

submitted to

IEEE Transactions on Neural Networks and Learning Systems, 2021.

Ensemble Quantile Networks: Uncertainty-Aware Reinforcement Learning with Applications in Autonomous Driving

Carl-Johan Hoel, Krister Wolff, and Leo Laine

***Abstract*—Reinforcement learning (RL) can be used to create a decision-making agent for autonomous driving. However, previous approaches provide only black-box solutions, which do not offer information on how confident the agent is about its decisions. An estimate of both the aleatoric and epistemic uncertainty of the agent's decisions is fundamental for real-world applications of autonomous driving. Therefore, this paper introduces the Ensemble Quantile Networks (EQN) method, which combines distributional RL with an ensemble approach, to obtain a complete uncertainty estimate. The distribution over returns is estimated by learning its quantile function implicitly, which gives the aleatoric uncertainty, whereas an ensemble of agents is trained on bootstrapped data to provide a Bayesian estimation of the epistemic uncertainty. A criterion for classifying which decisions that have an unacceptable uncertainty is also introduced. The results show that the EQN method can balance risk and time efficiency in different occluded intersection scenarios, by considering the estimated aleatoric uncertainty. Furthermore, it is shown that the trained agent can use the epistemic uncertainty information to identify situations that the agent has not been trained for and thereby avoid making unfounded, potentially dangerous, decisions outside of the training distribution.**

***Index Terms*—Reinforcement learning, aleatoric uncertainty, epistemic uncertainty, autonomous driving, decision-making.**

I. Introduction

A DECISION-MAKING agent for an autonomous vehicle needs to handle a diverse set of environments and situations, while interacting with other traffic participants and considering uncertainty. A machine learning approach for creating a general decision-making agent is compelling, since it is not feasible to manually predict all situations that can occur and code a suitable behavior for each and every one of them. However, a drawback of learning-based agents is that they typically provide a black-box solution, which only outputs a decision for a given situation. It would be desirable if the agent also could provide an estimate of its confidence level, or equivalently, the estimated uncertainty of its decisions.

Uncertainty can be divided into two categories: aleatoric and epistemic uncertainty [1], [2], where both are important to consider in many decision-making problems. Aleatoric uncertainty refers to the inherent randomness of an outcome and can therefore not be reduced by observing more data. For example, when approaching an occluded intersection, there is an aleatoric uncertainty in if, or when, another vehicle will enter the intersection. To estimate the aleatoric uncertainty is important, since such information can be used to make risk-aware decisions. Contrarily, epistemic uncertainty arises due to a lack of knowledge and can be reduced by observing more data. For example, epistemic uncertainty appears if a decision-making agent has been trained to only handle 'normal' driving situations and then faces a speeding driver or an accident. An estimate of the epistemic uncertainty provides insight into which situations the trained agent does not know how to handle and can be used to increase the safety [3]. The epistemic uncertainty estimate could also be used to concentrate the training process to situations where the agent needs more training [4].

Reinforcement learning (RL) provides a learning-based approach to create decision-making agents, which could potentially scale to all driving situations. Many recent studies have applied RL to autonomous driving, for example, by using the Deep Q-Network (DQN) algorithm in intersections and highway situations [5]–[7], by using a policy gradient method for lane merging [8], or combining RL with Monte Carlo tree search [9]. A majority of these studies perform both the training and evaluation in simulated environments, whereas some train the agent in simulations and then apply the trained agent in the real world [10], [11], or for some limited cases, the training itself is also performed in the real world [12]. Overviews of RL for autonomous driving are given by Kiran et al. [13] and by Zhu et al. [14]. However, previous studies do not estimate the aleatoric or the epistemic uncertainty of the decision that the trained agent recommends. One exception is the study by Bernhard et al., where a distributional RL approach is used to create a risk-sensitive decision-making agent [15]. However, the method is not applied in a theoretically consistent way and can therefore cause arbitrary decisions, which is further discussed in Sect. V of this paper.

Bayesian probability theory can be used to estimate the epistemic uncertainty [16]. In the autonomous driving field, Bayesian deep learning has been used for, e.g., image segmentation [17] and end-to-end learning [18]. For RL, Bayesian techniques have been used to balance the exploration vs. exploitation trade-off [19], and more recent work has addressed similar problems in deep RL [20]. Furthermore, the aleatoric uncertainty of a decision can be obtained through distributional RL, which aims to model the distribution over returns, instead of only the mean return, as in standard RL [21], [22]. For

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems, and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation

C. J. Hoel and L. Laine are with Chalmers University of Technology, Gothenburg, Sweden and with Volvo Group, Gothenburg, Sweden (e-mail: {carl-johan.hoel, leo.laine}@chalmers.se).

K. Wolff is with Chalmers University of Technology, Gothenburg, Sweden (e-mail: krister.wolff@chalmers.se)

example, Bellemare et al. introduced a method for estimating the probability of a discrete set of returns [23], which also has been further developed for continuous control tasks [24].

In contrast to the related work, this paper presents methods for training an RL agent for autonomous driving, in which the trained agent provides an estimate of the epistemic and the aleatoric uncertainty of its decisions. The epistemic uncertainty estimate is obtained through a Bayesian RL approach, which extends and further analyses the approach from two studies by the authors of this paper [25], [26]. This method, based on the work by Osband et al. [20], uses an ensemble of neural networks with additive random prior functions to obtain a posterior distribution over the expected return (Sect. II-C). The aleatoric uncertainty is obtained through a distributional RL approach, based on the work by Dabney et al. [27], which estimates the probability distribution over returns by implicitly learning its quantile function[1]. This method also allows the agent to be trained in a risk-aware manner (Sect. II-B). Furthermore, this paper introduces the Ensemble Quantile Networks (EQN) method, which combines the two previously mentioned approaches, in order to provide a complete uncertainty estimate of both the aleatoric and epistemic uncertainty of an agent's decisions (Sect. II-D). The performance of the proposed methods is tested and analyzed in different intersection scenarios (Sect. III), where the results show that while they outperform the standard DQN method, the epistemic uncertainty estimate can be used to choose less risky actions in unknown situations, and the distributional risk-aware approach allows a trade-off between risk and time efficiency (Sect. IV). Another potential use for the epistemic uncertainty information is to identify situations that should be added to the training process. Further properties of the proposed approaches are discussed in Sect. V. The code that was used to implement the different algorithms and the simulated scenarios is available on GitHub [28].

The main contributions of this paper are:

1) Methods for estimating either the aleatoric or the epistemic uncertainty of a trained agent, together with confidence criteria, which can be used to identify situations with high uncertainty (Sect. II-B2, II-C1).
2) The introduction of the EQN algorithm, which simultaneously quantifies both the aleatoric and the epistemic uncertainty of a trained agent (Sect. II-D).
3) A detailed description of how the proposed methods can be applied to an autonomous driving setting (Sect. III).
4) A qualitative and quantitative performance analysis of the proposed methods for different intersection scenarios (Sect. IV).

II. Approach

This section first gives a brief introduction to RL and its notation, followed by a description of how an aleatoric and epistemic uncertainty estimate can be obtained. The details on how these approaches can be applied to driving in an intersection scenario follows in Sect. III.

[1]The quantile function is the inverse of the cumulative distribution function for a continuous random variable.

A. Reinforcement learning

Reinforcement learning is a branch of machine learning, where an agent learns a policy $\pi(s)$ from interacting with an environment [29]. The policy describes which action a to take in state s. The environment then transitions to a new state s' and the agent receives a reward r. The decision-making problem that the RL agent tries to solve is often modeled as a Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, R, T, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, R is a reward model, T is the state transition model, and γ is a discount factor. The goal of the agent is to maximize the expected future discounted return $\mathbb{E}[R_t]$, for every time step t, where

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{1}$$

The value of taking action a in state s and then following policy π is defined by the state-action value function

$$Q^\pi(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi], \tag{2}$$

where the Q-values for the optimal policy π^* are defined as $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. The DQN algorithm aims to approximate the optimal state-action value function Q^* by a neural network with weights θ, such that $Q(s,a;\theta) \approx Q^*(s,a)$ [30]. Based on the Bellman equation, the temporal difference (TD) error

$$\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \tag{3}$$

is used to optimize the weights by iteratively minimizing the loss function $L_{\text{DQN}}(\theta) = \mathbb{E}_M[\delta_t^2]$. The loss is calculated for a mini-batch M of experiences, where each experience consists of the tuple (s_t, a_t, r_t, s_{t+1}), and the network weights θ are updated by stochastic gradient descent (SGD). Finally, θ^- is a target network that is updated regularly.

B. Aleatoric uncertainty estimation

In contrast to Q-learning, distributional RL aims to learn not only the expected return, but the distribution over returns [23]. This distribution is represented by the random variable $Z^\pi(s,a) = R_t$, given $s_t = s$, $a_t = a$, and policy π, where the mean is the traditional value function, i.e., $Q^\pi(s,a) = \mathbb{E}[Z^\pi(s,a)]$. The distribution over returns represents the aleatoric uncertainty of the outcome, which can be used to estimate the risk in different situations and to train an agent in a risk-sensitive manner.

The implicit quantile networks (IQN) approach [27] to distributional RL uses a neural network to implicitly represent the quantile function $F_Z^{-1}(\tau)$ of the random variable Z and then update the weights of the network with quantile regression. For ease of notation, define $Z_\tau := F_Z^{-1}(\tau)$, and note that for $\tau \sim \mathcal{U}(0,1)$ the sample $Z_\tau(s,a) \sim Z(s,a)$. The TD-error for two quantile samples, $\tau, \tau' \sim \mathcal{U}(0,1)$, is

$$\delta_t^{\tau,\tau'} = r_t + \gamma Z_{\tau'}\big(s_{t+1}, \pi^*(s_{t+1}); \theta^-\big) - Z_\tau(s_t, a_t; \theta), \tag{4}$$

Algorithm 1 IQN training process

Initialize θ randomly
$m \leftarrow \{\}$
$t \leftarrow 0$
while network not converged
  $s_t \leftarrow$ initial random state
  while episode not finished
    if $e \sim \mathcal{U}(0,1) < \epsilon$
      $a_t \leftarrow$ random action
    else
      $\tau_1, \ldots, \tau_{K_\tau} \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0, \alpha)$
      $a_t \leftarrow \arg\max_a \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} Z_{\tau_k}(s_t, a)$
    $s_{t+1}, r_t \leftarrow$ StepEnvironment(s_t, a_t)
    $m \leftarrow m \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    $M \leftarrow$ sample from m
    update θ with SGD and loss $L_{\text{IQN}}(\theta)$
    $t \leftarrow t + 1$

where $\pi^*(s) = \arg\max_a Q(s,a)$. A sample-based estimate of $\pi^*(s)$ is obtained from K_τ samples of $\tilde{\tau} \sim \mathcal{U}(0,1)$, as

$$\tilde{\pi}(s) = \arg\max_a \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} Z_{\tilde{\tau}_k}(s, a; \theta). \tag{5}$$

For a pair of quantiles τ, τ', the quantile Huber regression loss [31], with threshold κ, is calculated as

$$\rho_\kappa(\delta_t^{\tau,\tau'}) = |\tau - \mathbb{I}\{\delta_t^{\tau,\tau'} < 0\}| \frac{\mathcal{L}_\kappa(\delta_t^{\tau,\tau'})}{\kappa}. \tag{6}$$

Here, $\mathcal{L}_\kappa(\delta_t^{\tau,\tau'})$ is the Huber loss [32], defined as

$$\mathcal{L}_\kappa(\delta_t^{\tau,\tau'}) = \begin{cases} \frac{1}{2}(\delta_t^{\tau,\tau'})^2, & \text{if } |\delta_t^{\tau,\tau'}| \le \kappa, \\ \kappa(|\delta_t^{\tau,\tau'}| - \frac{1}{2}\kappa), & \text{otherwise,} \end{cases} \tag{7}$$

which gives a smooth gradient as $\delta_t^{\tau,\tau'} \to 0$. The full loss function $L_{\text{IQN}}(\theta)$ is obtained from a mini-batch M of sampled experiences, in which the quantiles τ and τ' are sampled N and N' times, respectively, according to

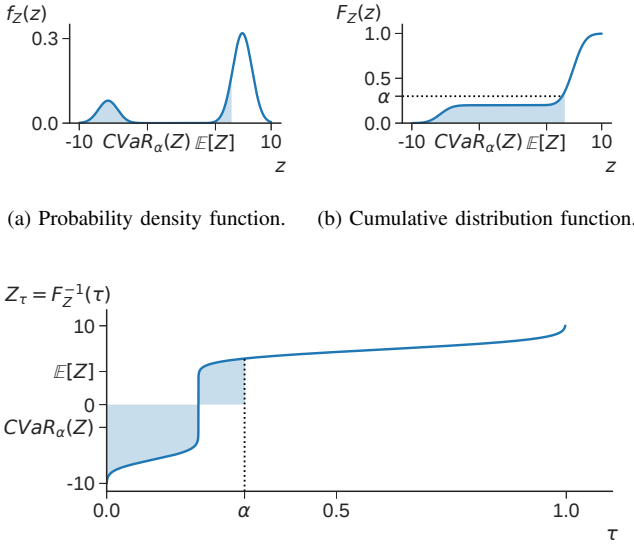
$$L_{\text{IQN}}(\theta) = \mathbb{E}_M \left[\frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_\kappa \left(\delta_t^{\tau_i, \tau_j'} \right) \right]. \tag{8}$$

The full training process of the IQN method is outlined in Algorithm 1.

1) Risk-sensitive RL: In the present context, risk refers to the aleatoric uncertainty of the potential outcome of an action. Eq. 5 represents a risk-neutral policy, which maximizes the Q-values. An alternative risk-averse policy is obtained by instead choosing the action that maximizes the conditional value-at-risk (CVaR) [33], where

$$\text{CVaR}_\alpha(Z(s,a)) = \mathbb{E}_{\tilde{\tau} \sim U([0,\alpha])}[Z_{\tilde{\tau}}(s,a)]. \tag{9}$$

The CVaR approach selects actions that maximize the mean outcome of quantiles less than α, which is graphically illustrated in Fig. 1. A detailed description of the CVaR approach and its use in solving MDPs is presented by Chow et al. [34]. Majumdar et al. further discuss the use of different distortion risk measures in robotics [35].



(a) Probability density function. (b) Cumulative distribution function.

(c) Quantile function.

Fig. 1. Illustration of the CVaR_α risk measure. The shaded regions represent quantiles $\tau \in [0, \alpha]$, here for $\alpha = 0.3$.

2) Uncertainty criterion: Dabney et al. show that the IQN method can achieve state-of-the-art results on the Atari-57 benchmark and reason about the performance of risk-sensitive training for a few of the Atari games [27]. However, as introduced in this paper, the estimated distribution over returns of the fully trained IQN agent can also be used to quantify the aleatoric uncertainty of a decision. One such uncertainty measure is the variance of the estimated returns for the evenly distributed sample set $\tau_\sigma = \{i/K_\tau \mid i \in [1, K_\tau]\}$. A threshold σ_{a}^2 can then be defined, such that the agent classifies a decision with a higher variance in returns as uncertain. In this study, the benefit of the introduced uncertainty classification is demonstrated by choosing a predefined backup policy $\pi_{\text{backup}}(s)$ if the sample variance is higher than the threshold, i.e., the fully trained agent follows the policy

$$\pi_{\sigma_{\text{a}}}(s) = \begin{cases} \arg\max_a \mathbb{E}_{\tau_\sigma}[Z_\tau(s,a)], & \text{if } \text{Var}_{\tau_\sigma}[Z_\tau(s,a)] < \sigma_{\text{a}}^2, \\ \pi_{\text{backup}}(s), & \text{otherwise.} \end{cases} \tag{10}$$

C. Epistemic uncertainty estimation

The DQN algorithm gives a maximum likelihood estimate of the Q-values, and the IQN algorithm outputs a maximum likelihood estimate of the distribution over returns. However, neither of these algorithms considers the epistemic uncertainty of the recommended actions. Statistical bootstrapping [36] can be used to train an ensemble of neural networks on different subsets of the available data, which provides a distribution over the estimated Q-values [4]. A better Bayesian posterior can be obtained by adding a randomized prior function (RPF) to each ensemble member, which creates a larger output diversity outside of the training distribution [20]. The Q-values

Algorithm 2 Ensemble RPF training process

for $k \leftarrow 1$ to K
  Initialize θ_k and $\hat{\theta}_k$ randomly
  $m_k \leftarrow \{\}$
$t \leftarrow 0$
while networks not converged
  $s_t \leftarrow$ initial random state
  $\nu \sim \mathcal{U}\{1, K\}$
  while episode not finished
    $a_t \leftarrow \arg\max_a Q_\nu(s_t, a)$
    $s_{t+1}, r_t \leftarrow$ StepEnvironment(s_t, a_t)
    for $k \leftarrow 1$ to K
      if $p \sim \mathcal{U}(0,1) < p_{\text{add}}$
        $m_k \leftarrow m_k \cup \{(s_t, a_t, r_t, s_{t+1})\}$
      $M \leftarrow$ sample from m_k
      update θ_k with SGD and loss $L_{\text{RPF}}(\theta_k)$
    $t \leftarrow t + 1$

of ensemble member k is then given by the sum of two neural networks, f and p, with identical architecture, i.e.,

$$Q_k(s,a) = f(s,a;\theta_k) + \beta p(s,a;\hat{\theta}_k). \tag{11}$$

The parameters θ_k are trained, whereas, importantly, the parameters of the prior function $\hat{\theta}_k$ are fixed during the training process. A hyperparameter β scales the relative importance of the networks. The additional prior network of the RPF method gives a slightly modified TD-error compared to the DQN method (Eq. 3), which results in the loss function

$$L_{\text{RPF}}(\theta_k) = \mathbb{E}_M\Big[(r_t + \gamma \max_a (f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s_{t+1}, a) - (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s_t, a_t))^2\Big]. \tag{12}$$

Algorithm 2 outlines the training process of the ensemble RPF method. An ensemble of K prior and trainable networks is first initialized randomly. Each ensemble member is also assigned an individual experience replay buffers m_k (although in a practical implementation, the replay buffers can be constructed such that they use negligible more memory than a single shared buffer). For each new training episode, an ensemble member ν is chosen uniformly at random and is then used to greedily select the actions with the highest Q-values throughout the episode. This procedure, which corresponds to an approximate Thompson sampling of the actions, efficiently balances the exploration vs. exploitation trade-off. Each new experience, (s_t, a_t, r_t, s_{t+1}), is added to the individual replay buffers m_k with probability p_{add}. The trainable parameters θ_k of each ensemble member are then updated through SGD, using a mini-batch M of experiences from the corresponding replay buffer and the loss function in Eq. 12. Finally, when the training process is finished and the agent is tested, the trained agent applies a policy which maximizes the mean Q-value of all the ensemble members.

1) Uncertainty criterion: Osband et al. illustrate the efficient exploration properties of the ensemble RPF algorithm [20], but do not use the estimated distribution over Q-values further. In a similar approach as for the aleatoric uncertainty (Sect. II-B2), the variance of the estimated Q-values of the ensemble RPF agent can be used to quantify the epistemic uncertainty of a decision, which we introduced in a recent paper [25]. With this approach, decisions that has a higher variance than a predefined threshold σ_{e}^2 are classified as uncertain. The benefit of the epistemic uncertainty classification is here demonstrated by choosing a predefined backup policy $\pi_{\text{backup}}(s)$ if the sample variance is higher than the threshold, which means that a trained agent follows the policy

$$\pi_{\sigma_{\text{e}}}(s) = \begin{cases} \arg\max_a \mathbb{E}_k[Q_k(s,a)], & \text{if } \text{Var}_k[Q_k(s,a)] < \sigma_{\text{e}}^2, \\ \pi_{\text{backup}}(s), & \text{otherwise.} \end{cases} \tag{13}$$

Further applications of an epistemic uncertainty classification are discussed in Sect. V.

D. Aleatoric and epistemic uncertainty estimation

A complete uncertainty estimation of both the aleatoric and the epistemic uncertainty can be obtained by combining the properties of the IQN and ensemble RPF methods into a new algorithm, which we call the Ensemble Quantile Networks method. An agent that is trained by the EQN method can then take actions that consider both the inherent uncertainty of the outcome and the model uncertainty in each situation.

As the name suggests, the EQN method uses an ensemble of networks, where each ensemble member k individually estimates the distribution over returns as

$$Z_{k,\tau}(s,a) = f_\tau(s,a;\theta_k) + \beta p_\tau(s,a;\hat{\theta}_k). \tag{14}$$

Similarly as for the RPF method, f_τ and p_τ are neural networks with identical architecture, θ_k are trainable network parameters, whereas the parameters $\hat{\theta}_k$ are fixed. The TD-error of ensemble member k and two quantile samples, $\tau, \tau' \sim \mathcal{U}(0,1)$, is

$$\delta_{k,t}^{\tau,\tau'} = r_t + \gamma Z_{k,\tau'}(s_{t+1}, \tilde{\pi}_k(s_{t+1})) - Z_{k,\tau}(s_t, a_t), \tag{15}$$

where $\tilde{\pi}_k(s) = \arg\max_a \frac{1}{K_\tau} \sum_{j=1}^{K_\tau} Z_{k,\tilde{\tau}_j}(s,a)$ is a sample-based estimate of the optimal policy. Quantile Huber regression is applied to a mini-batch of experiences, which gives the loss function

$$L_{\text{EQN}}(\theta_k) = \mathbb{E}_M\left[\frac{1}{N'}\sum_{i=1}^{N}\sum_{j=1}^{N'} \rho_\kappa\left(\delta_{k,t}^{\tau_i,\tau_j'}\right)\right]. \tag{16}$$

For each new training episode, the agent follows the policy $\tilde{\pi}_\nu(s)$ of a randomly selected ensemble member ν. The full training process of the EQN agent is outlined in Algorithm 3.

1) Uncertainty criterion: The EQN agent provides an estimate of both the aleatoric and epistemic uncertainty, based on the variance of the returns and the variance of the Q-values. The agent is considered confident about a decision if $\text{Var}_{\tau_\sigma}[\mathbb{E}_k[Z_{k,\tau}(s,a)]] < \sigma_{\text{a}}^2$ and $\text{Var}_k[\mathbb{E}_{\tau_\sigma}[Z_{k,\tau}(s,a)]] < \sigma_{\text{e}}^2$. The trained agent then follows the policy

$$\pi_{\sigma_{\text{a}},\sigma_{\text{e}}}(s) = \begin{cases} \arg\max_a \mathbb{E}_k[\mathbb{E}_{\tau_\sigma}[Z_{k,\tau}(s,a)]], & \text{if confident,} \\ \pi_{\text{backup}}(s), & \text{otherwise.} \end{cases} \tag{17}$$

Algorithm 3 EQN training process

for $k \leftarrow 1$ to K
  Initialize θ_k and $\hat{\theta}_k$ randomly
  $m_k \leftarrow \{\}$
$t \leftarrow 0$
while networks not converged
  $s_t \leftarrow$ initial random state
  $\nu \sim \mathcal{U}\{1, K\}$
  while episode not finished
    $\tau_1, \ldots, \tau_{K_\tau} \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0, \alpha)$
    $a_t \leftarrow \arg\max_a \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} Z_{\nu,\tau_k}(s_t, a)$
    $s_{t+1}, r_t \leftarrow$ StepEnvironment(s_t, a_t)
    for $k \leftarrow 1$ to K
      if $p \sim \mathcal{U}(0,1) < p_{\text{add}}$
        $m_k \leftarrow m_k \cup \{(s_t, a_t, r_t, s_{t+1})\}$
      $M \leftarrow$ sample from m_k
      update θ_k with SGD and loss $L_{\text{EQN}}(\theta_k)$
    $t \leftarrow t + 1$

III. Implementation

The presented algorithms, for estimating the aleatoric or epistemic uncertainty of an agent, are tested in simulated intersection scenarios in this study. However, these algorithms provide a general approach and could in principle be applied to any type of driving scenarios. This section describes how the different test scenarios are set up, the MDP formulation of the decision-making problem, the design of the neural network architecture, and the details of the training process.

A. Simulation setup

Two occluded intersection scenarios are used in this study, shown in Fig. 4a and 7a. The first scenario includes sparse traffic and aims to illustrate the qualitative difference between risk-neutral and risk-averse policies. The second scenario includes dense traffic and is used to compare the different algorithms, both qualitatively and quantitatively. The scenarios were parameterized to create complicated traffic situations, where an optimal policy has to consider both the occlusions and the intentions of the other vehicles, sometimes drive through the intersection at a high speed, and sometimes wait at the intersection for an extended period of time.

The Simulation of Urban Mobility (SUMO) was used to run the simulations [37]. The controlled ego vehicle, a 12 m long truck, aims to pass the intersection, in which it must yield to the crossing traffic. In each episode, the ego vehicle starts $s_{\text{start}} = 200$ m south from the intersection, with its desired speed $v_{\text{set}} = 15$ m/s. Passenger cars are randomly inserted into the simulation from the east and west end of the road network, with an average rate of $\rho_{\text{s}} = 0.1$ and $\rho_{\text{d}} = 0.5$ inserted vehicles per second for the sparse and dense traffic scenarios, respectively. The cars intend to either cross the intersection or turn to the right. The desired speed of the cars is uniformly distributed in the range $[v_{\min}, v_{\max}] = [10, 15]$ m/s, and the longitudinal speed is controlled by the standard SUMO speed controller (which is a type of adaptive cruise controller, based on the intelligent driver model (IDM) [38]), with the exception that the cars ignore the presence of the ego vehicle. Normally, the crossing cars would brake to avoid a collision with the ego vehicle, even when the ego vehicle violates the traffic rules and does not yield. With this exception, however, more collisions occur, which gives a more distinct quantitative difference between different policies. Each episode is terminated when the ego vehicle has passed the intersection, when a collision occurs, or after $N_{\max} = 100$ simulation steps. The simulations use a step size of $\Delta t = 1$ s.

Note that the setup of these scenarios includes two important sources of randomness in the outcome for a given policy, which the aleatoric uncertainty estimation should capture. From the viewpoint of the ego vehicle, a crossing vehicle can appear at any time until the ego vehicle is sufficiently close to the intersection, due to the occlusions. Furthermore, there is uncertainty in the underlying driver state of the other vehicles, most importantly in the intention of going straight or turning to the right, but also in the desired speed.

Epistemic uncertainty is introduced by a separate test, in which the trained agent faces situations outside of the training distribution. In these test episodes, the maximum speed $v_{\max}$ of the surrounding vehicles are gradually increased from 15 m/s (which is included in the training episodes) to 25 m/s. To exclude effects of aleatoric uncertainty in this test, the ego vehicle starts in the non-occluded region close to the intersection, with a speed of 7 m/s.

B. MDP formulation

The following Markov decision process describes the decision-making problem.

1) State space, $\mathcal{S}$: The state of the system,

$$s = (\{x_i, y_i, v_i, \psi_i\}_{i \in 0,\ldots,N_{\text{veh}}}), \tag{18}$$

consists of the position x_i, y_i, longitudinal speed v_i, and heading ψ_i, of each vehicle, where index 0 refers to the ego vehicle. The agent that controls the ego vehicle can observe other vehicles within the sensor range $x_{\text{sensor}} = 200$ m, unless they are occluded.

2) Action space, $\mathcal{A}$: At every time step, the agent can choose between three high-level actions: *'stop'*, *'cruise'*, and *'go'*, which are translated into accelerations through the IDM. The action *'go'* makes the IDM control the speed towards v_{set} by treating the situation as if there are no preceding vehicles, whereas *'cruise'* simply keeps the current speed. The action *'stop'* places an imaginary target vehicle just before the intersection, which causes the IDM to slow down and stop at the stop line. If the ego vehicle has already passed the stop line, *'stop'* is interpreted as maximum braking. Finally, the output of the IDM is limited to $[a_{\min}, a_{\max}] = [-3, 1]$ m/s^2. Note that the agent takes a new decision at every time step Δt and can therefore switch between, e.g., *'stop'* and *'go'* multiple times during an episode.

3) Reward model, R: The objective of the agent is to drive through the intersection in a time efficient way, without colliding with other vehicles. A simple reward model is used to achieve this objective. The agent receives a positive reward $r_{\text{goal}} = 10$ when the ego vehicle manages to cross the intersection and a negative reward $r_{\text{col}} = -10$ if a collision occurs.

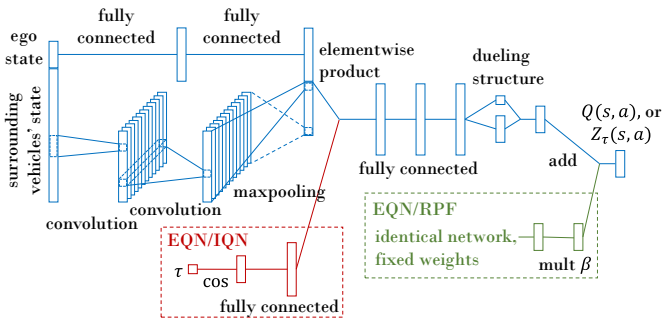


Fig. 2. The neural network architecture that is used for the different agents. The red part is included for the EQN and IQN agents, whereas the green part is included for the EQN and RPF agents.

If the ego vehicle gets closer to another vehicle than 2.5 m longitudinally or 1 m laterally, a negative reward $r_{\text{near}} = -10$ is given, but the episode is not terminated. At all other time steps, the agent receives 0 as reward.

4) Transition model, T: The state transition probabilities are not known by the agent, and they are implicitly defined by the simulation model, described in Sect. III-A.

C. Backup policy

A simple backup policy $\pi_{\text{backup}}(s)$ is used together with the uncertainty criteria. This policy selects the action *'stop'* if the vehicle is able to stop before the intersection, considering the braking limit $a_{\min}$. Otherwise, the backup policy selects the action that is recommended by the agent. If the backup policy would always consist of *'stop'*, the ego vehicle could end up standing still in the intersection and thereby cause more collisions. Naturally, more advanced backup policies would be considered in a real-world implementation, for example based on optimal control [39], but such a policy would not significantly change the results of this study.

D. Neural network architecture

In previous work, we introduced a one-dimensional convolutional neural network architecture, which improves both the training speed and final performance, compared to a standard fully connected architecture [7]. By applying convolutional layers and a maxpooling layer to the input that describes the state of the surrounding vehicles, the output becomes both invariant to the ordering of the surrounding vehicles in the input vector and independent of the number of surrounding vehicles. A more detailed description of this architecture is provided in the previous work [7].

Fig. 2 shows the neural network architecture that is used in this study. The size and stride of the first convolutional layers are set to four, which is equal to the number of states that describe each surrounding vehicle, whereas the second convolutional layer has a size and stride of one. Both convolutional layers have 256 filters each, and all fully connected layers have 256 units. Finally, a dueling structure [40], which separates the estimation of the value of a state and the advantage of an action, outputs $Z_\tau(s,a)$ or $Q(s,a)$, depending on which algorithm that is used. All layers use rectified linear units (ReLUs) as activation functions, except for the dueling layer, which has a linear activation function. Before the state s is fed to the network, each entry is normalized to the range $[-1,1]$ by considering the possible minimum and maximum values.

The network architecture for the IQN agent has an additional input for the sample quantile τ, shown in Fig. 2. As proposed by Dabney et al. [27], an embedding from τ is created by setting $\phi(\tau) = (\phi_1(\tau),\dots,\phi_{64}(\tau))$, where $\phi_j(\tau) = \cos(\pi j \tau)$, and then passing $\phi(\tau)$ through a fully connected layer with 512 units. The output of the embedding is then merged with the output of the concatenating layer as the element-wise (Hadamard) product.

TABLE I
HYPERPARAMETERS OF THE DIFFERENT ALGORITHMS

IQN, EQN	Number of quantile samples, N, N', K_τ	32
	CVaR parameter, α	1
RPF, EQN	Number of ensemble members, K	10
	Prior scale factor, β	300
	Experience adding probability, p_{add}	0.5
DQN, IQN, RPF, EQN	Discount factor, γ	0.95
	Learning start iteration, N_{start}	50,000
	Replay memory size, N_{replay}	500,000
	Learning rate, η	0.0005
	Mini-batch size, $\lvert M\rvert$	32
	Target network update frequency, N_{update}	20,000
	Huber loss threshold, κ	10
DQN, IQN	Initial exploration parameter, ϵ_0	1
	Final exploration parameter, ϵ_1	0.05
	Final exploration iteration, N_ϵ	500,000

E. Training process

Algorithm 1, 2, and 3 were used to train the IQN, RPF, and EQN agents, respectively. Additionally, the Double DQN trick was used to reduce overestimation of the Q-values [41], which subtly changes the maximization part in Eq. 3 and 12. During the training of the DQN and IQN agents, an ϵ-greedy exploration policy was followed, where ϵ was linearly decreased from ϵ_0 to ϵ_1 over N_ϵ training steps. Huber loss was applied to the TD-error of all the algorithms, in order to improve the robustness of the training process, and the neural network weights were updated by the Adam optimizer [42]. The training process was parallelized for the ensemble-based versions, in order to reduce the training time. Table I displays the hyperparameters that were used for the different algorithms. Due to the computational complexity, a systematic grid search was not performed. Instead, the hyperparameter values were selected from an informal search, based upon the values given by Mnih et al. [30], Dabney et al. [27], and Osband et al. [20]. Additional results are also presented for a set of different values of α, β, and K, in order to demonstrate how the choice of these parameters influence the behavior of the agent.

As mentioned in Sect. III-A, an episode is terminated due to a timeout after maximally $N_{\max}$ steps, since otherwise the current policy could make the ego vehicle stop at the intersection indefinitely. However, since the time is not part of the state space, a timeout terminating state is not described by the MDP. Therefore, in order to make the agents act as if the

episodes have no time limit, the last experience of a timeout episode is not added to the experience replay buffer.

All the agents are trained for 3,000,000 training steps, at which point the agents' policies have converged, and then the trained agents are tested on 1,000 randomly initialized test episodes. The test episodes are generated in the same way as the training episodes, described in Sect. III-A, but they are not present during the training phase. Furthermore, the set of test episodes is identical for all the trained agents, in order to provide an appropriate comparison. Each agent is trained with five random seeds and the mean results are presented, together with the corresponding standard deviation.

IV. Results

The results show that the IQN method can be used to estimate the aleatoric uncertainty in a traffic situation and the uncertainty criterion can be used to identify situations with high uncertainty, in order to prevent collisions. The results also illustrate that the ensemble RPF method can provide an estimate of the epistemic uncertainty and use the uncertainty criterion to classify situations as within or outside the training distribution. Furthermore, the results of the EQN method demonstrate that this approach provides a complete estimate of both types of uncertainty. This section presents the results in detail, together with an analysis of the characteristics of the results, whereas a broader discussion on the properties of the algorithms follows in Sect. V. Animations of the presented scenarios are available on GitHub [28].

A. Aleatoric uncertainty estimation

To illustrate the behavior of the trained IQN agent and provide intuition on how risk-sensitive training affects the obtained policy, results for the sparse traffic scenario are first displayed. Table II shows nearly identical quantitative results for a trained risk-neutral IQN agent and a DQN agent. Both agents find a policy that drives through the intersection at the maximum speed if no crossing vehicles are observed, see Fig. 3. Since crossing traffic is sparse, this policy maximizes the expected return, but causes collisions in around one out of ten test episodes. An IQN agent that is trained in a risk-sensitive way, by setting the CVaR parameter $\alpha = 0.5$, instead slows down and passes the occluded area with a low speed, which allows the ego vehicle to stop before the intersection if a crossing vehicle appears. Such a policy solves almost all test episodes without collisions, but increases the mean duration of an episode, hereafter referred to as crossing time, with around 50%. An example of a situation that causes a collision with risk-neutral training but is collision-free with risk-sensitive training, is shown in Fig. 4a. In this situation, the ego vehicle is driving at 15 m/s, while an occluded vehicle is approaching from the west. Fig. 4b and 4c display the estimated quantile function of the return distribution $Z_\tau(s, a)$, which reveal that both agents are aware of the risk of a collision, indicated by the small probability ($\tau < 0.1$) of a negative return. However, different policies are obtained, due to the difference in risk-sensitivity. The reason for the high aleatoric uncertainty in actions *'go'* and *'cruise'* of the risk-averse agent (Fig. 4c) is that the agent will not be able to later decide to stop before the intersection, due to the limited braking capacity.

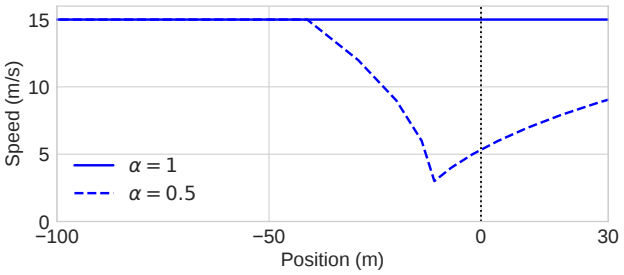
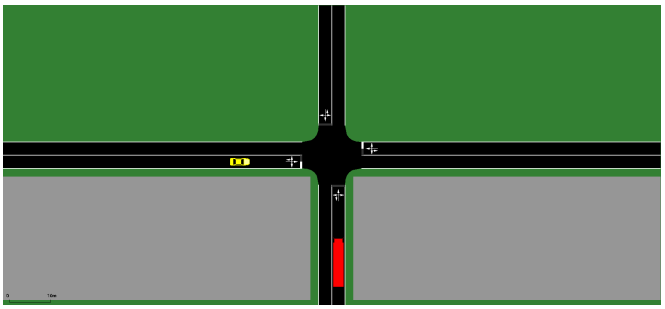


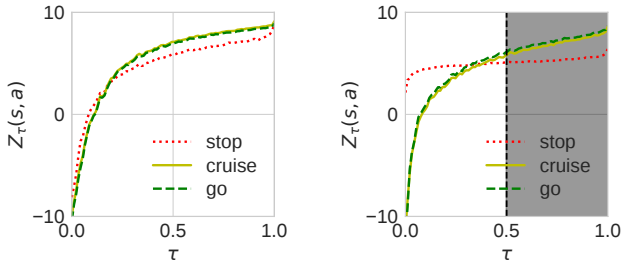
Fig. 3. Speed of the ego vehicle as a function of distance to the occluded intersection, positioned at the dotted vertical line, for a sparse traffic scenario. In this episode, no crossing vehicles are observed.

TABLE II
Sparse traffic scenario

		collisions (%)	crossing time (s)
IQN	$\alpha = 1$	10.8 ± 0.2	15.8 ± 0.1
IQN	$\alpha = 0.5$	0.1 ± 0.1	24.0 ± 0.4
DQN		10.7 ± 0.2	15.9 ± 0.1



(a) The ego vehicle is shown in red, the occluded vehicle in yellow, and the areas that cause occlusions are displayed in gray.



(b) $\alpha = 1$ (c) $\alpha = 0.5$

Fig. 4. Example of a situation that results in a collision for an agent with risk-neutral training ($\alpha = 1$) but is solved without collisions with risk-sensitive training ($\alpha = 0.5$). The estimated quantile function of the random variable $Z_\tau(s, a)$ is shown for both cases and indicates that both agents are aware of the aleatoric uncertainty in the situation.

Table III shows how the trained IQN agent performs in the second test scenario, in which traffic is dense and the occluding objects are placed further from the intersection. The results illustrate the natural trade-off between time efficiency and safety. With a more risk-averse training (lower value of the CVaR parameter α), the number of collisions is reduced, but

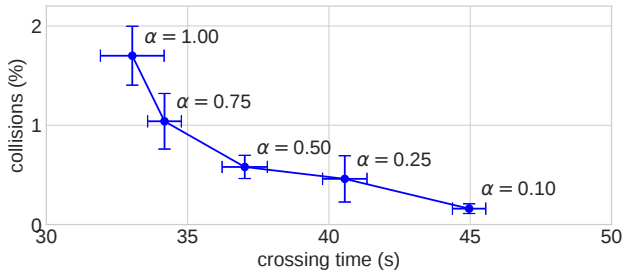


Fig. 5. Number of collisions and crossing time for different levels of risk-sensitive training, which is achieved by varying the CVaR parameter α.

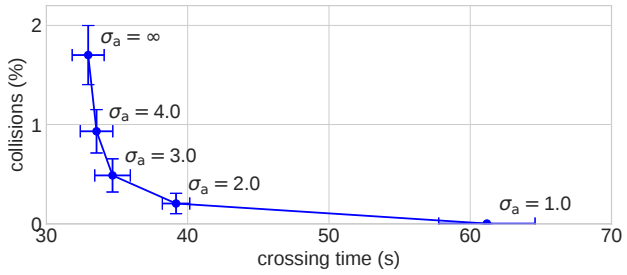


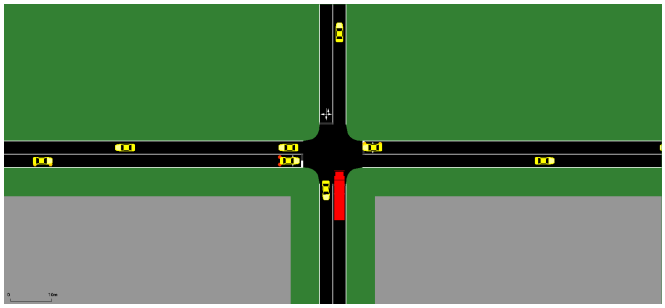
Fig. 6. Number of collisions and crossing time for the IQN algorithm for different levels of allowed aleatoric uncertainty, which is achieved by varying the parameter σ_a.

the time it takes to cross the intersection increases, see Fig. 5.

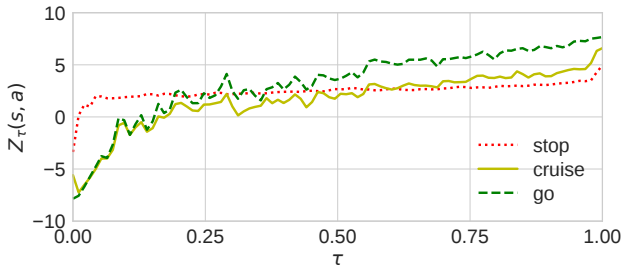
Furthermore, the results in Table III and Fig. 6 demonstrate that the IQN method, combined with the aleatoric uncertainty criterion, can be used to detect situations with high aleatoric uncertainty. When the maximum allowed uncertainty is reduced (lower values of σ_a), the number of collisions is reduced. However, similarly to training in a risk-averse manner, a more conservative policy increases the time it takes to cross the intersection. An example situation with high aleatoric uncertainty, due to uncertainty in the intention of another vehicle, is shown in Fig. 7a. The car that is approaching the intersection from the west has here slowed down due to a preceding car, which turned to the south. Due to the low speed, the IQN agent expects that the approaching car will also turn to the south. Therefore, the agent estimates that in most cases it would be best to choose the action *'go'*, to immediately cross the intersection, see Fig. 7b. However, the agent also estimates that with a low probability, this action can cause a collision, which is indicated by the negative values of the estimated return distribution Z_τ. Since the sample variance is high in this situation, $\mathrm{Var}_{\tau_\sigma}[Z_\tau(s, a_\mathrm{go})] = 12.0$, an uncertainty criterion with $\sigma_\mathrm{a}^2 < 12.0$ prevents a collision by choosing the backup policy, i.e., stopping at the intersection.

B. Epistemic uncertainty estimation

The trained RPF agent performs similarly as the risk-neutral IQN agent in the dense traffic scenario, see Table III. The parameter β, which scales the importance of the random prior network and thereby influences the exploration strategy, has a relatively small effect on the performance within the training distribution. Even setting $\beta = 0$, which completely removes



(a) The ego vehicle is shown in red and has a speed of 1 m/s.



(b) Estimated quantile function of the random variable $Z_\tau(s, a)$.

Fig. 7. Example of a situation with high aleatoric uncertainty for the actions *'go'* and *'cruise'*, due to uncertainty in the intention of the vehicle that is about to enter the intersection from the west.

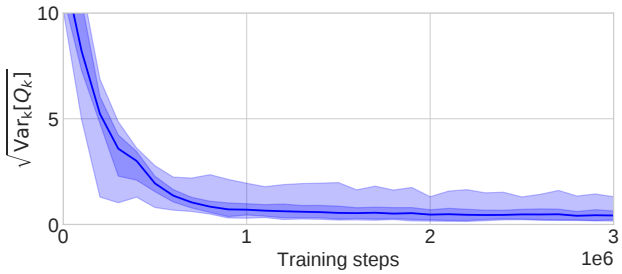


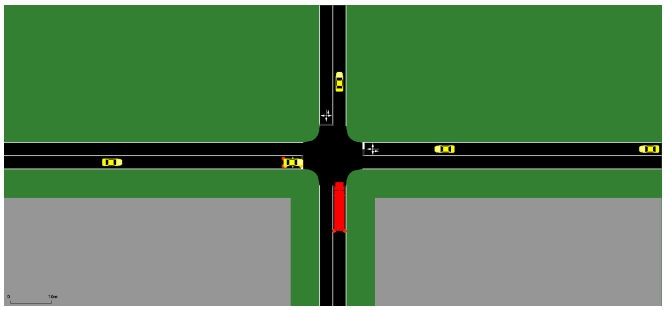
Fig. 8. Epistemic uncertainty of the chosen actions for the ensemble RPF agent, with parameters $\beta = 300$ and $K = 10$, during testing episodes within the training distribution. The solid line shows the mean, while the shaded regions indicate percentile 10 to 90 and 1 to 99.

the effect of the random prior network and only relies on statistical bootstrapping for exploration, gives reasonable results. Similarly, the number of networks K have a low effect on the performance. Fig. 8 shows how the epistemic uncertainty of the chosen actions during the test episodes is reduced during the training process.

To illustrate that the RPF agent can estimate the epistemic uncertainty and detect situations that are outside of the training distribution, the trained agent is exposed to crossing traffic with a higher speed than during the training episodes, see Sect. III-A. Fig. 10 shows that if no epistemic uncertainty threshold is used, i.e., setting $\sigma_\mathrm{e} = \infty$, the number of collisions increases significantly when the speed of the crossing vehicles increases. If the threshold σ_e is reduced, the number of collisions is reduced to almost zero, whereas the number of timeouts increases. An example of a situation with high epistemic uncertainty, in which a collision is avoided by limiting the allowed uncertainty, is shown in Fig. 9.

TABLE III
DENSE TRAFFIC SCENARIO, TESTED WITHIN THE TRAINING DISTRIBUTION

algorithm, fixed param.	variable parameter	collisions (%)	crossing time (s)
DQN	-	4.0 ± 0.5	31.7 ± 1.1
IQN, $\sigma_\mathrm{a} = \infty$	$\alpha = 1.0$	1.7 ± 0.3	33.0 ± 1.1
	$\alpha = 0.75$	1.0 ± 0.3	34.2 ± 0.6
	$\alpha = 0.5$	0.6 ± 0.1	37.0 ± 0.8
	$\alpha = 0.25$	0.4 ± 0.2	40.6 ± 0.8
	$\alpha = 0.1$	0.2 ± 0.1	45.0 ± 0.6
IQN, $\alpha = 1$	$\sigma_\mathrm{a} = \infty$	1.7 ± 0.3	33.0 ± 1.1
	$\sigma_\mathrm{a} = 4.0$	0.9 ± 0.2	33.5 ± 1.2
	$\sigma_\mathrm{a} = 3.0$	0.5 ± 0.2	34.7 ± 1.3
	$\sigma_\mathrm{a} = 2.0$	0.2 ± 0.1	39.2 ± 1.0
	$\sigma_\mathrm{a} = 1.0$	0.0 ± 0.0	61.2 ± 3.4
RPF, $K = 10$	$\beta = 0$	3.0 ± 0.2	29.4 ± 0.3
	$\beta = 100$	2.8 ± 0.4	32.1 ± 0.5
	$\beta = 300$	1.5 ± 0.3	38.0 ± 1.8
	$\beta = 1000$	1.8 ± 0.4	44.6 ± 1.0
RPF, $\beta = 300$	$K = 3$	3.0 ± 1.0	34.8 ± 1.6
	$K = 10$	1.5 ± 0.3	38.0 ± 1.8
	$K = 30$	1.9 ± 0.4	34.6 ± 1.4
EQN, $\alpha = 1.0$, $K = 10$, $\beta = 300$	$\sigma_\mathrm{a} = \infty$	0.9 ± 0.1	32.0 ± 0.2
	$\sigma_\mathrm{a} = 3.0$	0.6 ± 0.2	33.8 ± 0.3
	$\sigma_\mathrm{a} = 2.0$	0.5 ± 0.1	38.4 ± 0.5
	$\sigma_\mathrm{a} = 1.5$	0.3 ± 0.1	47.2 ± 1.2
	$\sigma_\mathrm{a} = 1.0$	0.0 ± 0.0	71.1 ± 1.9
	$\sigma_\mathrm{a} = 1.5$, $\sigma_e = 1.0$	0.0 ± 0.0	48.9 ± 1.6

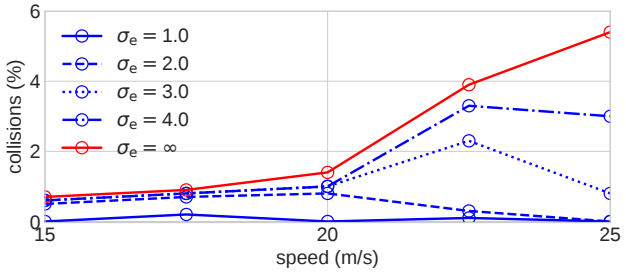


(a) $t = 0$, $\mathrm{Var}_k[Q_k(s, a_\mathrm{go})] = 57.8$

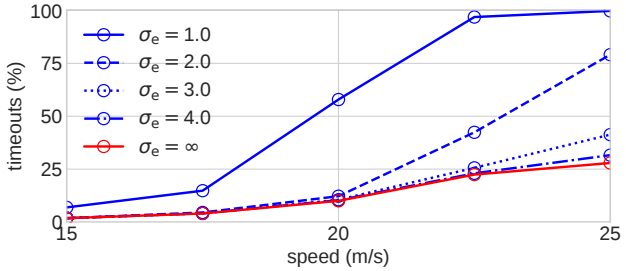
(b) $t = 3$ s, $\sigma_\mathrm{e} = \infty$ (c) $t = 3$ s, $\sigma_\mathrm{e} = 4$

Fig. 9. Example of a situation with high epistemic uncertainty (a), in which the eastmost vehicle approaches the intersection with a speed of 23 m/s. Without the epistemic uncertainty threshold, the RPF agent chooses to cross the intersection, which causes a collision (b), whereas the collision is avoided when the uncertainty criterion is applied (c).

The result at 15 m/s, which is included in the training distribution, shows that the number of collisions is also somewhat reduced within the training distribution. We hypothesize that the reason for this effect is that some situations that cause



(a) Collisions



(b) Timeouts

Fig. 10. Number of collisions and timeouts for the RPF agent ($\beta = 300$, $K = 10$), in situations outside the training distribution. The maximum speed of the crossing vehicles is 15 m/s during the training process, and then the speed is gradually increased in the testing episodes.

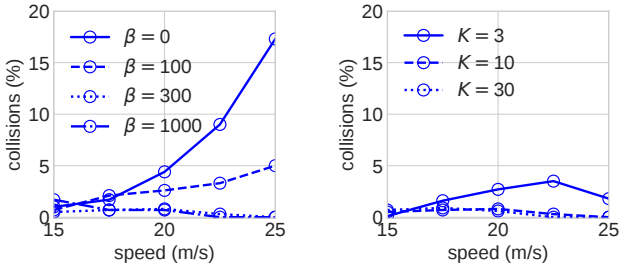
collisions are seldom seen during the training process, and therefore the epistemic uncertainty in those situations remains high.

As previously mentioned, the scaling factor β and the number of networks K do not substantially influence the performance of the RPF agent inside the training distribution. However, these parameters determine how well the agent can estimate the epistemic uncertainty and detect dangerous situations, which is illustrated in Fig. 11. In short, β and K need to be sufficiently large to give a reasonable uncertainty estimate. How to set these parameter values are further discussed in Sect. V.

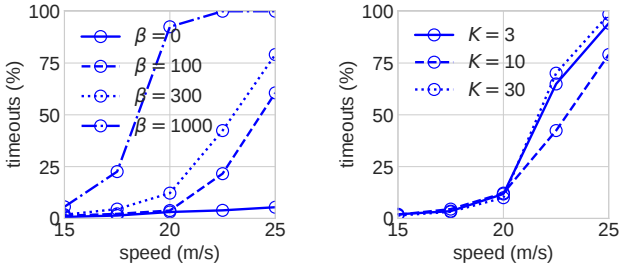
C. Aleatoric and epistemic uncertainty estimation

The EQN agent performs better than the RPF agent and similar to the IQN agent within the training distribution, see Table III. Importantly, the EQN agent combines the advantages of the other two agents and can estimate both the aleatoric and epistemic uncertainty of a decision. When the aleatoric uncertainty criterion is applied, the number of situations that are classified as uncertain depends on the parameter σ_a, see Fig. 12. Thereby, the trade-off between risk and time efficiency, here illustrated by number of collisions and crossing time, can be controlled by tuning the value of σ_a.

The performance of the epistemic uncertainty estimation of the EQN agent is illustrated in Fig. 13, where the speed of the surrounding vehicles is increased. Similarly as for the RPF agent, a sufficiently strict epistemic uncertainty criterion, i.e., sufficiently low value of the parameter σ_e, prevents the number of collisions to increase when the speed of the surrounding vehicles increases. The result at 15 m/s also indicates that the number of collisions within the training distribution is



(a) Collisions, fixed $K = 10$. (b) Collisions, fixed $\beta = 300$.



(c) Timeouts, fixed $K = 10$. (d) Timeouts, fixed $\beta = 300$.

Fig. 11. Number of collisions and timeouts for the RPF agent, with uncertainty threshold $\sigma_e = 2$, varying values of the prior scaling factor β and the number of ensemble members K, in situations outside of the training distribution.

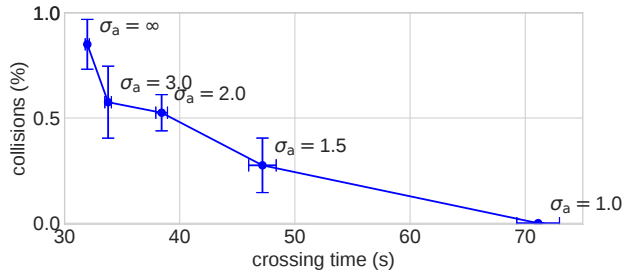
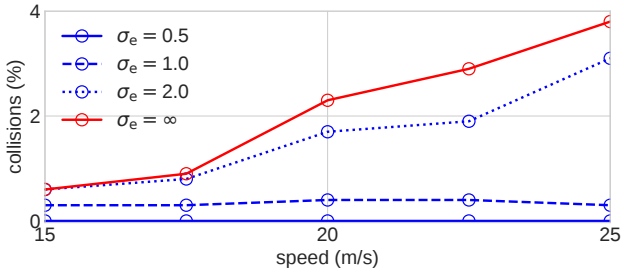


Fig. 12. Number of collisions and crossing time for the EQN algorithm for different levels of allowed aleatoric uncertainty, which is achieved by varying the parameter σ_a.

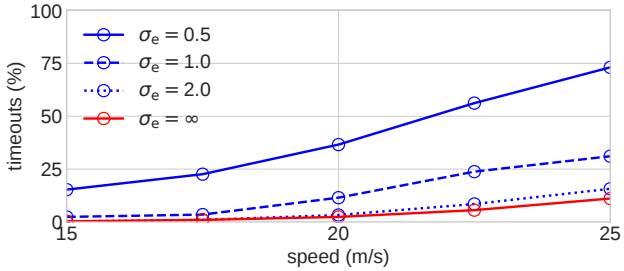
somewhat reduced when the epistemic uncertainty condition is applied. Interestingly, when combining moderate aleatoric and epistemic uncertainty criteria, by setting $\sigma_a = 1.5$ and $\sigma_e = 1.0$, all the collisions within the training distribution are removed, see Table III. These results show that it is useful to consider the epistemic uncertainty even within the training distribution, where the detection of uncertain situations can prevent collisions in rare edge cases.

V. Discussion

The results show that the IQN and RPF agents can provide estimates of the aleatoric and epistemic uncertainties, respectively. When combined with the uncertainty criteria, situations with high uncertainty are identified, which can be used to make safer decisions. Further use and characteristics of the uncertainty information are discussed below. The results also demonstrate that the EQN agent combines the advantages of



(a) Collisions



(b) Timeouts

Fig. 13. Number of collisions and timeouts for the EQN agent in situations outside the training distribution. The maximum speed of the crossing vehicles is 15 m/s during the training process, and then the speed is gradually increased in the testing episodes.

the individual components and provides a full uncertainty estimate, including both the aleatoric and epistemic dimensions.

The aleatoric uncertainty estimate, given by the IQN or EQN algorithms, can be used to balance risk and time efficiency, either by training in a risk-sensitive way (varying the CVaR parameter α, see Fig. 5) or applying the aleatoric uncertainty criterion (varying the allowed variance σ_a^2, see Fig. 6 and 12). An important advantage of the uncertainty criterion approach is that its parameter σ_a can be tuned after the training process has been completed, whereas the agent needs to be retrained for each CVaR parameter α. However, the uncertainty criterion only works in practical applications, such as autonomous driving, where a backup policy can be defined. The CVaR approach does not require a backup policy and is therefore suitable for environments where such a policy is hard to define, e.g., the Atari-57 benchmark [27]. Bernhard et al. first trained a risk-neutral IQN agent, then lowered the CVaR threshold after the training process had been completed, and showed a reduction of collisions in an intersection driving scenario [15]. However, such a procedure does not provide the correct estimate of the return distribution $Z_\tau(s, a)$ for a risk-averse setting ($\alpha < 1$) and could lead to arbitrary decisions. The problem with this approach is easily seen for the simple MDP shown in Fig. 14. The risk-neutral policy is $\pi_{rn}(s_1) = a_1$ and $\pi_{rn}(s_2) = a_1$, whereas the risk-averse policy is $\pi_{ra}(s_1) = a_1$ and $\pi_{ra}(s_2) = a_2$, for CVaR parameter $\alpha < 0.75$. For this risk-averse policy, the return of the initial state is $Z^{\pi_{ra}}(s_1, \pi_{ra}(s_1)) = 1$, with probability 1. However, if $Z_\tau(s, a)$ is first estimated for the risk-neutral policy, and then the action that maximizes the CVaR for $\alpha < 0.6$ is chosen, this risk-averse policy gives $\tilde{\pi}_{ra}(s_1) = a_2$, and the return of

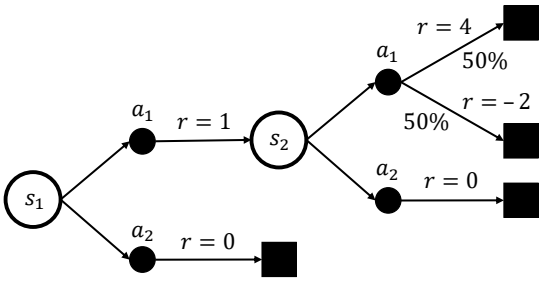


Fig. 14. A simple MDP, which illustrates that a risk-sensitive IQN policy needs to be retrained for each value of the CVaR parameter α.

the initial state is $Z^{\tilde{\pi}_{\mathrm{ra}}}(s_1, \tilde{\pi}_{\mathrm{ra}}(s_1)) = 0$, with probability 1. In short, the policy $\tilde{\pi}_{\mathrm{ra}}$ becomes sub-optimal, since it considers the aleatoric uncertainty of the risk-neutral policy, which is irrelevant in this case.

An alternative to estimating the distribution over returns and still consider aleatoric risk in the decision-making is to adapt the reward function. Risk-sensitivity could be achieved by, for example, increasing the size of the negative reward for collisions. However, rewards with different orders of magnitude create numerical problems, which can disrupt the training process [30]. Furthermore, for a complex reward function, it would be non-trivial to balance the different components to achieve the desired result.

The epistemic uncertainty information provides insight into how far a situation is from the training distribution. In this study, the usefulness of an epistemic uncertainty estimate is demonstrated by increasing the safety, through classifying the agent's decisions in situations far from the training distribution as unsafe and then instead applying a backup policy. If it is possible to formally guarantee safety with a learning-based method is an open question, and likely an underlying safety layer is required in a real-world application [43]. The RPF and EQN agents can reduce the activation frequency of such a safety layer, but possibly even more importantly, the epistemic uncertainty information could be used to guide the training process to regions of the state space in which the current agent requires more training. Furthermore, if an agent is trained in a simulated world and then deployed in the real world, the epistemic uncertainty information can identify situations with high uncertainty, which should be added to the simulated world.

The algorithms that were introduced in this paper include a few hyperparameters, whose values need to be set appropriately. The aleatoric and epistemic uncertainty criteria parameters, σ_{a} and σ_{e}, can both be tuned after the training is completed and allow a trade-off between risk and time efficiency, see Fig. 6, 10, 12, and 13. Note that both these parameters determine the allowed spread in returns, between quantiles or ensemble members, which means that the size of these parameters are closely connected to the magnitude of the reward function. In order to detect situations with high epistemic uncertainty, a sufficiently large spread between the ensemble members is required, which is controlled by the scaling factor β and the number of ensemble members K. The choice of β scales with the magnitude of the reward function. A too small parameter value creates a small spread, which makes it difficult to classify situations outside of the training distribution as uncertain, see Fig. 11a and 11c. On the other hand, a too large value of β makes it difficult for the trainable network to adapt to the fixed prior network. Furthermore, an increased number of ensemble members K naturally increases the accuracy of the epistemic uncertainty estimate, see Fig. 11b and 11d, but induces a higher computational cost.

All the tested methods have a similar sample complexity, but the uncertainty-aware approaches require more computational resources than the baseline DQN method. The IQN method uses N quantile samples in the loss function, the RPF method trains an ensemble of K neural network, and the EQN method combines these two features. However, the design of the algorithms allows a parallel implementation, which in practice reduces the difference in training time. All agents were trained on a standard desktop computer, where the DQN agent required 12 hours, the IQN agent 24 hours, the RPF agent 72 hours, and the EQN agent 96 hours. However, since the focus of this study is not to optimize the implementation, the time efficiency can be significantly improved.

VI. Conclusion

The results show that the proposed EQN algorithm combines the advantages of the IQN and RPF methods, and can thereby provide a complete uncertainty estimate of its decisions, including both the aleatoric and the epistemic uncertainty. The aleatoric uncertainty criterion allows an agent to balance risk and time efficiency after the training is completed and achieves similar results as an agent that is trained in a risk-sensitive way, with the benefit that the agent does not need to be retrained for each uncertainty threshold. Furthermore, the results show that the epistemic uncertainty criterion can be used to identify situations that are far from the training distribution, in which the agent could make dangerous decisions. The awareness of such situations can be used to enhance the safety of the trained agent and to improve the training process.

The EQN algorithm provides a general approach to create an uncertainty-aware decision-making agent for autonomous driving. However, in order to apply the method to other driving scenarios than the intersections that were considered in this study, the MDP formulation needs to be adapted to the new scenarios, or an MDP that covers multiple scenarios needs to be constructed. While the DQN-family of methods have proved to work well for different types of driving scenarios [14], [44], future work involves to test the EQN method in more scenarios and other simulation environments, before performing tests in the real world. It would also be interesting to investigate how the algorithm would handle different aspects of noise in the sensor signals. Another topic of future work is to investigate how the epistemic uncertainty estimation can be used to bridge the gap between simulators and reality. An RPF or EQN agent that has been trained in a simulated world could potentially detect traffic situations in the real world where the epistemic uncertainty is high and then automatically add these situations to the simulated environment.

References

[1] A. D. Kiureghian and O. Ditlevsen, "Aleatory or epistemic? Does it matter?," *Struct. Saf.*, vol. 31, no. 2, pp. 105–112, 2009.

[2] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," *Mach. Learn.*, vol. 110, no. 3, pp. 457–506, 2021.

[3] R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning," in *Proc. 26th Int. Joint Conf. on Artif. Intell.*, pp. 4745–4753, 2017.

[4] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Adv. in Neural Inf. Process. Syst. 29*, pp. 4026–4034, 2016.

[5] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep Q-learning," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, pp. 3169–3174, 2018.

[6] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *IEEE Int. Conf. on Robot. and Automat. (ICRA)*, pp. 2034–2039, 2018.

[7] C. J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, pp. 2148–2155, 2018.

[8] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving." arXiv:1610.03295, 2016.

[9] C. J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," *IEEE Trans. on Intell. Veh.*, vol. 5, no. 2, pp. 294–305, 2020.

[10] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017.

[11] M. Bansal, A. Krizhevsky, and A. S. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," *Robot: Sci. & Syst. (RSS)*, 2019.

[12] A. Kendall *et al.*, "Learning to drive in a day," in *IEEE Int. Conf. on Robot. and Automat. (ICRA)*, pp. 8248–8254, 2019.

[13] B. R. Kiran *et al.*, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. on Intell. Transp. Syst.*, 2021.

[14] Z. Zhu and H. Zhao, "A survey of deep RL and IL for autonomous driving policy learning." arXiv:2101.01993, 2021.

[15] J. Bernhard, S. Pollok, and A. Knoll, "Addressing inherent uncertainty: Risk-sensitive behavior generation for automated driving using distributional reinforcement learning," in *IEEE Intell. Veh. Symp. (IV)*, pp. 2148–2155, 2019.

[16] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[17] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," in *Proc. of the Brit. Mach. Vision Conf. (BMVC)*, pp. 57.1–57.12, 2017.

[18] R. Michelmore, M. Kwiatkowska, and Y. Gal, "Evaluating uncertainty quantification in end-to-end autonomous driving control." arXiv:1811.06817, 2018.

[19] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," in *Proc. 15th AAAI Conf. on Artif. Intell.*, p. 761–768, 1998.

[20] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," in *Adv. in Neural Inf. Process. Syst. 31*, pp. 8617–8629, 2018.

[21] M. J. Sobel, "The variance of discounted markov decision processes," *J. of Appl. Prob.*, vol. 19, no. 4, pp. 794–802, 1982.

[22] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Nonparametric return distribution approximation for reinforcement learning," in *Proc. 27th Int. Conf. on Mach. Learn. (ICML)*, p. 799–806, 2010.

[23] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. 34th Int. Conf. on Mach. Learn. (ICML)*, pp. 449–458, 2017.

[24] G. Barth-Maron *et al.*, "Distributed distributional deterministic policy gradients," in *Int. Conf. on Learn. Repr. (ICLR)*, 2018.

[25] C. J. Hoel, K. Wolff, and L. Laine, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," in *IEEE Intell. Veh. Symp. (IV)*, pp. 1563–1569, 2020.

[26] C. J. Hoel, T. Tram, and J. Sjöberg, "Reinforcement learning with uncertainty estimation for tactical decision-making in intersections," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, pp. 318–324, 2020.

[27] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *Proc. 35th Int. Conf. on Mach. Learn. (ICML)*, pp. 1096–1105, 2018.

[28] C. J. Hoel, "Source code for 'Ensemble quantile networks: Uncertainty-aware reinforcement learning with applications in autonomous driving'." https://github.com/carljohanhoel/EnsembleQuantileNetworks, 2021.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[30] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[31] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. 32nd AAAI Conf. on Artif. Intell.*, pp. 2892–2901, 2018.

[32] P. J. Huber, "Robust estimation of a location parameter," *The Ann. of Math. Statist.*, vol. 35, no. 1, pp. 73–101, 1964.

[33] R. Rockafellar and S. Uryasev, "Conditional value-at-risk for general loss distributions," *J. of Banking & Finance*, vol. 26, no. 7, pp. 1443–1471, 2002.

[34] Y. Chow and M. Ghavamzadeh, "Algorithms for CVaR optimization in MDPs," in *Adv. in Neural Inf. Process. Syst. 27*, pp. 3509–3517, 2014.

[35] A. Majumdar and M. Pavone, "How should a robot assess risk? Towards an axiomatic theory of risk in robotics." arXiv:1710.11040, 2017.

[36] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*. Soc. for Ind. and Appl. Math., 1982.

[37] P. A. Lopez *et al.*, "Microscopic traffic simulation using SUMO," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, pp. 2575–2582, 2018.

[38] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, 2000.

[39] L. Svensson *et al.*, "Safe stop trajectory planning for highly automated vehicles: An optimal control problem formulation," in *IEEE Intell. Veh. Symp. (IV)*, pp. 517–522, 2018.

[40] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. on Mach. Learn. (ICML)*, pp. 1995–2003, 2016.

[41] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 39th AAAI Conf. on Artif. Intell.*, pp. 2094–2100, 2016.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Int. Conf. on Learn. Repr.*, 12 2014.

[43] S. Underwood, D. Bartz, A. Kade, and M. Crawford, "Truck automation: Testing and trusting the virtual driver," in *Road Veh. Automat. 3* (G. Meyer and S. Beiker, eds.), pp. 91–109, Springer, 2016.

[44] F. Ye, S. Zhang, P. Wang, and C.-Y. Chan, "A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles." arXiv:2105.14218, 2021.

Carl-Johan Hoel received the B.S. and M.S. degrees in physics from Chalmers University of Technology, Gothenburg, Sweden. He is currently working towards the Ph.D. degree at Chalmers University of Technology, together with the Volvo Group, Gothenburg, Sweden. His research focuses on robust reinforcement learning methods for creating general decision-making agents, applied to autonomous driving.

Krister Wolff received the M.S. degree in physics from Gothenburg University, Gothenburg, Sweden and the Ph.D. degree from Chalmers University of Technology, Gothenburg, Sweden. He is currently an Associate Professor of adaptive systems, and he is also the Vice head of Department at Mechanics and maritime sciences, Chalmers. His research is within the application of AI in different domains, such as autonomous robots and self-driving vehicles, using machine learning and bio-inspired computational methods as the main approaches.

Leo Laine received the Ph.D. degree from Chalmers University of Technology, Gothenburg, Sweden, within Vehicle Motion management. Since 2007, he has been with the Volvo Group Trucks Technology (VGTT) in the Vehicle Automation department. Since 2013, he has also been an Adjunct Professor in vehicle dynamics with Chalmers Vehicle Engineering and Autonomous Systems. Since 2013, he is a specialist within complete vehicle control. Since 2019, he is a technical advisor within Vehicle Motion and Energy Management within VGTT.