High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning

Branka Mirchevska[1], Christian Pek[1], Moritz Werling[1], Matthias Althoff[2], and Joschka Boedecker[3]

Abstract— **Machine learning techniques have been shown to outperform many rule-based systems for the decision-making of autonomous vehicles. However, applying machine learning is challenging due to the possibility of executing unsafe actions and slow learning rates. We address these issues by presenting a reinforcement learning-based approach, which is combined with formal safety verification to ensure that only safe actions are chosen at any time. We let a deep reinforcement learning (RL) agent learn to drive as close as possible to a desired velocity by executing reasonable lane changes on simulated highways with an arbitrary number of lanes. By making use of a minimal state representation, consisting of only 13 continuous features, and a Deep Q-Network (DQN), we are able to achieve fast learning rates. Our RL agent is able to learn the desired task without causing collisions and outperforms a complex, rule-based agent that we use for benchmarking.**

I. INTRODUCTION

A. Motivation

Motion planning for autonomous vehicles is challenging due to the influence of surrounding traffic participants, traffic rules, and conflicting optimization criteria. One possibility to handle this complexity is by separating the planning task into high-level decision-making and maneuver execution. The decision-making layer plans high-level maneuvers, such as changing lanes or keeping the lane, while in the maneuver execution layer, the detailed motion is planned and executed.

High-level decision making involves building a system of rules and deducing the optimal maneuver online. The disadvantages of these systems become obvious when considering traffic scenarios that have not been considered during the construction of the rule-based system. In this case, new rules need to be added and integrated in the existing rules-based system. Usually, this vastly increases the complexity of the decision-making process while the comprehensibility is reduced during the development process.

In recent years, the evolution of computing power, as well as the increased ubiquity of data, has paved the way for applying machine learning techniques to decision-making. Machine learning provides the possibility for vehicles to learn from data and improve the learned strategy in the future based on gathered experience. In particular, reinforcement learning (RL) allows the vehicle to learn certain tasks based on interaction with its environment. RL can deal with large-scale systems with possibly infinite state and action spaces in a model-free way.

Coming up with a minimal, yet sufficient representation of the environment can drastically accelerate the learning process. Additionally, guaranteeing that the vehicle only performs safe maneuvers, i. e., does not cause a collision, increases the complexity or the design even more. Safety at any time is especially important when performing learning in real traffic with imminent danger for other traffic participants.

B. Related Work

One of the first machine learning approaches for autonomous vehicles was proposed in 1989 [1]. They developed a neural network-based autonomous land vehicle which uses camera and laser range inputs to stay on the road. Another significant contribution in road following has been shown in the DARPA challenge by the vehicle Stanley [2]: it was capable of driving at relatively high speeds through diverse and unstructured off-road environments using machine learning and probabilistic reasoning for planning. However, Stanley was only able to drive in static environments, without the ability to deal with dynamic obstacles.

In [3], RL was applied to the continuous control of autonomous vehicles. They applied the Neural Fitted Q Iteration (NFQ) [4] method for teaching a controller to steer a car based on continuous inputs. However, although the policy manages to steer the car, it is not optimal with respect to jerk minimization. In a similar direction, [5] relies on the Asynchronous Actor Critic (A3C) framework for end-to-end race driving. They use data from a single front camera as an input, and their agent learns to drive on various racing tracks. However, they were not able to teach the agent to avoid scene objects in all cases. A case study showing that an agent can learn a low-level control task from scratch by using RL was conducted in [6]. They teach an agent to steer a front wheel in order to drive as close as possible to a predefined course. The work of [7] relies on a policy-gradient RL approach to teach an agent to follow a front vehicle safely. The agent learned steering and braking control based on three continuous input features. Even though learning low-level control has been shown to work, complex scenarios with multiple other dynamic obstacles remain challenging. Furthermore, mature control techniques for maneuver execution exist, which can be used to focus on high-level decision making instead [8], [9].

[1] Branka Mirchevska, Christian Pek, and Moritz Werling are with BMW Group, D-85716 Unterschleissheim, Germany. Branka.Mirchevska@bmw.de, Christian.Pek@bmw.de, Moritz.Werling@bmw.de

[2] Matthias Althoff is with the Department of Computer Science, Technical University of Munich, D-85748 Garching, Germany. althoff@in.tum.de

[3] Joschka Boedecker is with the Department of Computer Science, Freiburg University, D-79110 Freiburg, Germany. jboedeck@informatik.uni-freiburg.de

In the deep RL domain, the works in [10], [11] teach an agent to perform continuous actions directly from sensor data. The lateral control system of [10] indirectly learns the vehicle-road interaction dynamics for vision-based road following. It relies on a robust image processing algorithm, combined with neural networks-based RL. The agent in [11] learns to map an input image to a small number of key perception indicators that directly relate to the affordance of a road/traffic state for driving. Again, learning low-level control is a challenging task and has slow learning rates.

Finally, we review the high-level decision making RL-based approaches which are most related to our work. The work of [12] teaches an agent to reach a highway exit in minimal time. They use a binary occupancy grid as an input to a convolutional neural network for finding the best action, which is then masked-out by a simple safety checker if unsafe. Besides their high-level actions, they include discrete actions for accelerating and decelerating, increasing complexity due to the integration of the maneuver execution. Furthermore, their safety approach is not described and not proven to be formally correct. As opposed to our approach, they are dealing with a finite horizon problem, learning to reach a predefined highway exit.

In our previous work [13], we use fitted Q-learning for high-level decision making by applying extremely randomized trees [14] as a function approximator. Our agent was able to perform high-level decision-making on a busy simulated highway. However, we only used a two-lane highway, simplifying the problem and decreasing the chance of causing collisions as only one lane change option has to be considered at any time.

C. Contributions

This paper proposes a DQN-based RL approach [15] for learning safe high-level decision-making for autonomous driving on highways. More specifically, we present a novel approach that is able to

1) learn the high-level decision making of performing lane changing or lane keeping on highways with an arbitrary number of lanes while a low-level system executes the maneuver,
2) use a minimal representation of the environment to reduce the dimensionality of the state space for accelerated learning, and
3) incorporate safety verification in the system to ensure that the agent only executes safe actions.

We train the RL agent to drive as close as possible to a desired velocity. However, other driving goals can be integrated as well. By incorporating a computationally efficient safety verification method, our approach guarantees collision-free learning even in real traffic. We are able to train a neural network of small size and demonstrate the results of our learning strategy by comparing the RL agent to an existing rule-based approach.

The remainder of this paper is organized as follows: Sec. II introduces the RL problem statement. In Sec. III, we describe the learning process using Deep Q Networks (DQN). Subsequently, our procedure to verify the safety of actions prior to execution is explained in Sec. IV. The training procedure is explained in Sec. V. The application and the results of the presented approach are summarized in Sec. VI. We finish with conclusions and future work in Sec. VII.

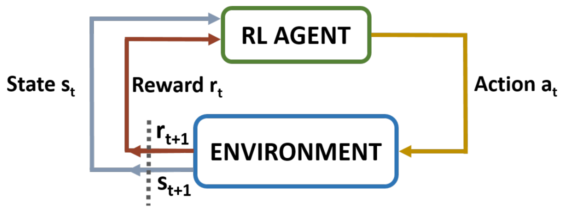

Fig. 1. Based on the current state s_t at time t, the RL agent performs an action a_t in the environment, resulting in a new state s_{t+1}. The RL agent receives a reward r_{t+1} for the chosen action.

II. Problem Statement

We aim to teach an RL agent to make high-level decisions in highway scenarios by dealing with an infinite continuous state-space and a discrete action space, with the goal of driving as close as possible to the desired velocity. We consider a three-lane simulated highway with an arbitrary number of vehicles as illustrated in Fig. 2.

All other traffic participants are controlled by a simulation environment developed by the Institute for Automotive Engineering Aachen and the BMW Group [18] using a complex set of rules based on expert knowledge, which consists of environment, driver, and vehicle models. Chosen high-level actions from the RL agent are executed by a low-level motion planner performing the lane changes. The agent that is entirely controlled by the simulation is referred to as the *rule-based agent* throughout this work. The RL agent does not have any prior information on the intentions of the other vehicles or on the logic of the underlying system.

A. General RL problem definition

The control problem we consider can be described as a Markov Decision Process (MDP) [16]. An MDP is described by a set S of states, a set A of actions, a stochastic transition function $p(s, a, s')$ describing the system behavior, and an immediate reward function $r : S \times A \to R$, where R is the set of rewards. The goal is to find an optimal policy $\pi^* : S \to A$ that maximizes the expected cumulative rewards for each state. In particular, we allow S to be continuous and assume A to be finite for our RL agent and p to be unknown to our learning agent (model-free approach).

The RL agent and the environment interact during a sequence of discrete time steps $t = 0, 1, 2, ...$, as shown in Fig. 1. At each time step t the RL agent receives information about the environment's state $s_t \in S$ and based on that, selects an action $a_t \in A(s_t)$. One time step later, as a result of the action performed in s_t, the RL agent receives a reward $r_t \in R$, and finds itself in a new state s_{t+1}. A policy π is a mapping $S \to A$ that maps each state s to an action that should be executed when in s. The goal of the RL agent is

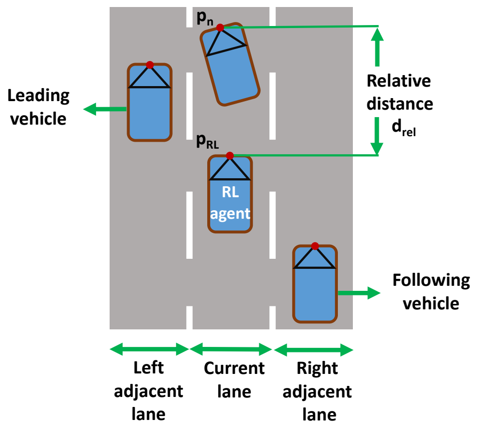

Fig. 2. The environment of the RL agent consists of three lanes with multiple other leading and following vehicles, which are controlled by the underlying rule-based system.

finding a policy to achieve the highest reward in the long run. The Q^{π} function yielding the policy π is formally defined as the expected cumulative future discounted reward $E\{R_t\}$ starting from state s at time-step t, performing the action a, and following the policy π afterwards [17, Eq. 3.13]:

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\{R_t | s_t = s, a_t = a\} = \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}, \end{aligned} \tag{1}$$

where γ is a discount factor between 0 and 1 [17, p. 352]. The state-value function which yields the optimal policy π^* is called optimal state-action value function Q^*, defined as [17, Eq. 3.17]:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall s \in S, a \in A. \tag{2}$$

B. State description

We assume that the agent in Fig. 2 can only see the leading and the following vehicles on its own lane and both its adjacent lanes. Let us introduce $p_{RL} \in \mathbb{R}$ as the longitudinal position of the RL agent and $p_n \in \mathbb{R}$ as the longitudinal position of the n-th surrounding vehicle in a curvilinear coordinate system of the lane (see Fig. 2). Furthermore, $v_{RL} \in \mathbb{R}$ and $v_n \in \mathbb{R}$ denote the absolute velocities of the RL agent and the n-th vehicle, respectively. The relative distance and velocity of the n-th surrounding vehicle are obtained as $dr_n = p_n - p_{RL}$ and $vr_n = v_{RL} - v_n$. We include

- the *relative distance* $dr \in \mathbb{R}$,
- the *relative velocity* $vr \in \mathbb{R}$ for each of the six surrounding vehicles, and
- the *absolute velocity* $v_{RL} \in \mathbb{R}$ for the ego vehicle

into the state representation. When there is no adjacent lane available, or there is no vehicle in an existing adjacent lane to consider, we set the relative distance to $dr = \pm\infty$, and the relative velocity to $vr = 0$. Our state vector s is composed of 13 continuous values:

$$\begin{aligned} s = \big[& dr_{ll}, vr_{ll}, dr_{ol}, vr_{ol}, dr_{rl}, vr_{rl}, v_{RL}, \\ & dr_{lf}, vr_{lf}, dr_{of}, vr_{of}, dr_{rf}, vr_{rf} \big]^T, \end{aligned}$$

where the indices for each of the surrounding vehicles indicate whether the vehicle is in the left, its own or the right lane, and whether the vehicle is leading or following, relative to the ego vehicle.

C. Actions

Let us first define the discrete set of actions[1] A:

a$_1$: *perform a lane change to the left*,
a$_2$: *keep the lane*, and
a$_3$: *perform a lane change to the right.*

We choose A this way since we want to teach the RL agent to perform high-level decisions in highway scenarios, and in general, these actions are the minimal set for achieving that goal. After one of these actions is chosen from the RL agent at a certain time step t, our safety system checks the safety of the chosen action. If the action is safe, the underlying system takes care of the low-level execution. The duration of a complete lane change maneuver is fixed to 3.5 seconds. We choose discrete actions, instead of low-level controls like steering or accelerating, since modularized systems have been reported to perform better in autonomous driving than end-to-end systems [19].

D. Reward Function

To reward behavior that maximizes the velocity of the vehicle, we choose the following reward function:

$$r_t = -\left| v_{RL,t} - v_{des,t} \right|,$$

where $v_{des,t}$ is the desired velocity of the RL agent in time step t.

III. RL METHOD

We apply the Deep Q Network (DQN) approach [15], which is based on Q-learning but leverages the power of neural networks for approximating the Q state-action value function. Alg. 1 describes the offline batch mode DQN approach that we use for learning.

Batch mode reinforcement learning is a sub-field of dynamic programming-based reinforcement learning [20]. It originally refers to a mode of RL in which the whole set (batch) of transition samples is known beforehand, from which the agent should derive the optimal policy (aka offline learning). The batch mode RL process can be divided into 3 separate phases, as shown in Fig. 3:

1) Collecting the data as a set of transition samples:
$$\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1}) \,|\, i = 1, ..., k\},$$
2) performing the batch mode RL algorithm for learning the best possible policy from the available data, and

[1]Note that the actions A only consider high-level decisions of the agent in lateral direction. Acceleration until the desired velocity has been reached is controlled by the low-level execution layer.

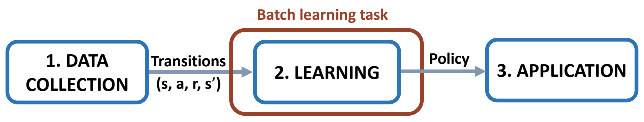

Fig. 3. Illustration of batch reinforcement learning.

3) applying the learned policy to a particular problem.

The actual batch mode learning task takes place only in the second phase and depends neither on the data-collection phase nor the application phase. With that in mind, we need to ensure that the agent has seen the relevant state-action combinations during training often enough. If successful, the RL agent is able to learn behaving optimally in unseen situations (within an ϵ range of the seen situations) during the application phase. *Optimally* in this case means to maximize the discounted cumulative reward the RL agent receives in the long run.

After acquiring the set of transition samples $\mathcal{D}$, we create the training samples by using the Q update as a target for each state s_j as shown in line 2 of Alg. 1. In each training iteration, we select a mini-batch from the training set, i.e., a subset of the complete training set $\mathcal{D}$. In each training iteration, we perform gradient descent on the loss between the target y_j (c.f. Alg. 1) and the estimated $Q(s_j, a_j; \theta)$ values for the particular mini-batch. We repeat this procedure until some predefined number of training iterations is reached, or until the difference between the target and the estimated values does not decrease anymore. It is worth mentioning that our approach achieves the same results by training online for a longer time. For training in real traffic in more challenging scenarios, online (or the combination of offline and online) learning is the reasonable choice. We used offline learning on the collected data for quickly trying out different architectures.

IV. Verifying Actions As Safe

Machine learning techniques are not capable of guaranteeing the safety of the agent at all times. Formal verification methods, on the other hand, are suited for verifying the safety of the agent in a computationally efficient way. Thus, combining ML techniques and formal verification provides a way for safe learning, even in real traffic.

Algorithm 1: Batch Deep Q-learning [15]

```
Collect a set of transition samples $\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1})\}$
Initialize action-value function $Q$ with random weights $\theta$
Repeat
1     Sample random mini-batch of transitions $s_j, a_j, r_j, s_{j+1}$ from $\mathcal{D}$
2     Set $y_j = \begin{cases} r_j, & \text{terminal } s_{j+1}. \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta), & \text{non-terminal } s_{j+1}. \end{cases}$
3     Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$,
      with respect to the network parameters $\theta$.
4 Until Termination conditions are reached
```

In order to guarantee that the agent only chooses safe actions at all times, we make use of formal verification for lane changes, as proposed in [21], which we briefly recall: According to traffic rules [22], the agent must keep a safe distance Δ_{safe} from other vehicles so that it can stop without colliding when the leading vehicles perform emergency braking. This safe distance is formally derived in [23] and accounts for the dynamics of both the agent and the other traffic participant. Thus, we can define safe states of the agent (position p_{RL}) with respect to the leading vehicle (position p_ℓ) and safe distance $\Delta_{\text{safe},\ell}$ as

$$p_{\text{RL}}(t) < p_\ell(t) - \Delta_{\text{safe},\ell}(t). \qquad (3)$$

Analogously, safe states with respect to the following vehicle with position p_f and the safe distance $\Delta_{\text{safe},f}$ are defined as

$$p_f(t) + \Delta_{\text{safe},f}(t) < p_{\text{RL}}(t). \qquad (4)$$

Consequently, we can define the safe free space $\mathcal{F}^t$ of the agent in a lane (cf. Fig. 4) by combining (3) and (4).

Definition 1 (Safe Free Space):
The safe free space $\mathcal{F}^t$ of the agent within a lane at a point in time t is defined as

$$\mathcal{F}^t := \left\{ p \in \mathbb{R} \,|\, p_f(t) + \Delta_{\text{safe},f}(t) < p < p_\ell(t) - \Delta_{\text{safe},\ell}(t) \right\}.$$

By enlarging the shape of each vehicle by its dimensions, collision-checks are reduced to checking the enclosure of its reference point in $\mathcal{F}^t$ [24].

Other vehicles may accelerate or decelerate over time. However, we assume that other vehicles do not drive backwards once they have stopped and that they do not accelerate beyond a velocity v_{max}. For instance, $v_{\max}$ may correspond to the speed limit multiplied with a parameterizable speeding factor $\phi \geq 1$. The maximum positive acceleration α of following vehicles with respect to the velocity v is modeled as

$$\alpha(v) = \begin{cases} \alpha_{\max} & \text{if } 0 \leq v < v_s, \\ \alpha_{\max} \frac{v_s}{v} & \text{if } v_s \leq v < v_{\max}, \\ 0 & \text{if } v_{\max} \leq v, \end{cases} \qquad (5)$$

where v_s is a parameterized switching velocity as described in [25] and $\alpha_{\max}$ corresponds to the maximal feasible acceleration of the vehicle. When accelerating, following vehicles are also obliged to respect safe distances to the leading vehicles (e. g., to the agent when considering the safe free space of the agent's lane) according to traffic rules [22].

By ensuring that the agent is driving within the safe free space (1), we are able to verify a given action a as safe prior to execution (cf. Fig. 4). Let us first introduce $\mathcal{F}_0^t$ as the safe free space of the agent's lane. Furthermore, $\mathcal{F}_l^t$ and $\mathcal{F}_r^t$ denote the safe free space on the left adjacent and right adjacent lanes[2], respectively. We use $p_a(t)$ to describe the position of the agent while executing the action $a \in \mathcal{A}$ over

[2]If a lane does not exist, we set its safe free space $\mathcal{F}^t = \emptyset$.

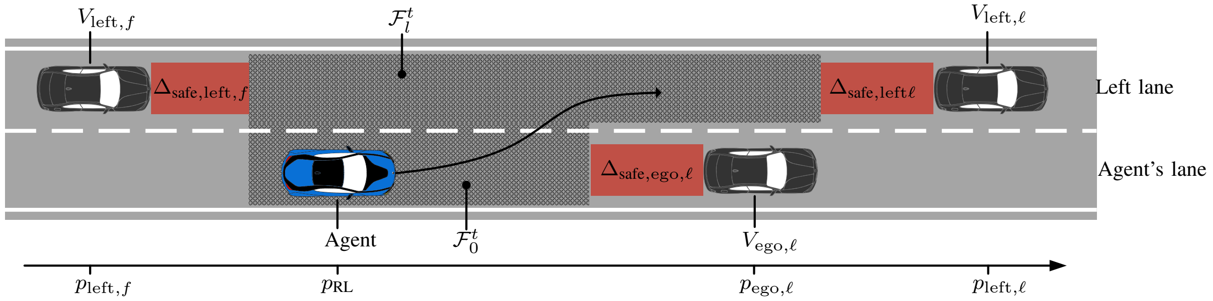

Fig. 4. Visualization of a lane change maneuver to the left lane: The agent with initial position p_{RL} driving behind a leading vehicle $V_{\text{ego},\ell}$ wants to change to the left lane. This lane contains two vehicles, $V_{\text{left},f}$ and $V_{\text{left},\ell}$. The safe distances are illustrated in red and the safe free spaces, $\mathcal{F}_0^t$ and $\mathcal{F}_l^t$, as a shaded area. If the agent is located inside the safe free space at any time, its safety is guaranteed.

time interval $t \in [t_0, t_h]$, where t_h is the finite horizon of action a.

Proposition 1 (Set of Safe Actions):
The set of safe actions $\mathcal{A}_{\text{safe}}(t) \subseteq \mathcal{A}$ for a point in time t is defined as

$$\mathcal{A}_{\text{safe}}(t) := \big\{a \in \mathcal{A} \,|\, p_a(t) \in \big(\mathcal{F}_0^t \cup \mathcal{F}_l^t \cup \mathcal{F}_r^t\big), \forall t \in [t, t_h]\big\}.$$

Proof: The soundness of Prop. 1 has been shown for all cases in [21]. ■

We assume that the agent starts in a safe state, i. e., the agent initially respects safe distances to leading vehicle. If the agent always chooses safe actions $a \in \mathcal{A}_{\text{safe}}(t)$, the agent remains in the safe free space indefinitely unless other traffic participants violate traffic rules.

V. Training phase

We collected 10^5 transition samples, which correspond to approximately 90 hours of driving in our simulation environment. One time step covers a duration of $\Delta t = 3.5\,\text{s}$, which is the time the maneuver execution requires to finish the chosen lane change.

In order to encounter as many diverse situations as possible during the data collection phase, we do not rely on randomly-collected data. Instead, we build a simple data-collecting agent whose goal is to perform safe lane changes whenever possible. The agent collects positive training samples that would increase its reward by changing in lanes that allow driving faster. Negative samples are collected when a certain chosen action decreases the velocity of the RL agent. For successful training, we ensure that the agent collects a balanced data set consisting of both positive and negative training samples.

In a preprocessing step, we perform min-max normalization over the offline training data, i.e.,

$$s_{\text{norm}} = \frac{s - \min(S)}{\max(S) - \min(S)}, \forall s \in S, \tag{6}$$

to speed up training [17, p. 226]. The neural network that estimates our Q state-action values during training has a 13-100-100-3 topology, which means 13 input features for the state vector, 2 hidden layers with 100 neurons each, and 3 output neurons for the Q values for each of the 3 possible actions. As an activation function for all 3 layers, we used ELU (exponential linear unit) and as an optimization algorithm, the Adam optimizer [26]. We chose the above parameters empirically, after trying out different architectures with different optimizers and activation functions. Tab. I summarizes all parameters used for training.

TABLE I
Training parameters

Number of input neurons	13
Number of hidden layers	2
Number of neurons in each hidden layer	100
Connection between layers	Fully connected
Number of output neurons	3
Activation function	ELU
Mini-batch size	5
Weights initialization	$\frac{1}{\sqrt{num\_hidden\_neuons}}$
Bias value initialization	0.01
Gamma	0.99
Learning rate for Adam	1E-5
Number of training iterations	~100

VI. Application phase and evaluation

After the data collection and the learning phase of the batch RL task are done, we can employ the learned policy. Fig. 5 shows the decision-making process, from observing a new state to choosing a safe action. After forwarding the input through the 2 hidden layers, as an output we get the estimates for $Q(s, left)$, $Q(s, keep)$, and $Q(s, right)$. We choose the action with maximum Q value and forward it to the safety algorithm. If the action is considered safe, it is executed; if not, we take the second best action. If that one is also unsafe, we stay in the current lane. Note that the action *keep* is always safe, since the RL agent respects the safe distance to leading vehicles at any time (cf. Sec. IV). While the safety algorithm guarantees that the agent does not cause an accident, unavoidable accidents are still possible: for example, if someone crashes into our vehicle from the side or rear-ends it at the end of a traffic jam. In our case, the underlying system prevents rear-end collisions.

In order to evaluate the performance of the RL agent, we created 10 simulated highway traffic scenarios. Each scenario lasts 500.5 seconds, which means the RL agent needs to make 143 decisions (one decision each 3.5 seconds). In each

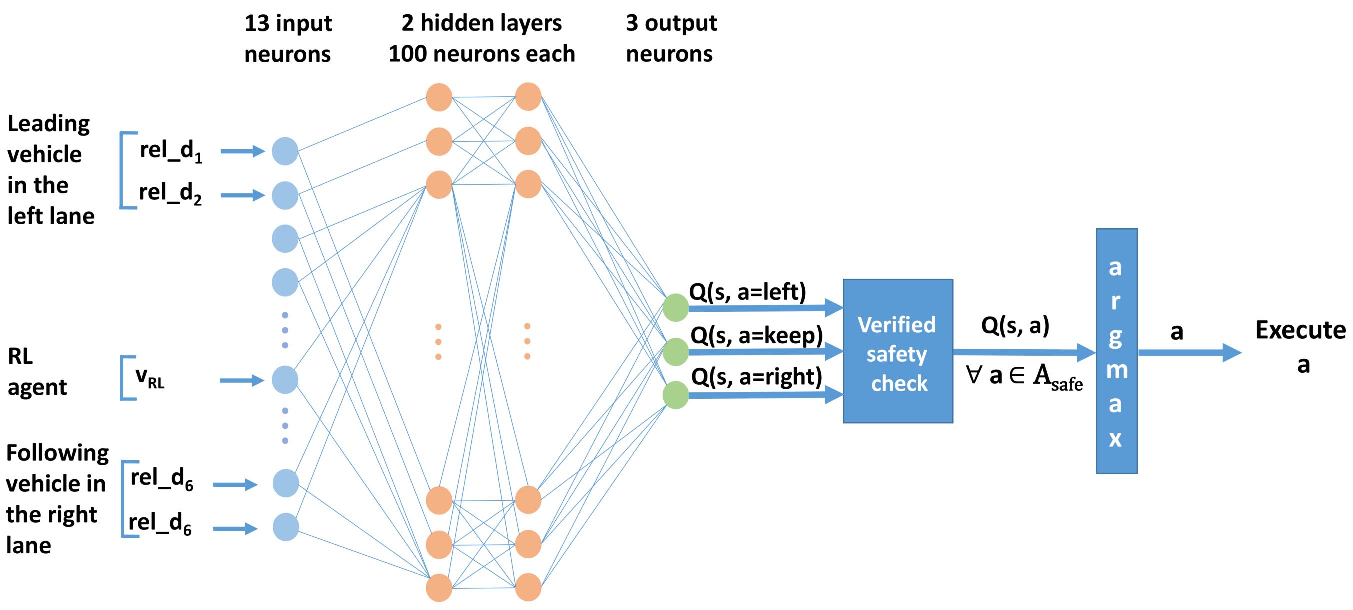

Fig. 5. In the application phase, the neural network takes the new unseen state as an input and outputs the estimated Q values for each of the three actions. Before execution, the safety verification algorithm proves the safety of the one with the highest Q estimate.

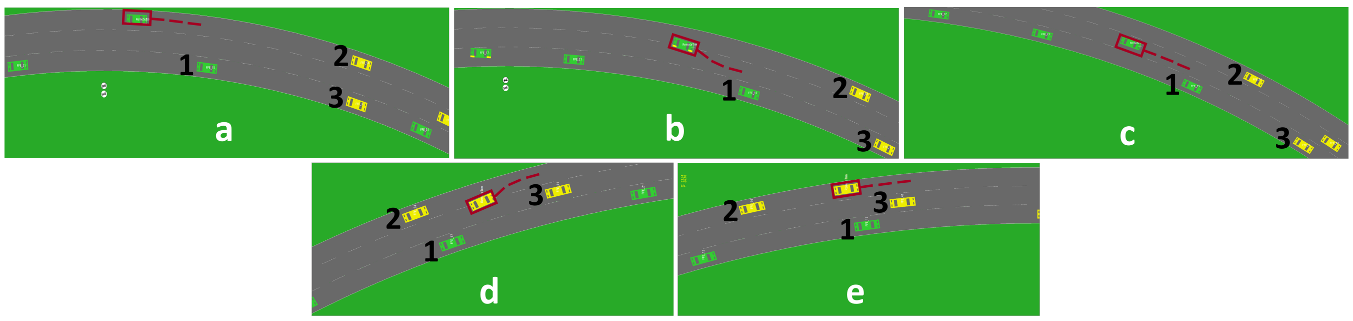

Fig. 6. The trained RL agent (shown in the red rectangle) executes two lane changes in order to overtake slower leading vehicles. a) the agent observes the slower leading vehicle 2 in its lane, b) executes a lane change to the right lane, c) observes the slower leading vehicle 3, d) executes a lane change left, and e) continues in the left most lane.

scenario, around 50 other vehicles are involved. We chose this number based on the length of $l = 1255\,\mathrm{m}$ and the width $w = 11.25\,\mathrm{m}$ of the highway, in order to create situations with high traffic densities, as realistic as possible.

The desired velocity of the RL agent was set to $v_{\mathrm{des}} = 19.5\,\mathrm{m/s}$ as the curvature of the highway constrains the maximal velocity to $v_{\max} = 24\,\mathrm{m/s}$. Each of the surrounding vehicles of the RL agent are randomly positioned on the highway with random initial velocities and are assigned arbitrary desired velocities $10\,\mathrm{m/s} < v < v_{\max}$. As mentioned before, the RL agent does not have any information about the intentions of the other traffic participants, and they change lanes and increase/decrease their velocities arbitrarily.

For comparison, we let the rule-based agent drive on the same test scenarios that we created and measured its average velocity as well. We observe that the RL agent achieves average velocity over all 10 scenarios of $17.3\,\mathrm{m/s}$, whereas the rule-based agent achieves $17.1\,\mathrm{m/s}$.

If we take a look at each of the 10 scenarios separately in Fig. 7, where on the x-axis we have ordered all 10 test simulations, and on the y-axis we plot the average velocity, we can see that the RL agent (shown in red) exceeds the average velocity of the rule-based agent (shown in green) in 7 out of 10 test scenarios.

Fig. 6 shows the performance of the trained RL agent in a segment of one of the testing simulations. The RL agent is marked with a red rectangle and its trajectory is shown with a red dashed line. The leading surrounding vehicles are marked with numbers from 1 to 3. Starting from a), the agent performs two lane changes. It first changes from the leftmost lane to the middle one (b and c), since vehicle 2 would have slowed it down. Afterwards, it encounters vehicle 3 which drives slower than the RL agent as well. The RL agent then decides to perform another lane change from the middle to the leftmost lane (d and e), where it is able to accelerate to the desired velocity.

In order to show the effectiveness of the safety layer, we let the trained RL agent drive on the same 10 test scenarios,

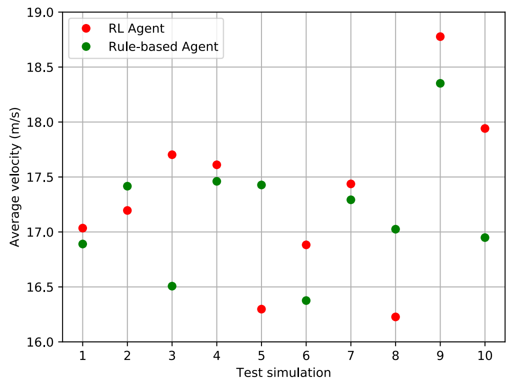

Fig. 7. Average velocity of the RL vs. the rule-based agent

with the safety layer turned off. As Tab. II shows, the agent crashed in 9 out of 10 scenarios, on average approximately after one third of the simulation time has passed.

TABLE II
RL AGENT PERFORMANCE WITHOUT THE SAFETY LAYER ON THE SAME 10 TEST SCENARIOS

Test simulation:	1	2	3	4	5	6	7	8	9	10
Crash after x sec:	249	96	no crash	159	30	208	412	219	40	327

VII. Conclusions

We present a reinforcement-learning-based approach for autonomous and safe lane-changing on simulated highways. The goal for the RL agent is to drive as close as possible to a desired velocity. We rely on the batch Deep Q Network approach for model-free, offline learning. This allows the agent to learn the task without knowing the dynamics of the system including the intentions of the other vehicles. We incorporate safety verification to guarantee that the RL agent performs safe actions at all times.

Our results show that our trained agent is able to achieve high velocities close to the desired speed limit while never causing any collision. Moreover, our trained RL agent shows better performance than a baseline rule-based agent with respect to the achieved average velocity in different scenarios.

Acknowledgments

The authors thank Carmella Schürmann for the voiceover in the video attachment. This work is partially funded by the German Federal Ministry of Economics and Technology through the research initiative Ko-HAF (`https://www.ko-haf.de/`).

References

[1] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989, pp. 305–313.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the DARPA grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[3] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *Proc. of the IEEE Int. Conf. on Frontiers in the Convergence of Bioscience and Information Technologies*, 2007, pp. 645–650.

[4] M. Riedmiller, "Neural fitted Q iteration–First experiences with a data efficient neural reinforcement learning method," in *Proc. of the European Conference on Machine Learning*, 2005, pp. 317–328.

[5] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," *arXiv preprint arXiv:1807.02371*, 2018.

[6] M. Lauer, "A case study on learning a steering controller from scratch with reinforcement learning," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2011, pp. 260–265.

[7] C. Desjardins and B. Chaib-Draa, "Cooperative adaptive cruise control: A reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1248–1260, 2011.

[8] 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*, vol. 1, no. 1, pp. 33–55, 2016.

[9] D. Gonzalez, J. Perez, V. Milanes, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[10] S.-Y. Oh, J.-H. Lee, and D.-H. Choi, "A new reinforcement learning vehicle control architecture for vision-based road following," *IEEE Transactions on Vehicular Technology*, vol. 49, no. 3, pp. 997–1005, 2000.

[11] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. of the IEEE Int. Conf. on Computer Vision*, 2015, pp. 2722–2730.

[12] 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.

[13] B. Mirchevska, M. Blum, L. Louis, J. Boedecker, and M. Werling, "Reinforcement learning for autonomous maneuvering in highway scenarios," in *Workshop for Driving Assistance Systems and Autonomous Driving*, 2017, pp. 32–41.

[14] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] R. A. Howard, *Dynamic programming and Markov processes*. Wiley, 1966.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[18] F. K. GmbH Aachen, "Pelops white paper," *Technical Report*, 2010.

[19] S. Shalev-Shwartz and A. Shashua, "On the sample complexity of end-to-end training vs. semantic abstraction training," *arXiv preprint arXiv:1604.06915*, 2016.

[20] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement learning*. Springer, 2012, pp. 45–73.

[21] C. Pek, P. Zahn, and M. Althoff, "Verifying the safety of lane change maneuvers of self-driving vehicles based on formalized traffic rules," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 1477–1483.

[22] Economic Comission for Europe: Inland Transport Committee, "Vienna Convention on Road Traffic," 1968. [Online]. Available: http://www.unece.org/fileadmin/DAM/trans/conventn/crt1968e.pdf

[23] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, and T. Nipkow, "Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL," in *Integrated Formal Methods*, 2017, pp. 50–66.

[24] J. Ziegler and C. Stiller, "Fast collision checking for intelligent vehicle motion planning," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2010, pp. 518–522.

[25] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 719–726.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. on Learning Representations*, 2014.