Controlling an Autonomous Vehicle with Deep Reinforcement Learning*

Andreas Folkers[1] Matthias Rick[1] Christof Büskens[1]

***Abstract*— We present a control approach for autonomous vehicles based on deep reinforcement learning. A neural network agent is trained to map its estimated state to acceleration and steering commands given the objective of reaching a specific target state while considering detected obstacles. Learning is performed using state-of-the-art proximal policy optimization in combination with a simulated environment. Training from scratch takes five to nine hours. The resulting agent is evaluated within simulation and subsequently applied to control a full-size research vehicle. For this, the autonomous exploration of a parking lot is considered, including turning maneuvers and obstacle avoidance. Altogether, this work is among the first examples to successfully apply deep reinforcement learning to a real vehicle.**

I. INTRODUCTION

Self driving cars have the potential to sustainably change modern societies which are heavily based on *mobility*. The benefits of such a technology range from self-providing car sharing to platooning approaches, which ultimately yield a much more effective usage of vehicles and roads [1]. In recent years, great progress has been made in the development of these systems, with a major factor being the results achieved through deep learning methods. One example is the neural network PILOTNET, which was trained to steer a vehicle solely based on camera images [2], [3].

More classic methods divide the processing of sensor data and the calculation of vehicle controls into separate tasks. The latter can be achieved by various model-based control approaches, one method being the linear quadratic controller [4], also known as Riccati controller. It minimizes a quadratic objective function in the state deviation and control energy while taking into account a linear model of the underlying system. There are various examples for the application of this technique to autonomous driving [5], [6], [7].

While a Riccati-controller is comparably fast, however, it does not directly allow the consideration of constraints such as obstacles or more advanced objective functions within the optimization. Such requirements are met by a general nonlinear model predictive control (MPC) approach based on solving an optimal control problem in every time step. Although the calculations required are considerably more complex, such methods were successfully implemented for autonomous vehicles, e. g., [8] or most recently [9] utilizing efficient solvers such as TRANSWORHP [10] based on WORHP [11].

A combination of the advantages of both, the speed of the Riccati controller and the generality of MPC, can be achieved by finding a function that maps state values to control variables, e. g., by training a deep neural network. Such a model could, for example, be learned supervised, as done for PILOTNET, or by reinforcement learning. The latter in particular led to excellent results in the training of such agents for controlling real-world systems such as robots [12] or helicopters [13].

Recent work also shows promising applications of reinforcement learning for autonomous driving by making strategic decisions [14], [15] or by the computation of control commands [16], [17], [18]. Although these results show the success of this approach in simulated environments, there are very few examples of evaluations on real vehicles. One of them was presented by the WAYVE research team where a policy for lane following is learned based on corresponding camera images. Training is done onboard, with the only feedback for improvement coming from the intervention of a safety driver. While this method works when training is carried out without the proximity of real obstacles and at low speeds, it may be difficult to implement this approach for more general situations.

In this work, we show how to realize the autonomous exploration of a parking lot based on deep reinforcement learning. In particular, this setting is much more challenging than simple lane following due to sharp turning maneuvers and road constrictions caused by obstacles. To this end, we describe how a policy is trained to compute sophisticated control commands which depend on an estimate of the current vehicle state. This is done by designing an appropriate Markov decision process and a corresponding proximal policy optimization [20] learning algorithm. For that purpose a simulated environment is used for data generation. Performance of the resulting *deep controller* is evaluated in both, simulation and real-world experiments. To the best of our knowledge, this work extends the state of the art results for successfully driving an autonomous vehicle by a deep reinforcement learning policy.

II. CONTINUOUS DEEP REINFORCEMENT LEARNING

In recent years, various methods from classical reinforcement learning [21] have been combined with neural networks and their optimization through backpropagation [22], leading to algorithms such as Deep Q-Learning [23], [24], [25], and several actor-critic-methods [18], [26], [27]. In particular, the latter class of algorithms allows agents to be trained with continuous action spaces, which is crucial for their

*The authors would like to thank *Deutsches Zentrum für Luft- und Raumfahrt (DLR) Raumfahrtmanagement, Navigation* in Bonn-Oberkassel for providing the research vehicle [grant number 50NA1615].

[1] WG Optimization and Optimal Control, Center for Industrial Mathematics, University of Bremen, Germany afolkers@uni-bremen.de

application as controller on a real-world system like an autonomous vehicle. Our training procedure is based on the proximal policy optimization (PPO) algorithm [20]. For this, an infinite Markov decision process (MDP) is considered, defined by the six-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}_{sa}^{s'}, \mathcal{P}_0, r, \gamma)$, with $\mathcal{S}$ and $\mathcal{A}$ being the continuous and bounded spaces of States and Actions, respectively. The probability density function $\mathcal{P}_{sa}^{s'}$ characterizes the transition from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ given action $a \in \mathcal{A}$ while $\mathcal{P}_0$ incorporates a separate distribution of possible start states. The reward function is denoted by $r : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and γ is the discount factor.

With these definitions the goal of a reinforcement learning algorithm is to find an optimal policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ representing the probability density of the agent's action, given a certain state. Optimality is specified in relation to the expected discounted return

$$\begin{aligned} &\eta(\pi) := \mathbb{E}_{s_0, a_0, \ldots} \left\{ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \right\}, \\ &s_0 \sim \mathcal{P}_0(s_0), \quad a_t \sim \pi(a_t|s_t), \quad s_{t+1} \sim \mathcal{P}_{s_t a_t}^{s_{t+1}}, \end{aligned} \tag{1}$$

with $r_{t+1} := r(s_{t+1}, s_t, a_t)$. While in the actor-critic setting the policy π is identified with the actor, the state value function

$$V^{\pi}(s_t) := \mathbb{E}_{a_t, s_{t+1}} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right\}$$

will behave as the critic which is responsible to evaluate the policy's actions. Correspondingly, the state action value function and the advantage are given by

$$\begin{aligned} &Q^{\pi}(s_t, a_t) := \mathbb{E}_{s_{t+1}, a_{t+1}} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right\} \quad \text{and} \\ &A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t). \end{aligned}$$

To find a (nearly) optimal policy in continuous state and action spaces, it proved helpful to use neural networks as function approximators, which leads to parameterized π_θ and V^θ. Altogether, a deep actor-critic training algorithm has to find parameters θ to both, maximize the true target (1) and approximate the corresponding state value function V^θ.

Regarding the former, proximal policy optimization is based on a first order approximation of η around a reference policy π_{θ_0} for the local optimization of the parameters of π_θ. The distance between both is approximately measured by

$$\xi_t^{\theta_0}(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_0}(a_t|s_t)}.$$

The samples (s_t, a_t) to be evaluated are generated within the computation of a rollout set $\mathcal{M}$. For this purpose a total of N data points is computed by following the reference policy π_{θ_0}. If a terminating state is reached in episodic tasks, a new start state is sampled with respect to $\mathcal{P}_0$. Altogether, the PPO objective for training the policy π_θ is given by

$$\begin{aligned} \zeta_\pi(\theta) := -\mathbb{E}_{(s_t, a_t) \in \mathcal{M}} \Big(&\min \big[\xi_t^{\theta_0}(\theta) \hat{A}_t, \\ &\operatorname{clip}(\xi_t^{\theta_0}(\theta), 1 - \varepsilon, 1 + \varepsilon) \hat{A}_t \big] \Big) \end{aligned} \tag{2}$$

which introduces a pessimistic balancing of two terms controlled by the clip parameter $\varepsilon > 0$ [20]. The advantage of the t^{th} data point is approximated by $\hat{A}_t$ which can further be used to define a cost function ζ_V for V^θ as the quadratic error

$$\zeta_V(\theta) := -(\hat{A}_t)^2. \tag{3}$$

Finally, a robust approximation $\hat{A}_t$ can be computed using the generalized advantage estimation [28] given as

$$\hat{A}_t^{\gamma, \lambda} := \sum_{k=0}^{\infty} (\gamma \lambda)^k \delta_{t+k}^{\theta},$$

which allows a sophisticated trade-off between bias and variance through the parameter $\lambda > 0$. Since the rollout set $\mathcal{M}$ is given by trajectories, this estimate can be computed for each data point by summing up to the end of the corresponding episode. Our implementation of the PPO method is summarized in Algorithm 1.

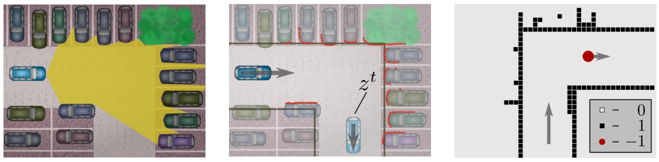
Algorithm 1: Proximal policy optimization

```
Initialize $\pi_\theta$ and $V^\theta$;
while not converged do
    Set $\pi_{\theta_0} \leftarrow \pi_\theta$;
    Generate a rollout set $\mathcal{M}$ following $\pi_{\theta_0}$ and
      compute $\hat{A}_t^{\gamma,\lambda}$ for each data point;
    for K steps do
        Draw a random batch of $M$ data points from
          $\mathcal{M}$;
        Update $\pi_\theta$ and $V^\theta$ using a stochastic gradient
          descent algorithm with backpropagation and
          the cost functions $\zeta_\pi$ (2) and $\zeta_V$ (3);
    end
end
```

III. DEEP CONTROLLER FOR AUTONOMOUS DRIVING

This work is part of the research project AO-CAR whose objective is the development of algorithms for navigation and optimal control of autonomous vehicles in an urban environment. Here, information about the vehicle's surrounding are measured by, e. g., laser scanners and are further extended by a rough knowledge about the geometry of the drivable area (see Fig. 1). Based on this, a desired target state z^t, including a speed value, is defined as illustrated in Fig. 1b for the exemplary situation of the autonomous parking lot exploration. The measurements and targets are updated at high frequency, ultimately resulting in a control loop. The task of the corresponding controller is to provide steering and acceleration values at every iteration, so that a safe trajectory to the target is obtained. Within this setting, all other vehicles are assumed to be static [9].

Training a deep reinforcement learning agent to implement a controller for solving this task involves the definitions of a corresponding MDP and the topologies of the neural networks π_θ and V^θ in accordance to it. Finally, an appropriate environment has to be designed in which the network parameters are learned through Algorithm 1. Here, in contrast



(a) Sensors. (b) Target state z^t. (c) Perception $\emptyset$.

Fig. 1: During the exploration of a real parking lot, obstacles are perceived by sensors (a). Together with its knowledge about the drivable area, a current target state is defined (b). The surrounding from the perspective of the vehicle can be described by a coarse perception map where the target is represented by a red dot (c).

to [19], training is preformed within a simulation in order to quickly obtain a model as general as possible from various and diverse situations. Within the resulting episodic MDP, the policy has to make the vehicle reach the desired target state including a specified speed value. In particular, this should lead to a controlled stop, if the latter is zero. After this training step, the resulting agent can be deployed to control a real-world vehicle. Details of the simulated MDP are presented in the following.

A. State and Action Spaces

The research vehicle can be controlled algorithmically by specifying the steering wheel angle ν as well as the longitudinal acceleration a. The former can be mapped bijectively to the mean angle of the front wheels, defined as β. To prevent arbitrary fast changes, the angular velocity $\omega = \dot{\beta}$ is defined as part of the action space instead of β or ν, leading to

$$\mathcal{A} := \{(a, \omega)^\top \in \mathbb{R}^2 \mid \text{both bounded}\}.$$

These control variables have direct influence on the set of vehicle coordinates

$$\mathcal{Z} := \{(x, y, v, \varrho, \beta)^\top \in \mathbb{R}^5 \mid v, \varrho, \beta\, \text{bounded}\}$$

as shown in Fig. 2. The tuple (x, y) describes the position of the center of the vehicle's rear axle with respect to an inertial system. The speed in the longitudinal direction is called v, where $\dot{v} = a$, and the orientation of the vehicle with respect to the inertial system is referred to as ϱ.

Both, position and orientation of the current target $z^t \in \mathcal{Z}$, can be expressed by the bounded relative position (x^r, y^r) with respect to the vehicle coordinates $z \in \mathcal{Z}$ and the complex number representation $(\varrho^r_{\Re}, \varrho^r_{\Im})$ of the corresponding relative orientation. The form of the latter has the advantage of avoiding discontinuities. Moreover the controller needs to know the desired target speed as well as the current speed of the vehicle in order to predict its next states and to allow safe driving maneuvers. The target steering angle β^t is not of interest, which ultimately leads to the first part of the state space

$$\tilde{\mathcal{S}} = \{(x^r, y^r, v, v^t, \beta, \varrho^r_{\Re}, \varrho^r_{\Im})^\top \in \mathbb{R}^7 \mid \text{all bounded}\}.$$

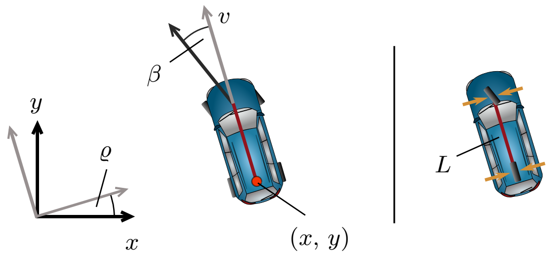


Fig. 2: Left: Coordinates of the vehicle state vector. Right: Single track model simplification.

The second component is given by the vehicle's relative perception of its surrounding, combined with the a priori knowledge about the drivable area and the position of the target. This can be defined by a coarse grid $\emptyset$ of size $n \times m$ with entries in $\{-1, 0, 1\}$, as shown in Fig. 1c. Finally the MDP state space is specified by

$$\mathcal{S} := \{(\tilde{s}, \emptyset) \mid \tilde{s} \in \tilde{\mathcal{S}}, \emptyset \in \{-1, 0, 1\}^{n \times m}\}.$$

B. State Transitions

The action-dependent transition between states of a real vehicle can be incorporated within the simulation using a system of differential equations to describe its behavior. This leads to an update of the relative vehicle coordinates as well as a new measurement of the obstacle perception. For low speeds, kinematic considerations such as simple single-track-models as in [29] or [30] are sufficient. Here, the vehicle is assumed to have only one wheel at the front and back respectively, each centered between the real ones as shown in Fig. 2. This leads to the system of equations

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\varrho} \\ \dot{\beta} \end{pmatrix} = \begin{pmatrix} v \cos(\varrho) \\ v \sin(\varrho) \\ a \\ {}^{v}\!/\!{}_{L} \tan(\beta) \\ \omega \end{pmatrix},$$

where L is the vehicle-specific wheelbase. Further physical constraints, such as a limited steering angle, can be directly considered within the simulation.

C. Reward Function

The reward function should encode the goal of the agent to reach the given target position with an appropriate orientation and speed. However, since the MDP at hand is continuous, it is almost impossible to ever fulfill this task starting with a random policy and to learn from such a success. This is in particular true in the difficult case of a target speed of zero, which requires an exact stopping maneuver at the end of the corresponding episode.

As a result, training is performed in two phases and two separate policies, DRIVER and STOPPER, are learned depending on the task. While the former is rewarded for quickly reaching the desired speed, the STOPPER should approach the target slowly and stop in the end. For both models, the first learning phase rewards proximity at every time step of the episode. Reaching the final position or rather performing a stop is especially highly rewarded. For the exemplary case of the DRIVER this results in

$$r_1^D := c_p\overline{\Delta p} + [r_0]_p + [{}^1\!/\!{}_2 r_0]_{p,\varrho}, \quad c_p, r_0 \in \mathbb{R}_+,$$

where $\overline{\Delta p}$ measures the squared proximity to the target and $[\cdot]_p$ (or $[\cdot]_{p,\varrho}$) is only granted if the agent reaches the desired position (or position and orientation respectively). As soon as a policy learned to fulfill the goals of the first phase, additional behaviors such as an appropriate speed and small steering angles are taken into account, which is given by

$$r_2^D := r_1^D + c_v\overline{\Delta v} + c_\beta\overline{\Delta\beta}, \quad c_v, c_\beta \in \mathbb{R}_+$$

in case of the DRIVER. Here, $\overline{\Delta v}$ and $\overline{\Delta\beta}$ measure the proximity of speed and current steering angle. For learning the STOPPER model, similar rewards apply and are complemented with, e. g., an additional weight on the desired speed in the first phase.

D. *Policy and Value Function as Neural Networks*

As suggested in Sect. II, two neural networks are trained to be identified as policy π_θ and value function V^θ. Here, both share the same topology but not the same parameters. The state component $\tilde{s} \in \tilde{\mathcal{S}}$ is processed by two dense layers of each 200 ReLU activations, as shown in Fig. 3. On the other hand, the evaluation of the perception map $\varnothing$ is based on two convolutions, which allow to learn from the structure of the input. The spatial dimension is halved by a max pooling operation both times. While the first convolution consists of $C = 30$ feature maps $\{\Sigma_c\}_{c=1,\dots,C}$, the latter is reduced to only one, which is then flattened and also processed by a dense layer of 200 ReLU activations. The result of both inputs is then concatenated and passed through a last 200 ReLU layer.

Finally, the output of the value function model V^θ is computed by a subsequent dense layer of one linear activation. Furthermore, the return of the policy π_θ is a pair (μ, σ) for each possible action, which is identified with the mean and standard deviation of a gaussian probability distribution. While the former is computed based on a $\tanh$ activation, the latter is defined to be independent from the input, which yields a general measure of how certain the model is about its actions. In particular, the noise introduced by σ controls the policy's level of exploration when defining the rollout set $\mathcal{M}$.

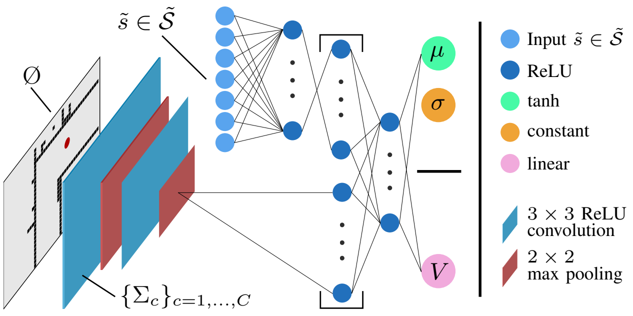

$\tilde{s} \in \tilde{\mathcal{S}}$
$\varnothing$
$\{\Sigma_c\}_{c=1,\dots,C}$
μ
σ
V
Input $\tilde{s} \in \tilde{\mathcal{S}}$
ReLU
tanh
constant
linear
3×3 ReLU convolution
2×2 max pooling

Fig. 3: Neural network topology of policy π_θ (top) and value function V^θ (bottom).

Furthermore, it results in robustness against disturbances in the expected state transition with respect to the most preferred action μ.

Constraints on the control commands can be incorporated by properly scaling the $\tanh$ activation and an additional clip operation in the case of action selection based on the normal distribution.

IV. TRAINING AND RESULTS

We evaluate our training procedure and present the performance of the resulting policy in simulation and on a real vehicle. Learning is realized on a GTX 1080 GPU while the simulator is conducted by an Intel Xeon E5 CPU kernel. Within the latter, random control tasks are generated as shown in Fig. 4. The reward is shaped to make the policy drive the vehicle to the target state, which would define the end of an episode. Alternative termination criteria are the collision with an obstacle or the boundary polygon, exceeding a speed value of $3.3\,{}^{\text{m}}\!/\!{}_{\text{s}}$ ($12\,{}^{\text{km}}\!/\!{}_{\text{h}}$) as well as reaching the maximum time step $T = 250$. The simulated time between two such steps is defined to be 100 ms. The control values a and ω are bounded by $\pm 1.2\,{}^{\text{m}}\!/\!{}_{\text{s}^2}$ and $\pm 1.2\,{}^{\text{rad}}\!/\!{}_{\text{s}}$ respectively.

Proximal policy optimization is carried out by collecting rollout sets consisting of 16,384 time steps, which are then used within $K = 16$ optimization steps with batches of size $M = 1{,}024$. One such epoch takes on average 150 s, while a maximum of 350 epochs is required for convergence. This results in a training time of less than 15 h. However, policies of similar quality can be obtained even without incorporating other vehicles as obstacles during the learning process, as outlined in the next section. The resulting simulation is much faster without detailed computation of the agent's perception, which leads to optimization cycles of 90 s. This reduces the maximum training time to 9 h.

The typical development of the average reward during training is shown in Fig. 5. As one would suggest, convergence of a STOPPER policy is much slower in the beginning, due to the fact that it is harder for the agent to learn about

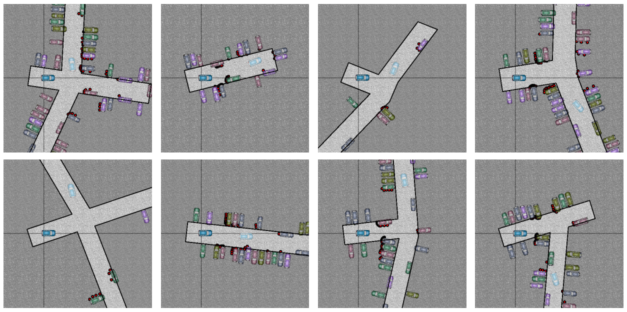


Fig. 4: Randomly selected scenarios computed by the simulator. The initial state is represented by the dark blue vehicle at the center of the coordinate system and includes random values for speed v and steering angle β. The light blue car defines the target. Red dots indicate sensor measurements of obstacle vehicles and the drivable area is defined by a closed polygon.

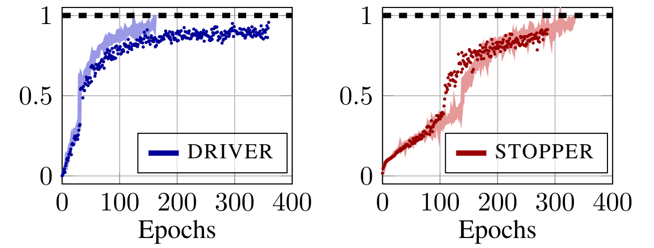


Fig. 5: Normalized average reward during learning of the DRIVER and STOPPER policy. The dotted lines represent training with obstacle vehicles (type A), while the solid lines indicates that they were excluded (type B). Progress is displayed until the respective maximum value is reached.

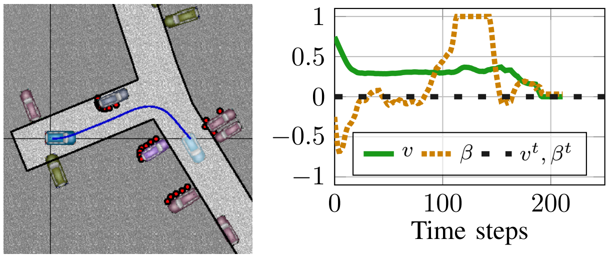


Fig. 6: Exemplary solution of a STOPPER agent which was trained without obstacle vehicles (type B). The values in the right plot are normalized to their respective maximum.

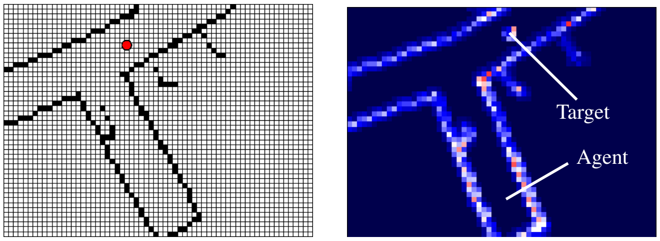


Fig. 7: Perception input $\varnothing$ (left) and its attention map $\mathbb{A}$ (right) of the initial situation in Fig. 6. While dark blue pixels indicate lowest attention, it is at maximum for red ones.

its final goal. DRIVER models usually converge in less than 200 epochs in the simplified setting, which results in a computation time of 5 h. Furthermore, the overall rewards are considerably lower in the case of obstacle vehicles included during learning. Since that task is more difficult, this result is also to be expected.

A. *Results within the Simulation*

We evaluate DRIVER and STOPPER models resulting from learning with obstacle vehicles (type A) and without them (type B), while all other parameters remain identical. In particular, we demonstrate their performance within the simulated environment which is summarized in table I. The evaluation is done based on 10,000 randomly generated control tasks including other vehicles. Results show high success rates with other terminations mainly due to obstacles. Further investigations show that this outcome is very often caused by an infeasible combination of initial orientation, speed and steering angle with respect to the placement of obstacles which would trigger an emergency brake in reality. Even though type B models were only trained based on the polygon representing the drivable area, they still achieve results of similar quality than the type A agent. In case of the DRIVER model they are even better. Most importantly this indicates that type B models are capable of handling unknown obstacles, even if those lead to constrictions on the lane.

This result is further supported by Fig. 6 and Fig. 7 which present an exemplary task solved by a type B STOPPER policy. The former displays the path as well as the corresponding normalized speed and steering values when applying this agent. In particular, the agent is not pulling out before turning to take the obstacle vehicle on the lane into account. Fig. 7 shows the perception map $\varnothing$ of the initial situation in Fig. 6 relative to the agent. To evaluate what parts of it are most important to the neural network, an *attention map* $\mathbb{A}$ can be computed as suggested in [31]. For that the feature maps $\{\Sigma_c\}_{c=1,\dots,C}$ of the first convolutional filter (see Fig. 3) are squared and summed up, yielding $\mathbb{A} := \sum_{c=1}^{C} \Sigma_c^2$. In the situation at hand, high attention is paid to the agent's immediate surrounding as well as to important points in the long term such as the corner, where it has to turn around, or the surrounding of the target. In addition, comparably high attention is paid to the obstacle standing on the lane in front of it. Altogether, these results show that a high quality agent can be obtained even with a simple simulation in short training time.

TABLE I: Comparison of termination criteria and average reward of models trained with (type A) and without obstacles (type B). Values are generated in 10,000 simulation runs including other vehicles. The average cumulated reward per episode is normalized to their respective maximum.

Policy	Obstacle Training	$\varnothing$ Rew.	Termination [%]			
			Succ.	Coll.	Time	Speed
DRIVER	✓ (A)	1.0	90.7	6.1	3.2	0.0
	- (B)	0.984	87.5	10.5	2.0	0.0
STOPPER	✓ (A)	0.967	80.8	11.5	7.8	0.0
	- (B)	1.0	82.9	14.5	2.6	0.0

B. *Results on a Real Vehicle*

After the training in simulation has been completed, a STOPPER and DRIVER model are combined into one deep controller. This is applied within the system for autonomous driving, developed as part of the research project AO-CAR [9], as the main controller during the autonomous exploration of a parking lot (as described in Sect. III). Experiments are performed on a standard Volkswagen Passat GTE Plug-in-Hybrid with additional laser scanners at its front and rear[2].

During exploration, an estimate of the vehicle's state is computed every 20 ms using an Extended Kalman Filter as

[2] For further details visit `www.math.uni-bremen.de/zetem/aocar`

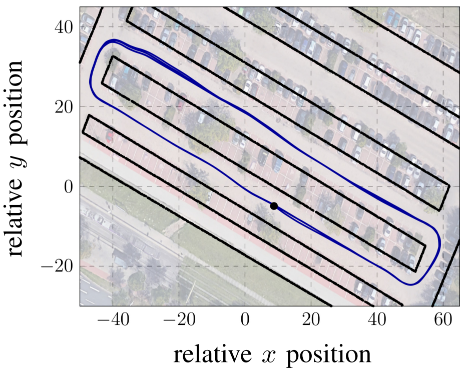


Fig. 8: Exemplary path (blue line) taken by the deep controller when five times circling a real parking area counterclockwise. The positions are relative to the prior knowledge about the parking lot (black lines). The start is marked by a black dot. The underlying satellite image (© 2009 GeoBasis-DE/BKG) is inserted as a reference only (so it does not show the actual occupancy of parking spaces).

described in [32]. From this, a current target state is deduced and corresponding control commands are computed by the controller. Due to the target being updated at high frequency, the vehicle continues driving until a position where to stop is provided. Since the outputs of the deep controller are the acceleration and the steering angle velocity, the latter is numerically integrated for another 20 ms to define the desired steering angle and thus ultimately the vehicle control. The computation of one control command by the neural network representing the policy takes approximately 1.2 ms on an Intel Core i7-4790 CPU.

General characteristics of the deep controller can be evaluated on the basis of Fig. 8. The driven path is in the center of the lane at all times, leading to a high safety of the corresponding trajectory. This applies in particular to the turning maneuvers, after which the vehicle is immediately aligned with the lane again. One can further notice that the paths taken in every turn are very similar to each other. The only deviations occur at the start or in the case of a third party vehicle influencing the control commands (top left).

Fig. 9 shows three specific scenarios which can successfully and safely be handled by the proposed controller. In particular, we overlaid a series of state estimates and the corresponding trajectories that the agent would execute based on the vehicle's current perception. One can see that the controller is able to perform highly challenging maneuvers such as sharp turnings with unknown obstacles. The system is further able to make decisions based on new objects which makes the controller execute an evasive maneuver or a soft stop. In all cases, following the deep controller leads to very smooth driving behavior of the research vehicle. Altogether, the presented method is able to provide sophisticated control commands while still being able to fulfill strict requirements on the computation time.

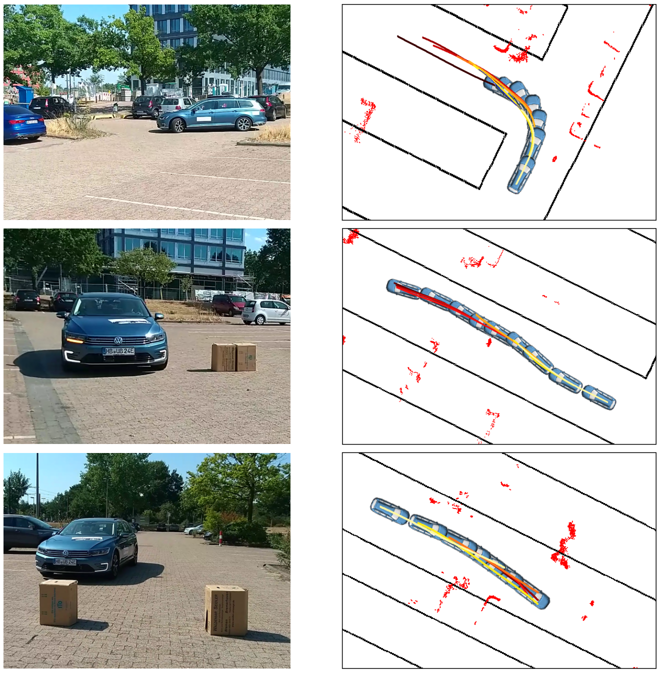


Fig. 9: Application of the deep controller on a real vehicle in three exemplary scenarios. Red dots indicate laser measurements and the drivable area is marked by black lines. The currently planned path of the controller is given by trajectories with a color gradient from light yellow to dark red. Top: Turning. Middle: Driving around an obstacle and stopping afterwards. Bottom: Detection of a blockade on the street and stopping in front of it.

V. Conclusion

We presented a general design approach for a nonlinear controller that is able to provide highly advanced control commands in extremely short computation time. This was done by approximating the control problem within a simulation as a Markov decision process which was then solved by an agent in the setting of a reinforcement learning problem. Training was performed by a proximal policy optimization method with the policy being defined as a neural network. We evaluated our approach in the context of autonomous driving and showed that a high quality controller could be obtained within a few hours of training. Furthermore, we demonstrated its performance on a full-size research vehicle during the autonomous exploration of a parking lot. For example, the controller was able to handle sharp turnings as well as unknown obstacles by performing an evasive maneuver or a stop. In particular, this work is one of the first successful and, to the best of our knowledge, the currently most general application of a deep reinforcement learning agent to a real autonomous vehicle. Future work will include a more detailed analysis of the neural network structure and the state representation as well as applications to further scenarios. The training process as well as the evaluation on the research vehicle are available as a video at `https://youtu.be/1HwHdL7bY3A`.

APPENDIX

TABLE II: Overview of hyperparameters

TRAINING

Descript.	Var.	Value
Steps / episode	T	250
Step length		0.1 [s]
Rollout size	N	16,384
Optim. / rollout	K	16
Minibatch size	M	1,024
Discount	γ	0.99
Gener. advantage	λ	0.95
clip parameter	ε	0.1
Scaling of ζ_V		0.1
Adam optimizer	α	5e−5
(cf. [33])	β_1	0.9
	β_2	0.999
	ϵ	1e−5

REWARD

Descript.	Var.	Value
DRIVER	c_p	0.1
	c_v	0.5
	c_β	0.5
	r_0	50

DYNAMICS

Descript.	Bounds
Acc.	$\lvert a\rvert \leq 1.2$ [m/s^2]
Ang. vel.	$\lvert w\rvert \leq 1.2$ [rad/s]
Steering	$\lvert\beta\rvert \leq 0.55$ [rad]
Speed	$v \in [0, 3.3]$ [m/s]

REFERENCES

[1] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, Eds., *Autonomous Driving: Technical, Legal and Social Aspects*. Heidelberg: Springer, 2016.

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[3] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," 2017.

[4] K. Ogata, *Modern Control Engineering*, 5th ed. Pearson, 2010.

[5] D. Kim, J. Kang, and K. Yi, "Control strategy for high-speed autonomous driving in structured road," *International IEEE Conference on Intelligent Transportation Systems*, pp. 186–191, 2011.

[6] F. Lin, Z. Lin, and X. Qiu, "LQR controller for car-like robot," in *35th Chinese Control Conference (CCC)*, 2016, pp. 2515–2518.

[7] N. Tavan, M. Tavan, and R. Hosseini, "An optimal integrated longitudinal and lateral dynamic controller development for vehicle path tracking," *Latin American Journal of Solids and Structures*, vol. 12, pp. 1006–1023, 2015.

[8] P. Falcone, F. Borrelli, J. Asgari, and D. Hrovat, "Low complexity MPC schemes for integrated vehicle dynamics control problems," *International Symposium on Advanced Vehicle Control*, 2008.

[9] L. Sommer, M. Rick, A. Folkers, and C. Büskens, "AO-Car: transfer of space technology to autonomous driving with the use of WORHP," in *Proceedings of the 7th International Conference on Astrodynamics Tools and Techniques*, 2018.

[10] M. Knauer and C. Büskens, "From WORHP to TransWORHP," in *Proceedings of the 5th International Conference on Astrodynamics Tools and Techniques*, 2012.

[11] C. Büskens and D. Wassel, "The ESA NLP solver WORHP," *Modeling and Optimization in Space Engineering*, vol. 73, pp. 85–110, 2013.

[12] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016.

[13] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 1–8.

[14] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," 2017.

[15] B. Mirchevska, M. Blum, L. Louis, J. Boedecker, and M. Werling, "Reinforcement learning for autonomous maneuvering in highway scenarios," in *Workshop Fahrerassistenzsysteme und automatisiertes Fahren*, 2017.

[16] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging, Autonomous Vehicles and Machines*, pp. 70–76, 2017.

[17] 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," 2015.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48, 2016, pp. 1928–1937.

[19] 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," 2018.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 2010.

[22] D. E. Rumelhart, G. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] 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.

[25] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48, New York, New York, USA, 2016, pp. 1995–2003.

[26] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," 2016.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., vol. 37, Lille, France, 2015, pp. 1889–1897.

[28] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[29] P. Polack, F. Altch, B. d'Andra Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818.

[30] A. D. Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," in *Robot Motion Planning and Control*. Springer, 1998, ch. 4.

[31] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," 2016.

[32] J. Clemens and K. Schill, "Extended Kalman filter with manifold state representation for navigating a maneuverable melting probe," in *19th International Conference on Information Fusion (FUSION)*. IEEE, 2016, pp. 1789–1796.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.