High-Speed Collision Avoidance using Deep Reinforcement Learning and Domain Randomization for Autonomous Vehicles

Georgios D. Kontes[1], Daniel D. Scherer[1], Tim Nisslbeck[1], Janina Fischer[1] and Christopher Mutschler[1]

***Abstract*— Recently, deep neural networks trained with Imitation-Learning techniques have managed to successfully control autonomous cars in a variety of urban and highway environments. One of the main limitations of policies trained with imitation learning that has become apparent, however, is that they show poor performance when having to deal with extreme situations at test time – like high-speed collision avoidance – since there is not enough data available from such rare cases during training. In our work, we take the stance that training complex active safety systems for vehicles should be performed in simulation and the transfer of the learned driving policy to the real vehicle should be performed utilizing simulation-to-reality transfer techniques. To communicate this idea, we setup a high-speed collision avoidance scenario in simulation and train the safety system with Reinforcement Learning. We utilize Domain Randomization to enable simulation-to-reality transfer. Here, the policy is not trained on a single version of the setup but on several variations of the problem, each with different parameters. Our experiments show that the resulting policy is able to generalize much better to different values for the vehicle speed and distance from the obstacle compared to policies trained in the non-randomized version of the setup.**

I. INTRODUCTION

Recent success of Deep Learning in both long-standing research challenges and applied problems paved the way for the adoption of this technology by the automotive industry as a core component for the realization of self-driving vehicles [1]. Deep neural networks not only have replaced classical computer vision algorithms for scene understanding and feature extraction in autonomous vehicles [2], but have also been trained to provide the desired trajectory of the vehicle [3] or even the low-level control actions directly [4], utilizing input from cameras and other sensors.

Most of these networks are trained using Imitation-Learning techniques. Here, the training algorithm is provided several example executions of a task by an expert (e.g. human drivers) and the goal of the learning algorithm is to arrive at a driving policy that not only mimics the actions of the expert, but is also able to generalize the expert strategy to unseen situations.

The main problem with Imitation Learning is that even if a large amount of data is available, it might still not be enough [3]. This is because the source data might not contain critical information on the application environment, e.g. how to recover from error/risk situations that do not appear in the training data. In this case, the training must be augmented with data collected in the application environment to accumulate possible errors and learn a robust policy for this domain [5].

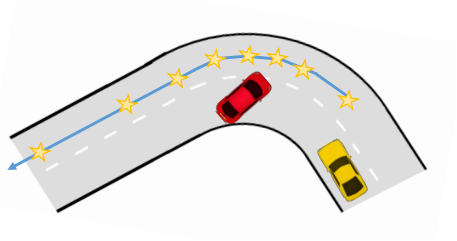

Fig. 1: The agent's goal is to steer its vehicle (in yellow) around the stationary obstacle vehicle (in red) by following a list of waypoints (stars).

This drawback becomes even more limiting if we address "corner cases" in autonomous driving. For example, in this paper we aim at training an Advanced Driver-Assistance System (ADAS) that would take control of the vehicle (steering, acceleration and braking) to avoid a collision – a situation which is usually challenging for human drivers [6]. As this is a rare event in every-day driving, the available data for training will not contain a sufficient amount of imminent collision (and subsequent avoidance) cases.

To tackle this problem, we build upon prior work [7], [8] and propose to utilize a realistic simulator for the design of such a system. For the training, we leverage state-of-the-art (Deep) Reinforcement Learning (RL) algorithms combined with simulation-to-reality (sim2real) practices that have been shown to enable the transfer of learned policies to the real world with minimum or no fine-tuning [8].

Our contributions are the following: i) we design a high-speed collision avoidance scenario in a realistic simulator and train a controller/policy (a deep neural network) using the Twin Delayed DDPG (TD3) RL algorithm; ii) we use a simulation-to-reality transfer methodology (Domain Randomization) to design a robust policy that will perform well in a wide variety of variations of the problem at hand; and iii) we illustrate the generalization performance of the trained policy as opposed to controllers trained on fixed versions of the collision avoidance task.

The remainder of this paper is structured as follows. Section II presents related work from the fields of RL,

*This work was supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology through the Center for Analytics-Data-Applications (ADACenter) within the framework of "BAYERN DIGITAL II".

[1] All Authors are with the Precise Positioning and Analytics Department, Fraunhofer Institute for Integrated Circuits IIS, Nordostpark 84, 90411, Nuremberg, Germany {georgios.kontes, daniel.scherer2, tim.nisslbeck, janina.fischer, christopher.mutschler}@iis.fraunhofer.de

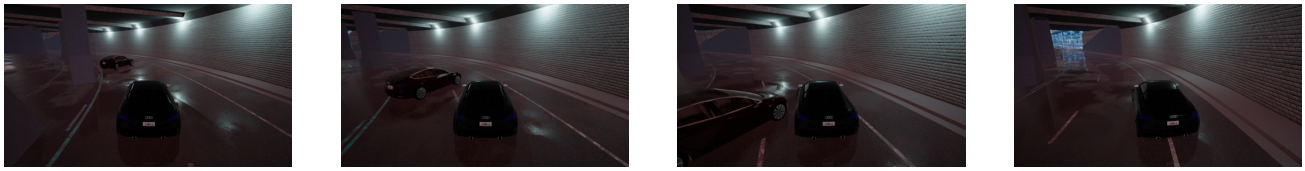

Fig. 2: The ego vehicle avoiding the obstacle.

autonomous driving and robotics; Section III provides an in-depth analysis of the proposed methodology; Section IV presents the evaluation scenario, the experiments performed and the results; Section V analyses the strengths and the possible weaknesses of the methodology and offers insights on future work; and Section VI concludes the paper.

II. Related Work

Our methodology is based on the use of a high-fidelity simulator as a proxy for the real-world collision-avoidance scenario. The policy is trained using Deep Reinforcement Learning in the simulator, where there is room for trial-and-error exploration. As the real world conditions can be quite different from the simulation environment, we leverage a simulation-to-reality transfer methodology called Domain Randomization to ensure that the trained policy can be deployed in the target environment with little or no further tuning

A. Reinforcement Learning for autonomous driving

Recently, Deep Reinforcement Learning has been proposed for tackling several autonomous driving sub-tasks [9], [10]. The training schemes reported in the literature can be classified into two broad categories: i) image-based (also called end-to-end) solutions, which utilize a deep neural network policy that takes camera images as inputs and provides vehicle trajectories or even low-level control actions as outputs; and ii) solutions based on learning so-called affordances [11]. Affordances are high-level features like the distance from the center of the lane, distance from traffic light ahead, speed limit, etc. A Convolutional Neural Network (CNN) is trained to approximate the values of these features in each frame; these are then used as inputs to the deep neural network policy that controls the vehicle.

One of the first real-world applications of end-to-end RL is presented in [12], where the authors combine images from a monocular camera with the observed vehicle speed and steering angle to learn a neural network policy that is able to follow the road lanes.

In simulation, [13] train an end-to-end RL algorithm that controls a race car at high speeds. The action space for the steering, throttle, brake and handbrake actuators is discrete; this design decision allows for faster training without sacrificing the final driving performance, as indicated by the author's results.

Feature-based training on the other hand usually addresses high-level problems, such as safe lane-change or intersection crossing. Here, in most cases, the ego vehicle state is complemented with information about the surrounding vehicles' states (e.g. relative distances, velocity, acceleration, etc.) and the augmented state representation is used as input to the neural network policy. For example, the authors in [14] introduce a risk-based human lane-change model that evaluates the control actions the RL agent is generating for a lane-change scenario, while the authors in [15] explicitly define a network that learns to evaluate the risk of possible actions and thus steer training towards agents that make safer decisions, and successfully evaluate their approach in lane-keeping and intersection-crossing scenarios.

A notable exception where RL is used for both high- and low-level control in autonomous driving scenarios is the work presented in [11]. The authors manage to train a policy using Reinforcement Learning, that is able to successfully drive a simulated vehicle in an urban environment, by constantly solving tasks like traffic-light and road-sign detection, pedestrian and vehicle avoidance, lane keeping, complying to traffic rules, etc. They achieve this by pre-training a CNN to extract specific features like semantic segmentation of the image input, distance and angle from the center of the current road lane, status of visible traffic lights and presence of intersections; these features are then used as an input to the driving policy network that is trained using RL algorithms. The authors argue that with this decomposition of the problem they are able to train more complex networks and with less samples compared to end-to-end approaches.

Our work differentiates from the papers presented above along two main axes: i) we explicitly study high-speed collision avoidance for autonomous vehicles, which is a problem with different requirements compared to the relatively low-speed settings of urban driving scenarios or lane-change highway settings; ii) we fuse the training of our policy with a simulation-to-reality transfer approach aiming at designing a robust policy that can be transferred to the real-world application with minimum effort.

B. Domain Randomization

It is known that Deep Reinforcement Learning algorithms can generate solutions that overfit to the training environment and are not able to generalize – not even to small variations on the training setup [16], especially if the policy representation is highly expressive (e.g. as in the case of deep neural networks). To add to this, RL algorithms are notorious for discovering and exploiting spurious features to maximize their reward on a given task [17].

A simple and straightforward, but also efficient solution to this problem is Domain Randomization (DR) [18], i.e., to train not in a single environment, but on a *distribution of similar environments*, with small variations between them. This

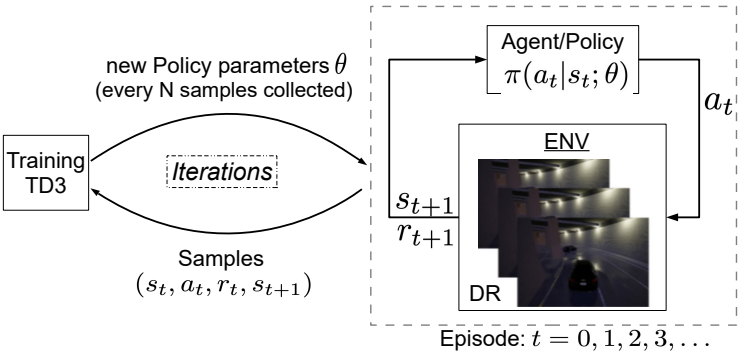

Fig. 3: Schematic of our approach.

way, during deployment, the real environment will appear to the policy as yet another variation. Domain Randomization can take the form of variations in the underlying physics and system dynamics, different levels and types of sensor noise (e.g. blurred and rotated images) or even of an adversary preventing the system from solving a given task [19].

In the autonomous driving domain, the only study on DR to our knowledge is the work presented in [8]. Here the authors train policies in simulation and then apply in real vehicles. The goal is to learn to drive within the road lane in a variety of scenarios. Different levels of domain randomization are applied as variations in the image data and noise in the sensor measurements and several types of policies are trained.

Even though we select our policy structure based on the reported results of [8], our main differences with this work are: i) we study the problem of high-speed collision avoidance, which can be more challenging compared to low-speed lane following settings; and ii) we drastically change the settings of the problem in the DR configuration instead of limiting the variations to only noisy sensor measurements.

III. Methodology

We propose the use of simulation to design a safety system that will take full control of a vehicle before an imminent collision and manage to avoid it. A schematic of our approach is illustrated in Figure 3. Here, we use the Twin Delayed DDPG RL algorithm for the system training in the simulator (i.e. developing our policy/controller). The algorithm collects data from simulation runs and periodically uses these data to design better control parameters for the agent/policy. In the simulation side, the agent utilizes the most recent policy parameters to generate control actions for the vehicle in each simulation timestep. In our approach, instead of defining a fixed simulation environment for the training, we leverage Domain Randomization, i.e. our agent is trained under several variations of the environment. This will allow to transfer the learned policy to the application environment with minimum or no additional training.

A. Separation of Perception and Collision-Avoidance Modules during training

Instead of adopting an end-to-end training approach, where the policy uses images as inputs and provides low-level control actions as outputs, we separate the perception functionality from the control policy. To achieve this, we build upon the work of [20], [11] in learning affordances, as described in Section II-A.

This approach has numerous benefits. First of all, it has been shown to lead to faster training of larger networks with less samples [11]. Second, as the perception and control modules are separated, we can use our method to train the collision avoidance functionality and then leverage any scene understanding and perception technology from any vendor available in the market without needing to retrain an end-to-end solution; our only requirement is that our policy is provided with the appropriate features. Finally, this decomposition leads to more modular and interpretable solutions compared to end-to-end approaches [20].

B. Twin Delayed DDPG

In an RL setup, usually the problem at hand is modeled as a Markov Decision Process (MDP) [21], which is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, \rho)$. Here, $\mathcal{S}$ and $\mathcal{A}$ are finite sets of states and actions, respectively, $\mathcal{T}$ is a transition model that expresses the probability to end up in state s_{t+1} at timestep $t+1$ if the environment is in state s_t and action a_t is applied at timestep t, $\mathcal{R}$ is a reward function with $r_t(s_t, a_t, s_{t+1})$ being a real number indicating the quality of this transition, $\gamma \in [0, 1)$ is a discount factor for future rewards and ρ the initial state distribution, i.e., $s_0 \sim \rho(s)$.

The RL task is that an agent learns which actions to take in specific situations in order to maximize the cumulated, expected future reward. The action selection is determined by sampling from a stationary policy $\pi(a_t|s_t)$ for the MDP, which indicates the probability of selecting an action a_t in state s_t (see also Figure 3).

In the typical RL scenario, we do not know the transition model and we need to learn an optimal policy using only available data. One of the classical algorithms that addresses this problem is Q-learning [22]. The algorithm iterates between estimating the state-action value function $Q(s, a)$ of

the MDP from the available data and using this estimate to infer a better policy until convergence. The state-action value function for a given policy π is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{a_t \sim \pi; s_t \sim \mathcal{T}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \,|\, s_0 = s, a_0 = a \right) \quad (1)$$

and expresses the estimated total reward when taking action a in state s and following the policy π thereafter.

Using the available transitions, the estimate for the Q value is updated in each timestep as follows:

$$\begin{aligned} y_t &= r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) \\ Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha (y_t - Q(s_t, a_t)) \end{aligned} \quad (2)$$

where y_t is called the target, $[y_t - Q(s_t, a_t)]$ is called the Temporal Difference (TD) error and α a learning rate.

The policy-improvement step calculates the improved greedy policy over the estimated Q value function *for all states* s *in* $\mathcal{S}$ as: $\pi_{\text{new}}(s) = \text{argmax}_{a \in A} Q(s,a)$. In practice, we need a policy that explores all parts of the state space. To achieve this, we use the $\epsilon-$greedy policy defined as follows: when the environment is in state s, the best action according to the greedy policy is applied with probability $1-\epsilon$, otherwise a random action is selected from the set of possible actions.

When the state space of the MDP is large (or continuous), then the Q value function can be approximated with a function with features $\phi(s,a)$ and parameters θ. For linear approximators, this can be done analytically, but for non-linear approximators, such as neural networks, training can be unstable and might even diverge [21]. To add to this, when for problems with continuous action spaces, the greedy policy improvement is no longer applicable.

Recently, the authors of [23] managed to implement Q-learning with neural network approximators for the Q value in an algorithm called DQN, which was further extended to continuous action spaces in a new algorithm called Deep Deterministic Policy Gradient (DDPG) [24]. To stabilize learning, instead of using each sample only once (as shown in Equation 2) a replay buffer $\mathcal{B}$ is defined where all the samples are stored. In addition, a second network for the Q estimator with parameters θ' is introduced, called the target network. This network is used to calculate the target y_t for each sample and its weights are updated less frequently compared to the value network. This update can be done by mixing the two network weights with Polyak averaging: $\theta' = \tau\theta + (1-\tau)\theta'$. Finally, to support continuous actions a parametric function μ for the policy (a deep neural network) with parameters ψ is defined, accompanied by a respective target network with parameters ψ'.

In each training iteration, a mini-batch is sampled from the replay buffer and the weights of the value network (parameters θ) are updated using back-propagation with the following loss function:

$$\begin{aligned} \mathcal{L}(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{B}} \big[r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \psi'); \theta') \\ - Q(s_t, a_t; \theta) \big]^2 , \end{aligned} \quad (3)$$

with μ the policy and ψ' and θ' the weights of the policy and value target networks respectively. The weights of the policy are updated by gradient ascent corresponding to the following maximization procedure:

$$\max_{\psi} \mathbb{E}_{s \sim \mathcal{B}} \left[Q(s, \mu(s; \psi); \theta) \right], \quad (4)$$

while keeping value weights θ fixed. As the policy now is deterministic, exploration is achieved by adding noise to the action selection during training.

One of the main issues of DQN and DDPG is that they overestimate the Q values for some actions, which leads to unstable learning [25]. The Twin Delayed DDPG (TD3) algorithm [26] addresses this problem by learning two value networks (and a separate target network for each) and uses the smallest Q value of the two for calculating the target y_i. In addition, to further stabilize learning, two additional modifications are made compared to DDPG algorithm: i) the parameters of the policy and *all* the target networks are updated less frequently compared to the value network; ii) noise is added to the action selected by the target policy network to enable the value network to predict similar actions to neighboring states, thus smoothing out the value predictions.

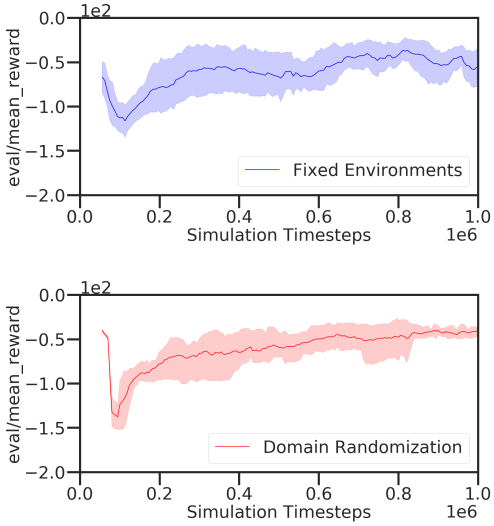

Fig. 4: Evolution of training for the all the agents trained on the fixed environment configurations (upper plot) and agent trained on the DR environment (lower plot).

C. Domain Randomization

As stated before, to enable seamless simulation-to-reality transfer we utilize the Domain Randomization approach. In the work presented here, we choose to randomize the initial speed and the distance from the (static) obstacle blocking the road (see Section IV-D).

One of the main deviations from the Domain Randomization for RL literature is that even though they suggest the use of Long Short-Term Memory (LSTM) policies [19], we use a

much simpler three-layer feed-forward network. The reason for this choice is twofold: i) a smaller network requires significantly less samples to train and much less hyper-parameter tuning compared to LSTM-based networks; and ii) as the problem we address here is simpler compared to the dexterous-manipulation setup defined in [19] for example, a high-capacity LSTM network would be able to memorize the best sequence of actions for all environment variations in our Domain Randomization setup, thus exhibiting poor generalization properties. These points were also observed by the authors in [8], which verified that an LSTM policy showed limited generalization abilities to the real world compared to a feed-forward policy for a lane-following task.

IV. Experiments and Results

A. The Collision Avoidance Setup

To evaluate our method, we define a high-speed obstacle avoidance setup as shown in Fig. 1. The scenario is as follows: the ego vehicle drives into a city tunnel with speed=s_{ego} $\mathrm{km/h}$ using two PID controllers for longitudinal and lateral control and it detects an obstacle on its lane (a stationary vehicle) when its distance from the obstacle is d_{obs} meters; at this point the low-level control of the vehicle (throttle, steering and braking actions) is taken over by a neural network-based policy in order to perform a high-speed maneuver to avoid collision. The task is to design a policy that will consistently avoid the obstacle (i.e. will overtake it from the right side, where there is an opening as shown in Fig. 1), regardless of the initial vehicle speed s_{ego} and distance from the obstacle d_{obs}.

We setup the obstacle-avoidance scenario in a tunnel within the "Town03" world map of the open-source CARLA (v0.9.6) simulator [27]. The vehicle is controlled at 15 Hz and the episode ends after a fixed length of 100 time-steps.

B. States, Actions and Rewards

Following the RL setup, in each simulation timestep, the simulator provides a set of states/observations to the neural network policy which in turn provides a set of control actions to be applied on the simulated vehicle. At the same time, a reward is calculated based on the state of the environment and the selected action.

More specifically, we define the following states (scaled appropriately):

- s_1, s_2: the location of the ego vehicle as (x, y) coordinates in the global frame;
- s_3: the yaw angle of the ego vehicle;
- s_4, s_5: the velocity of the ego vehicle in the x, y directions of the global frame;
- s_6: the yaw rate of the ego vehicle;
- s_7, s_8: the acceleration in the x, y directions;
- $s_9 - s_{11}$: the control actions applied in the previous timestep;
- $s_{12} - s_{16}$: the location, yaw angle and length and width of the bounding box of the obstacle;
- $s_{17} - s_{32}$: we define eight waypoints (represented by their (x, y) coordinates) that indicate the free road lane that the ego vehicle can use to overtake the obstacle, as shown in Fig. 1. These waypoints also serve as a representation of the road curvature that the ego vehicle must follow and in a real-vehicle application would be provided by an available map. As the state representation we use the Euclidean distance and relative angle between the ego vehicle and each of the eight waypoints.

The available control actions are the throttle and brake positions and the steering angle, all normalized in the $[-1, +1]$ interval.

For the reward definition, we have experimented with several configurations. Initially, we tried to penalize the impact of the collision for each collision in an episode, but the training process converged to a local optimum that favored light collisions instead of trying to overtake the obstacle. Due to this, as a second step we defined a reward shaping scheme [28], by utilizing the same eight waypoints used to create $s_{17} - s_{32}$ in the state representation as follows:

1) we identify the waypoints j out of the eight waypoints in the trajectory that still lie ahead (i.e. the ego vehicle has not reached yet);
2) we calculate the Euclidean distance d_{w_j} between the ego vehicle and these waypoints and define the waypoint-related reward r_t^w in each timestep t as the sum of the negative distances of the waypoints that still lie ahead

$$r_t^w = -\frac{1}{10} \sum_{w_j \text{ not passed}} j \cdot d_{w_j} \tag{5}$$

The distance is also multiplied by the waypoint index j, in order to provide an additional incentive to the agent to follow the trajectory.

Even with this reward shaping the trained policy required a large number of samples to discover the obstacle-overtake strategy, as it would again get trapped in a strong local optimum that led the car to make a U-turn and crash in the right wall in some cases. To avoid this, we added a small positive reward term for the vehicle speed s_{ego} as follows: $r_t = \frac{s_{ego,t}}{100} + r_t^w$. This way the policy is encouraged to find solutions that maintain high speed for the ego vehicle, hence forcing it to explore the overtake maneuver.

Note here that our initial reward design involved a Curriculum Learning setup [29]: once the agent learned to overtake the vehicle, we would add another term in the reward function penalizing collisions and continue training. As the results in the following Section indicate, the agent detected collision-free trajectories even without this extra step, so we did not fine-tuned the agent with a collision term in the reward. Still, the final evaluation criterion is whether the agent manages to avoid the obstacle without colliding.

C. Domain Randomization

For the Domain Randomization configuration, we generate 20 variations of the collision avoidance setup by selecting four values for the ego vehicle speed ($s_{ego} \in$

$\{85, 90, 95, 100\}$ km/h) and five values for the initial distance from the obstacle ($d_{obs} \in \{22, 25, 28, 31, 34\}$ m). During training, a new combination of speed and distance is sampled uniformly in the beginning of each episode and the values are fixed until the episode is finished.

D. Evaluation

To evaluate the performance of our solution we define 42 variations of the environment, with $s_{ego} \in \{82, 87, 92, 97, 102, 107\}$ km/h and $d_{obs} \in \{20.5, 23.5, 26.5, 29.5, 32.5, 35.5, 38.5\}$ m. We evaluate each combination of parameters five times and average the achieved total reward to account for small variations in the simulation. Notice that these evaluation sets also contain values that are outside the range used during training to explore the ability of the DR-trained policy to extrapolate. All other values do not overlap with the speed and distance configuration used in training to verify that the policy can interpolate in unseen variations of the environment and has not simply memorized the solutions for all the training configurations.

In addition, we train nine agents without DR for different configurations of the environment. Each agent is trained in an environment with a fixed combination of initial speed and distance from the obstacle, with $s_{ego} \in \{85, 92, 100\}$ km/h and $d_{obs} \in \{20, 25, 30\}$ m being the fixed values. Each of these agents is then evaluated in the same 42 environment realizations as the DR agent. The results will illustrate whether the generalization ability of the final agent is due to the DR-enabled training or the agents trained in fixed environments can also generalize.

The reward for the evaluation setup is different compared to training. Here, *as an evaluation metric for each already trained agent* we use the collision intensity (provided in kg m/s by the CARLA simulator) summed over all collisions occurring in an episode.

E. Training Parameters

For the development of our policy/controller using TD3 algorithm, we utilize the Stable Baselines library [30]. The Actor and the Critic are modeled as three-layer feed-forward neural networks with 1000, 500 and 300 hidden units, respectively. All hidden units have ReLU activation [31].

The replay-buffer size is set to 1000000 transitions and the batch size for each gradient update to 100 samples. The Critic networks (predicting Q values) are updated every 1000 timesteps, whereas the policy and target networks every 2000 timesteps. Each update iteration involves 1000 gradient steps (with new sampled minibatches from the replay buffer for each step).

We use zero-mean Gaussian noise both for the action exploration (with standard deviation set to 0.5) and for the target policy (with standard deviation set to 0.2 and clipping at 0.5). The mixing factor τ for the Polyak averaging of the target networks is set to 0.005.

The discount factor γ is set to 0.998 and the algorithm runs for 1000000 iterations. We use Adam [31] with initial value 0.0025 for the learning rate that is linearly annealed to zero towards the end of training. Finally, we allow the collection of 50000 initial random exploration samples.

F. Results

We train the agents in both fixed (i.e. an agent is trained in an environment with fixed initial speed and distance from the obstacle) and DR-enabled environments until convergence. The evolution of training is shown in Fig. 4, where the upper plot shows the mean and variance of the reward per episode for *all* the agents trained on the fixed environments and the lower plot shows the same information for five training runs of the agent trained in the DR setup.

After each agent is trained, we assess its generalization properties using the evaluation described before. A successful avoidance maneuver is shown in Fig. 2, while Fig. 5 shows the total *evaluation reward per episode (see Section IV-D)* for each of the trained agents with the different environment configurations defined in Section IV-D. The title of each subplot in Fig. 5 indicates the combination of ego vehicle speed and distance from the obstacle each agent was trained with and the colormap indicates the performance of each agent in the different regions/combinations of the ego vehicle initial speed and distance from the obstacle in the evaluation setup.

Starting with the first row, where the agents are trained for initial speed 85 km/h and varying distance, it is evident that some of the agents perform well only within the region of values they are trained on and poorly far from them, while others are more robust. The agent trained with 20 m distance for example performs poorly for higher speeds and distances, with worse performance in the region between 90 and 100 km/h speed and 25 and 30 m distance. Similar observations can be made for the agent trained at 30 m, which exhibits rather poor performance for a region of higher speeds (95 – 100 km/h) compared to the training setup. Finally, the agent trained with 25 m distance generalizes well to unseen environment configurations, except for low-speed high-distance scenarios.

In the second row, agents trained with 92 km/h speed seem to generalize relatively worse compared to the previous ones. Here, the agent trained at 20 m distance generalizes well to large portions of the combined space, except from speeds higher than 100 km/h and distances higher than 30 m. The agent trained at 25 m distance exhibits really poor performance in configurations with speeds higher than 100 km/h and also struggles with distances higher and lower to 25 – 30 m. The agent trained at 30 m shows very strong generalization properties except from the region defined by speeds between 95 and 100 km/h and distances higher than 28 m. Also performs poorly for distances over 34 m combined with low initial speed.

In the third row, the agent trained with 100 km/h speed and 20 m distance exhibits the worst generalization properties, but this is to be expected, since the sequence of actions required to avoid the obstacle when moving at such high speed and at such low distance from it are different compared

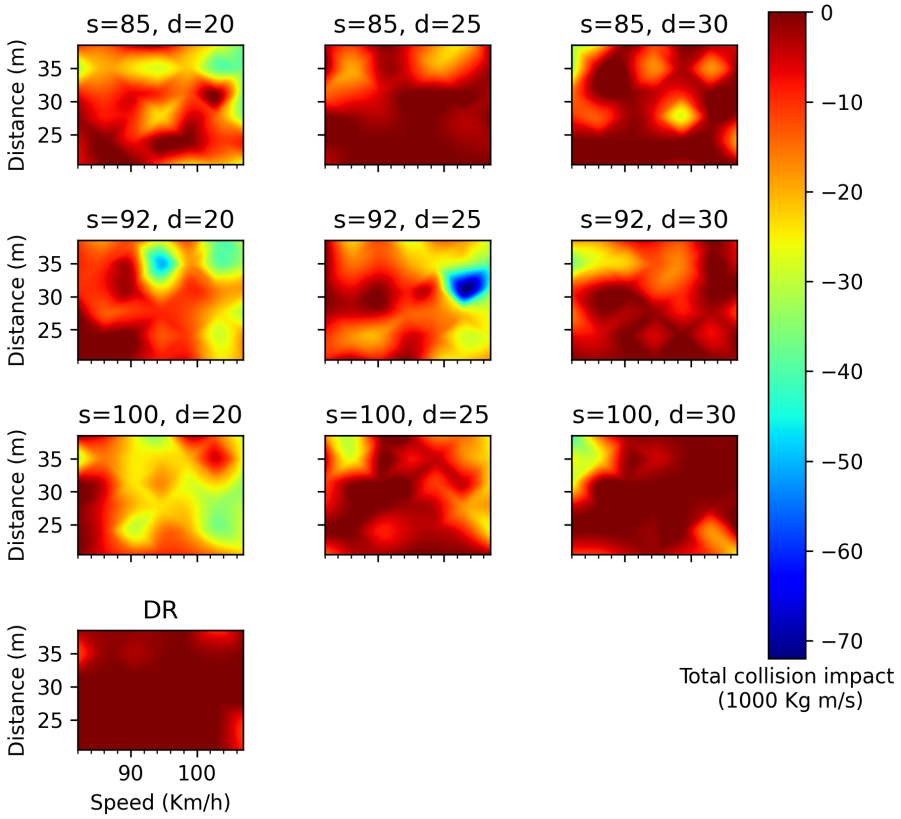

Fig. 5: Results on the evaluation setup, illustrating the generalization properties of each trained agent. The three lines of plots at the top show the results for agents trained on fixed environments while the plot in the last row illustrates the performance of the agent trained with DR. The titles of the subplots indicate which combination of ego vehicle speed and distance from the obstacle each agent was trained with. The reward per episode for each environment variation is the negative sum of the impacts (in $\mathrm{kg\,m/s}$) during the entire episode (divided by 1000 for better visualization) and is depicted for each agent according to the colormap on the right side of the figure. The evaluation runs consist of all combinations of initial ego vehicle speed $s_{ego} \in \{82, 87, 92, 97, 102, 107\}$ $\mathrm{km/h}$ and distance from the obstacle $d_{obs} \in \{20.5, 23.5, 26.5, 29.5, 32.5, 35.5, 38.5\}$ m and are interpolated for more clear presentation.

to configurations with lower speeds and/or higher distances. The agents trained with $25\,\mathrm{m}$ and $30\,\mathrm{m}$ distances on the other hand show very robust behavior to unseen environment configurations. Especially the agent trained with $100\,\mathrm{km/h}$ speed and $30\,\mathrm{m}$ distance fails only for combinations in the low-speed/high-distance or low-distance/high-speed edges of the grid.

Overall, what we can infer from the results is that agents trained in mid to high values for speeds and distances generalize much better compared to agents trained in the low-distance/high-speed combinations. Still, the agent trained with DR (shown in the forth row of Fig. 5) outperforms all other agents. The agent is able to avoid the obstacle without any collisions if the initial speed and distance from the obstacle are within the intervals used for DR training ($s_{ego} \in [85, 100]\,\mathrm{km/h}$) and $d_{obs} \in [22, 34]\,\mathrm{m}$ respectively), except from one case (for $97\,\mathrm{km/h}$ speed and $29.5\,\mathrm{m}$ distance) where there is a small, low-impact hit of the rear right tire with the side-walk in the right side of the tunnel (see Figure 2). On the other hand, we can see a clear drop of performance (i.e. collisions occurring) for combinations that lie outside the region defined by the speed and distance values used in training. This indicates that agents trained with DR *do not necessarily "extrapolate" their performance outside the training environment distribution*.

V. Discussion

The experimental results indicate that the RL algorithm combined with Domain Randomization during training is able to design a policy that manages to avoid the collision with the obstacle in several variations of the problem *consistently*. Even though some of the agents trained in specific combinations of initial speed and distance to the obstacle show good generalization properties, it is not clear why training on environments with these parameters leads to good results and how to reproduce in practice or other setups. Thus DR naturally emerges as a generic methodology to achieve a robust policy that can be seamlessly transferred to the application environment.

Additionally, we have observed that for some evaluation runs the vehicle would crash in the wall several meters *after* it had successfully avoided the obstacle. This of course can be attributed to the specific task the agent solves, which is avoiding the obstacle and not continue with a lane-following functionality once it has successfully avoid it. We can address this simply by handing over control to the normal driving policy once the obstacle has been avoided, as it would be the case in a real-world autonomous vehicle.

VI. Conclusions

In the work presented here we have elaborated on the idea of utilizing simulation for training an active driver assistance system that will take full control of a vehicle and manage to avoid an imminent collision. We combine Reinforcement Learning (RL) for the training of the collision avoidance policy with Domain Randomization to enable seamless transfer from the training to the application domain. We evaluate our approach in simulation under a high-speed collision avoidance scenario and verify that the trained policy is able to avoid the obstacle on the road for a wide distribution of initial vehicle speed and distance from the obstacle.

In terms of future work, we plan to change the input states for the agent from the global coordinate system provided by CARLA simulator to a local coordinate system (to e.g. polar coordinates, with the center of the new coordinate system being located at the obstacle position) and alternatively only use camera images as states. In addition, we plan to examine the possible combination of RL and Model Predictive Control (MPC) as a safer alternative to our approach. Here, the RL layer would make high-level decisions (e.g. overtake the obstacle from the right side following specific waypoints) and the MPC module would be responsible for generating and safely tracking the respective obstacle avoidance trajectory.

References

[1] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, 2019.

[2] K. El Madawi, H. Rashed, A. El Sallab, O. Nasr, H. Kamel, and S. Yogamani, "Rgb and lidar fusion based 3d semantic segmentation for autonomous driving," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 7–12.

[3] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[6] M. Heesen, M. Dziennus, T. Hesse, A. Schieben, C. Brunken, C. Löper, J. Kelsch, and M. Baumann, "Interaction design of automatic steering for collision avoidance: challenges and potentials of driver decoupling," *IET intelligent transport systems*, vol. 9, no. 1, pp. 95–104, 2014.

[7] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*. Springer, 2016, pp. 102–118.

[8] B. Osinski, A. Jakubowski, P. Milos, P. Ziecina, C. Galias, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," *arXiv preprint arXiv:1911.12905*, 2019.

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

[10] 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," *arXiv preprint arXiv:2002.00444*, 2020.

[11] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," *arXiv preprint arXiv:1911.10868*, 2019.

[12] 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 *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[13] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2070–2075.

[14] D. Chen, L. Jiang, Y. Wang, and Z. Li, "Autonomous driving using safe reinforcement learning by incorporating a regret-based human lane-changing decision model," *arXiv preprint arXiv:1910.04803*, 2019.

[15] L. Wen, J. Duan, S. E. Li, S. Xu, and H. Peng, "Safe reinforcement learning for autonomous vehicles through parallel constrained policy optimization," *arXiv preprint arXiv:2003.01303*, 2020.

[16] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *arXiv preprint arXiv:1812.02341*, 2018.

[17] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv preprint arXiv:1711.00138*, 2017.

[18] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[19] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[20] A. Sauer, N. Savinov, and A. Geiger, "Conditional affordance learning for driving in urban environments," *arXiv preprint arXiv:1806.06498*, 2018.

[21] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] 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, p. 529, 2015.

[24] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[25] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[26] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[28] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[29] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[30] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[31] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016.