CARLA Real Traffic Scenarios – novel training ground and benchmark for autonomous driving

Błażej Osiński[*], **Piotr Miłoś**[†], **Adam Jakubowski**[‡], **Paweł Zięcina**[§], **Michał Martyniak**
deepsense.ai

Christopher Galias
Jagiellonian University

Antonia Breuer, Silviu Homoceanu
Volkswagen AG

Henryk Michalewski
Google, University of Warsaw

Abstract

This work introduces interactive traffic scenarios in the CARLA simulator, which are based on real-world traffic. We concentrate on tactical tasks lasting several seconds, which are especially challenging for current control methods. The CARLA Real Traffic Scenarios (CRTS) is intended to be a training and testing ground for autonomous driving systems. To this end, we open-source the code under a permissive license and present a set of baseline policies. CRTS combines the realism of traffic scenarios and the flexibility of simulation. We use it to train agents using a reinforcement learning algorithm. We show how to obtain competitive polices and evaluate experimentally how observation types and reward schemes affect the training process and the resulting agent's behavior.

1 Introduction

The field of autonomous driving is a flourishing research field stimulated by the prospect of increasing safety and reducing demand for manual work. The ability to drive a car safely requires a diverse set of skills. First, it requires low-level control of a car – being able to keep the lane, make turns, and perform emergency stops. It also requires an understanding of the traffic rules (such as speed limits, right of way, traffic lights) and adhering to them. Finally, it requires high-level navigation – deciding which route to the destination is optimal.

Source dataset	Maneuver type	Our custom maps	No. of scenarios	
			train	validation
NGSIM	highway lane change	2	1750	467
openDD	drive through a roundabout	7	51752	12927

Table 1: CARLA Real Traffic Scenarios (CRTS).

Computer-based systems can tackle these different skills with varying levels of success. High-level navigation is effectively solved by widely available GPS-based systems. There is also a growing body of work showing that the car control level can be efficiently solved using

[*] worked performed while at deepsense.ai, now at Lyft and at the University of Warsaw. Correspondence to `b.osinski@mimuw.edu.pl`.

[†] also at the Institute of Mathematics of the Polish Academy of Sciences.

[‡] worked performed while at deepsense.ai, now at AdColony.

[§] worked performed while at deepsense.ai, now at Nvidia.

Machine Learning for Autonomous Driving Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

machine learning Tampuu et al. [2020], Kiran et al. [2020b]. Perhaps the hardest to automate is tactical planning tasks with a time horizon of several seconds, which require interactions with other traffic participants. A classic example is an unguarded 4-way intersection (investigated in e.g. Sadigh et al. [2016]). In this work, we focus on *changing lanes on highways* and *driving through roundabouts*. We present CARLA Real Traffic Scenarios (CRTS), including 9 new maps and a set of interactive traffic scenarios in the CARLA simulator Dosovitskiy et al., which may serve both for creating and evaluating autonomous driving systems. Importantly, these are extracted from datasets collected in the wild: NGSIM Halkias and Colyar [2006] and openDD Breuer et al. [2020] and thus pose real-world challenges. See Table 1 for basic statistics regarding datasets and new maps. Example videos are available at the project website: `https://sites.google.com/view/carla-real-traffic-scenarios/`.

To the best of our knowledge, this is the first publicly available work that adapts tactical-level real-world traffic data to a simulator with rich plausible physics and thus has a number of advantages. First and foremost, it allows for fast and relatively easy testing and comparison between various control methods and their implementation details. These may include planning versus policy approaches, learning algorithms, and methods with different levels of abstraction. Moreover, most recently published datasets (e.g. Krajewski et al., Halkias and Colyar [2006], Geiger et al. [2013]) provide data from a fixed configuration of sensors. Our simulation-based solution removes this restriction and allows for easy data collection and testing of multiple sensor sets.

Figure 1: Scene from openDD dataset together with its in simulation equivalent.

Using datasets listed in Table 1 as a source of trajectories offers an important benefit of facing situations that actually happened in the real world. Based on such trajectories, we build interactive scenarios. In a scenario, we replace a vehicle performing a maneuver with the ego vehicle simulated by CARLA and controlled by an algorithm in a closed-loop manner. The other vehicles follow the trajectory recorded in the dataset. The ego vehicle is tasked to execute the same maneuver. Performance is measured based on the final output (whether the ego-vehicle reached its target location) and negative events during driving (e.g. crashes, rapid turns, changes in speed). In particular, this implies that the policy steering the ego vehicle can use other than original strategies to perform maneuvers. We believe that it is important to understand these strategies if they are different from human driving styles and in what respects. Similarly to Henaff et al. [2019], we think that such an analysis, in the single-agent setting with realistic surroundings, might provide important clues for building multi-agent models. Moreover, one may be able to assess if the policy exploits the simulation's particularities, which might be critical in the goal of transferring the policy to the real world.

Our aim is to provide training and testing ground for other researchers. The set of train scenarios is big enough to train or fine-tune models used for control. We standardize the evaluation protocol by arranging train and validation split. We note that the validation scenarios allow us to test generalization to previously unseen circumstances. CRTS can be useful in various research applications, including learning methods and more classical planning. We note that drone-collected datasets can provided extensive stream of realistic road data as they contain information about multiple and interacting actors. Moreover, they required relatively small hardware investment. The price to pay is that they do not contain precise information about driver actions. For this reason, standard imitation learning, which is commonly used in literature Codevilla et al., is not easily applicable here. In this work we use reinforcement learning (RL) to circumvent this restriction. Another advantage of RL is it can acquire novel solutions or detect corner cases. We show the versatility of CRTS, which stems from the

fact that it runs in simulation. In particular, we compare how various observations modes (bird's-eye view, visual input, and lidar and camera) affect the training and performance of the resulting policies. We also study generalization from the train set to the test set, which has recently become a popular research direction in RL; see e.g. Cobbe et al. [2019], Zhang et al. [2018].

To summarize, we submit the following contributions:

- A set of driving scenarios based on real data in the CARLA simulator, including 9 new custom maps. The source code for the scenarios is available online. See Section 2
- Competitive driving polices utilizing the PPO algorithm Schulman et al. [2017] in various observation and rewards setups. See Section 4.
- Quantitative analysis of these policies with respect to task efficiency and generalization, which indicates that the bird's-eye view observations and dense rewards outperform other tested methods. See Section 4.

2 CARLA Real Traffic Scenarios (CRTS)

We use recent version 0.9.9.4 of CARLA Dosovitskiy et al., an open-source simulator for autonomous driving research based on Unreal Engine 4. CARLA features open assets, built-in maps, weather settings, and multiple vehicles with different physical parameters and various sensors. Two visual quality levels (LOW and EPIC) are supported. We utilize two datasets, NGSIM and openDD, of 2D trajectories collected by drones to obtain a set 3D scenarios with plausible physics delivered by CARLA.

NGSIM In these scenarios, the ego vehicle is tasked with performing a lane change maneuver on a highway. The source of the data is The Next Generation Simulation (NGSIM) dataset Halkias and Colyar [2006], U.S. Department of Transportation Federal Highway Administration [2016], which contains vehicles trajectories collected on American highways. For parsing the data, we used code from Henaff et al. [2019]. We developed two custom CARLA maps of the locations covered by the dataset (southbound US 101 in Los Angeles, CA, and eastbound I-80 in Emeryville, CA).

openDD The openDD dataset Breuer et al. [2020] is a large-scale trajectory dataset recorded from bird's eye view. It focuses on roundabouts, which are an example of interaction-intensive traffic not regulated by traffic lights. openDD is the largest publicly available trajectory dataset recorded from a drone containing over 84k trajectories distilled from 62 hours of video data. The data was collected on 7 roundabouts – five located in Wolfsburg and two in Ingolstadt (both in Germany); see an example in Figure 10. We developed 7 custom CARLA maps corresponding to these roundabouts.

Extraction algorithm Based on the dataset of trajectories we built interactive scenarios. In a given scenario, we replace the vehicle performing a maneuver with the ego vehicle, whose physics is simulated by CARLA and controlled by an algorithm in a closed-loop manner. The other vehicles follow the trajectory recorded in the dataset. The ego vehicle is tasked with executing the same maneuver. Performance is measured based on final output (whether the ego-vehicle reached its target location) and negative events during driving (e.g. crashes or performing the task for too long).

From the NGSIM dataset, we extracted over 2k scenarios of lane change maneuvers and assigned 1750 for training and 467 for validation. From the openDD dataset, we obtained over 64k scenarios of roundabout crossing and assigned 51752 for training and 12927 for validation. Details of the scenario extraction procedure are provided in Appendix A.1.1. The scenarios can be accessed using the standard OpenAI Gym interface Brockman et al. [2016]. The ego vehicle model is Audi A2.

ALC and ADD Artificial Line Change (ALC) and Artificial Roundabout (ADD) are two additional synthetic scenarios mainly intended to be an easy entry into our benchmark. They may also serve as a testbed for algorithms in isolation. They are parametrized by the speed of other traffic participants, which affects their difficulty. See details in Appendix A.1.1.

Metrics We suggest reporting the success rate of the performed maneuver as the primary metric and we recommend that the performance on a test scenario is reported using an average result from at least 50 episodes. As our code is open, other custom metrics can be defined. One could, for example, imagine measuring the comfort of driving.

2.1 Observations and actions

One of the crucial advantages of CRTS is its flexibility due to the use of simulation. In particular, one can easily test various observation settings. In our experiments we provide three major options: *bird's-eye view*, *visual input*, and *lidar and camera*; see examples in Figure 2.

The *bird's-eye view* input covers approximately $47m \times 38m$ meters of physical space encoded in 186×150 pixels. Semantic information is contained in 5 channels corresponding respectively to road, lanes, centerlines, other vehicles, and the ego vehicle. See details in Appendix A.2.1.

In *visual* experiments we use 4 cameras (front, left, right, and rear), each having a field of vision of 90 degrees and a resolution of 384×160. For more details see Appendix A.2.2.

In *lidar and camera* experiments, we use a single front camera and 360° lidar inputs. The simulated lidar has a range of 80 meters, 32 channels, horizontal field of view between -15°, and $+10^\circ$; these parameters resemble lidar settings available in autonomous vehicles (see e.g. Geyer et al. [2020]). In order to be able to use a convolutional neural network to process the lidar input, the raw data (pointcloud) is projected to a 2D view; this is a popular approach (see Yurtsever et al. [2020]).

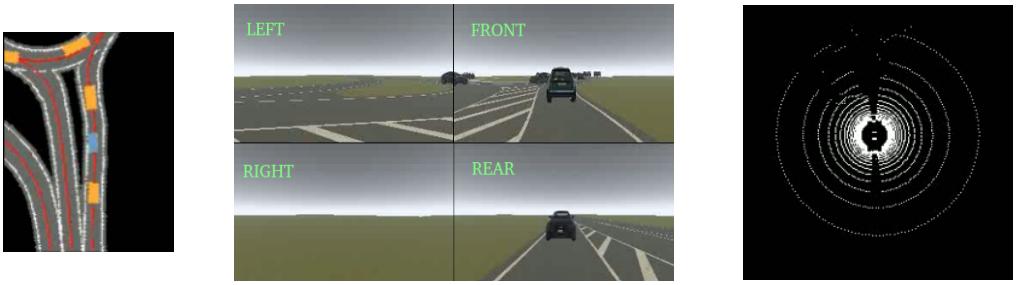


Figure 2: Different observations types: bird's-eye view, 4 cameras, and lidar's projection on 2D view.

The observation also includes a high-level navigational command (either `GO_STRAIGHT`, `TURN_RIGHT`, or `TURN_LEFT`), as inspired by Codevilla et al.. These are used to indicate the lane change direction in NGSIM and the roundabout exit in openDD.

The ego vehicle is controlled by steering and speed. The steering wheel's angle can be set continuously and is processed by the CARLA engine to steer the wheel (in the case of the Audi A2 model, the angle of the wheels can vary from -80° to 80°). The speed is controlled indirectly by issuing commands to a PID controller, which controls the throttle and break. More details are in Appendix A.2.4.

3 Related work

Autonomous driving is one of most important topics in contemporary AI research with a potential for huge impact. It is beyond the scope of this paper to discuss the whole field. We refer to Kiran et al. [2020b], Sadigh et al. [2016], Yurtsever et al. [2020] for general surveys.

Our work concerns several aspects: using realistic simulations, using real-world data, and training end-to-end polices using reinforcement learning.

Over the years, simulations have become a routinely-used tool for studying real-world control problems. Their clear advantages include making data collection cheaper and alleviating safety issues. From them to be useful, the critical research problem of transfer to the real environment has to be solved. Successful attempts, for example Akkaya et al. [2019], usually address the problem in two ways: by making the simulation more realistic and by achieving highly adaptable polices. The latter is often obtained using domain randomization, a technique pioneered in Sadeghi and Levine [2016] (and later extended in many works), which hinges on the assumption that a policy robust to a wide spectrum of simulated conditions would also be applicable to one special instance: reality.

Unsurprisingly, using simulation is an attractive research avenue for the autonomous driving community. The two popular packages CARLA Dosovitskiy et al. and AIRSIM Shah et al. [2018] use Unreal engine to build efficient photo-realistic simulations with plausible physics. Similarly, TORCS Wymann et al. [2000] is a widely-used racing car simulator. There are also other simulators which

are attractive for autonomous driving research; we refer to Rosique et al. [2019] for a comprehensive survey.

On the other hand autonomous driving companies amass ever-increasing datasets of driving data collected in the wild. There is also an increasing number of publicly available datasets. A non-exhaustive list includes NGSIM Halkias and Colyar [2006] and highD Krajewski et al. datasets recorded on highways. inD Bock et al. [2019] and openDD Breuer et al. [2020] concentrate on maneuvers on intersections and roundabouts. KITTI Geiger et al. [2013], A2D2 Geyer et al. [2020], and Houston et al. [2020] datasets consists of footage recorded from cars using various sensors (RGB cameras, lidar) along with various preprocessing. SDD Robicquet et al. and CITR Yang et al. put emphasis on humans and human-car interactions.

Methods using deep neural networks (DNN) and learning have become quite fruitful for autonomous driving. There are three major paradigms of building DNN-based solutions (which are partially analogous to the discussion in cognitive sciences). The first one, *mediated perception*, is perhaps the most prominent today; see e.g. Ullman [1980], Geiger et al. [2013]. It advocates using a vast number of sub-components recognizing all relevant objects (other cars, traffic lights, etc.) to form a comprehensive representation of the scene, which is then fed into a deep neural network. On the other side of the spectrum is the *behavior reflex* approach, which suggests learning a direct mapping from raw observations to control actions; see Pomerleau [1988], Bojarski et al. [2016] for two prominent examples. The third paradigm, *direct perception*, lays in the middle. It recommends learning affordances (e.g. the angle of the car relative to the road), which form a compact representation, which can then be used by another DNN to control the car; see e.g. Chen et al. [2015]. Some other works include Bansal et al. [2018], which utilizes an augmented form of behavioral cloning with perturbed trajectories to successfully train an agent in simulation and shows its performance on a real car. Codevilla et al. introduce high-level navigation commands and propose imitation learning conditioned on them to train an agent in simulation and test its performance on a toy car. Chen et al. show that training an agent in a teacher-student framework (where the teacher has access to privileged information) leads to state-of-the-art performance on previous benchmarks. Shalev-Shwartz et al. [2016a] presents a model-based approach to learning autonomous driving behaviors—a learned RNN model of dynamics is optimized directly with SGD. Shalev-Shwartz et al. [2016b] raises explicit concerns about the safety of RL-based solutions. They observe that policy gradient techniques suffer from high variance when dealing with catastrophic events of low risk. To alleviate that, they propose decomposing the policy into a high and a low level one. The former is learnable, while the latter is hard-coded to ensure safety. Bewley et al. and Osiński et al. [2019] train a reinforcement learning agent in simulation and show successful transfer to a real-world vehicle. As the field grows fast, we only can only scratch the surface; see [Kiran et al., 2020a] for a recent overview of deep reinforcement learning methods used for autonomous driving.

Performing maneouvers is an important aspect of driving. Wang et al. use a deep Q-network with continuous action and state spaces for automated lane change maneuvers. Chen et al. [2018] propose a new state representation and utilize an asynchronous DQN architecture in a multi-agent setting. See [Bevly et al., 2016] for a survey on more classical methods employed in lane change maneuvers.

Another initiative similar to ours is the CARLA autonomous driving leaderboard at `https://leaderboard.carla.org/`. It presents a suite of scenarios, infrastructure, and an evaluation protocol to standardize the evaluation of autonomous driving methods. This aim is in many aspects similar to ours. The key distinction is that our scenarios are modeled using real-life data. Moreover, CRTS is meant to be more of an open-ended research suite than a challenge; thus, it contains a suite of scenarios which is sufficiently large to enable meaningful training and does not impose strong constraints (e.g., sensor choice). All-in-all, these two initiatives (the CARLA autonomous driving leaderboard and CRTS) share many similarities and goals and we imagine they could be merged at some point.

4 Experiments

The main aim of our experiments was to show how CRTS can be used in a versatile way to answer research questions concerning autonomous driving, sometimes arriving at counter-intuitive conclusions. We asked the following specific research questions: a) can reinforcement learning be used to obtain maneuvering polices perform on test scenarios? b) how does the observation mode affect the

training and the quality of resulting polices? c) how do the trained maneuvering strategies compare to realistic (human) ones? Our results are intended to be baselines for other researchers who would like to utilize CRTS. There are numerous further open questions, some of which we list in Section 5.

In our experiments we used a slightly modified PPO algorithm Schulman et al. [2017]; see Appendix A.2.3 for details. PPO is one of the most popular model-free RL algorithms known for its flexibility and stability. Being an on-policy algorithm, it usually requires a substantial number of training samples. Importantly, using a simulator mitigates this issue.

Rewards The choice of the reward function is an important aspect of RL. This can both affect the training performance and the quality of the final policy. Our driving scenarios allow for easy experimentation with various choices. In our benchmark PPO experiments, we used dense rewards for intermediate progress towards completing a maneuver. This, in particular, made our training rather stable and independent of the random initialization. In Section 4.3, we provide more details and present experiments with other rewards schemes. Interestingly, in some cases, a sparse rewards setting offers competitive training performance.

Generalization Generalization is a critical and multi-faceted topic in machine learning. In our experiments, we test generalization between scenarios.[1] We note, however, that CRTS can be used to test generalization in other domains, such as perception (by varying weather setting and other visual details) or dynamics (by varying the car model and physics settings). Tables 2 and 3 presents the success rate on the test set. Details are provided in the subsequent sections.

Our result is a datapoint in the paradigm discussion outlined in Section 3, supporting mediated perception rather than the behavior reflex approach. The importance of these conclusions depends strongly on how well they transfer to the real world. Assessing such transfer is generally an open research question – two popular complementary methods are domain randomization (see e.g. Akkaya et al. [2019]) and quantifying sim-to-real alignment (see e.g. Tan et al. [2018]).

	Base experiments			Bird's-eye ablations		
	bird's-eye	visual	lidar & camera	front only	no centerline	framestack
ngsim	0.787 ± 0.029	0.770 ± 0.022	0.776 ± 0.014	0.782 ± 0.035	0.782 ± 0.016	0.805 ± 0.022
opendd	0.862 ± 0.037	0.886 ± 0.023	0.805 ± 0.106	0.824 ± 0.058	0.833 ± 0.029	0.762 ± 0.083

Table 2: Means and standard deviations of success rates of various models, obtained over three experiments. For each experiment success rate is calculated on the same scenarios from the validation set.

4.1 Bird's-eye view experiments

Bird's-eye view, see Section 2.1 and Figure 4, is a top-down view of the road situation. It contains privileged information, unavailable for normal road actors. It has been popularized in Chen et al. in their two-phase training procedures. It also resembles information that could be extracted from a perception system and HD-maps in a typical AV stack; see e.g. Breuer et al. [2019].

An interesting question is if (and how much) bird's-eye facilitates training; see Section 4.2. Here, we analyze the influence of various elements. Along with the "basic" experiment we ran the following modifications: *front only*, *no centerline*, and *frame stack*; see Appendix A.2.1 for technical details.

Experimentally, we found little difference during training, see Figure 3. These results are somewhat surprising for us. In the *front only* experiments, we remove the rear part of the observation. Contrary to our initial intuitions, it barely affects the training. One important piece of information missing in the bird's-eye view experiments is that of (relative) speed of other vehicles. A standard way to mitigate it is to use framestack (stacking a few consecutive observations, 4 in our case), which allows us to infer the speed. As expected, the training performance is better, although be a very small margin.

It is important to compare to evaluations on the test scenarios. Here the picture is very different and depends on the set of scenarios. On the one hand, on openDD all ablations present worse performance compared to the standard bird's-eye view setup. While underperforming of the *front only* experiments

[1] In this work we fix the low-quality CARLA settings, set the standard weather, and choose the ego vehicle to be Audi A2.

is expected, the seemingly redundant centerlines and much worse behavior of framestack policies might be surprising. The latter can be possibly explained by overfitting to bigger input and is subject to further studies. On the other hand, in NGSIM scenarios the performance of *framestack* is slightly better than baseline – this might validate the assumption that the network has learned to infer other traffic agent speed from the stacked frames, which is a useful information when performing lane change maneuver. More details about bird's-eye view experiments can be found in Appendix A.4.1).

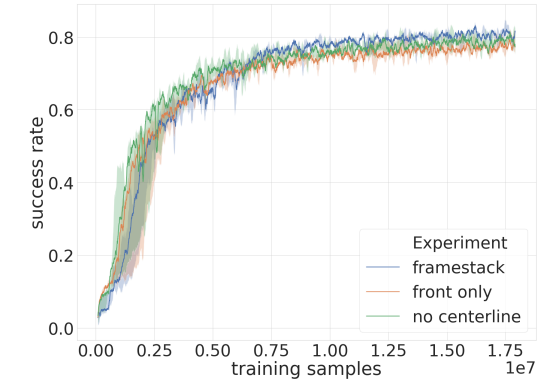


Figure 3: Above: training curve for different variants of bird's-eye view comparisons on NGSIM. In each case three experiments were run.
Below: the same situation from openDD in three views. From left to right: base, front only, no centerline.

Custom/semantic saliency map Using neural networks for policies has the disadvantage of being relatively hard to interpret and debug. In order to alleviate this issue, we developed a new simple method of generating *semantic saliency maps*, which measure the contribution of other traffic participants towards decisions taken by the policy. This is achieved by erasing a participant (a car) from the input, passing the modified observation through the network, and comparing how much the output (steering and speed) has changed. In Figure 4 you can see an example of a situation faced by the ego vehicle (green), together with its semantic saliency map. As expected, it mostly pays attention to the nearest cars.

4.2 Other modalities

Comparison of various observation modes As mentioned before, using a simulator makes it easy to test the performance of algorithms with different possible inputs. Apart from the bird's-eye view input, we examined visual input (consisting of 4 RGB cameras), and a lidar-based setup (consisting of lidar and one front-facing camera). The observation spaces for both experiments are detailed in Section 2.1.

During training, all three modalities showed similar performance (see Appendix A.4.2). However, as evident from Table 2, there is a difference in performance on the validation set and it depends on the scenarios set. The bird's-eye view policy was better on NGSIM, while visual input outperformed it on openDD. In both scenarios set, lidar plus camera was the weakest. The reason for these differences is not not yet well understood and would be a useful direction for further research.

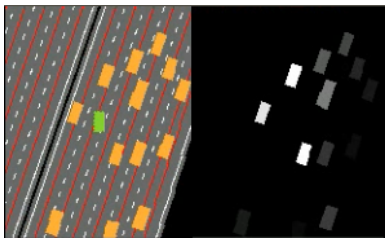


Figure 4: Bird's-eye view and saliency map

We run one additional ablation experiment: in the *front camera* experiments we use only one camera. Interestingly, in most cases, that is enough to successfully perform a lane change maneuver, because cars on the highway usually maintain a certain distance and the learned policy will try to exploit this by changing the lane as fast as possible when there are no other cars in front.

4.3 Reward experiments

In all the above experiments, we used the *dense reward* scheme. In this section we show comparison to two other schemes: *sparse* and *no failure penalty*. In the sparse scheme, the agent gets $+1$ for the successful completion of a scenario maneuver and -1 when it fails. In the dense scheme, additional rewards are given for partial completion of a maneuver. In our case, we divide every scenario into

10 parts, each part worth 0.1. The *no failure* scheme is the same as dense except that the -1 failure penalty is not issued. The details of the rewards are described in Appendix A.1.2.

The sparse rewards are perhaps the most natural, as they are close to the success rate metrics. We add the -1 penalty to make the learning of failure cases faster. Making rewards denser is often used in practice (sometimes referred to as "reward shaping"). The rationale is to make learning easier by decreasing the effective planning horizon. The major drawback is that such "shaping" is only a proxy of the desired objective and may lead training into an unintended direction (often unexpected to the experimenter).

	Rewards experiments		
	dense	sparse	no failure penalty
ngsim	0.828	0.741	0.707
opendd	0.914	0.829	0.757

Table 3: Success rates on test scenarios for CRTS scenarios with bird's-eye view input.

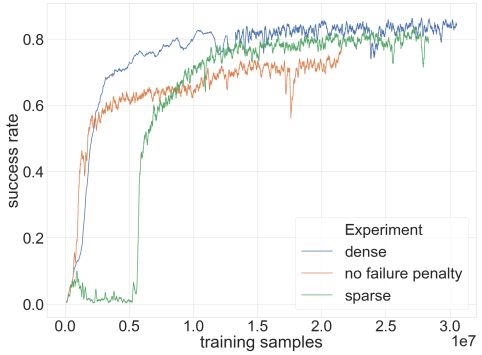


Figure 5: Reward schemes comparison for NGSIM scenarios with bird's-eye view input.

In Figure 5 we report *success rate* on the training set. We can see that the dense rewards perform best; the sparse starts training considerably later, but after that, quickly catches up reaching slightly lower level. Perhaps surprisingly, removing the failure penalty significantly hinders the later phases of training. The results on the test set, see Table 3, "stretch" the above differences, making the dense reward setting a clear winner. More details are contained in Appendix A.4.3.

4.4 Qualitative analysis of driving styles

There are many interesting questions concerning obtained driving polices. Perhaps the most compelling one is asking if they are "natural". In an attempt to answer it we developed a tool for comparing the ego vehicle behavior with the original drive from the data-set; see Figure 6 and videos on project website: `https://sites.google.com/view/carla-real-traffic-scenarios/`. We observed that, in general, our policies drive much faster than the original ones. This aggressive driving style might be due to a lack of rewards for comfort and traffic rule following (we also observed a couple of situations driving of on the left lane). The quality of driving looks correct, although a few cases of avoidable crashes were observed. In the front-view experiments, there are rare instances of crashes with an unobserved rear vehicle. In Appendix A.6 we present further analysis, including statistics regarding performance on particular maps and cross-analysis of pair of policies.

5 Conclusions and further work

Our work introduces CRTS – a new training and evaluation ground for autonomous driving based on real-world data. Our benchmark follows good practices of providing a train/test split. We hope that CRTS will serve as a platform for developing new methods and a sanity check before deployment in the real world. We also provide benchmark evaluations using reinforcement learning and observe that there is still much room for improvement. We compare various observation settings and asses generalization and realism of obtained behaviors.

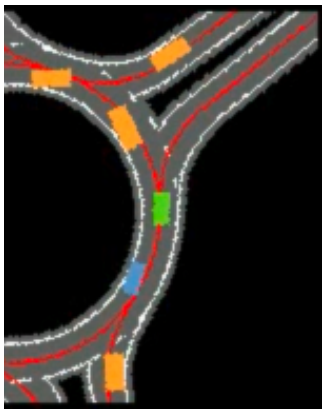


Figure 6: Ego-vehicle position (green) and original car (blue).

There are many research directions to pursue further. Perhaps the most apparent is adding more scenarios in number and type (e.g. Krajewski et al.). In this work we focused on the maneuver success rate. This can be extended to serve more specific purposes, e.g. for measuring comfort. Generalization in other domains, like changing weather settings and vehicle models can also be tested. We saw that a sparse reward setting yields considerably worse results. It could be an interesting reinforcement learning problem to achieve better performance, perhaps using an

algorithm like SAC that would provide better exploration and sample efficiency. Another direction is experiments with neural architecture – perhaps using LSTMs would bring better results. Our scenarios could be used with other learning methods (e.g. imitation learning or steering via waypoints) or more classical planning methods.

Another research avenue is casting the problem into a multi-agent setup by making other driving actors controllable. An important new challenge, in this case, will be maintaining behaviors resembling real-world ones.

References

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, et al. Solving rubik's cube with a robot hand. *arXiv:1910.07113*, 2019.

Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst. *arXiv:1812.03079*, 2018.

D. Bevly, X. Cao, M. Gordon, G. Ozbilgin, D. Kari, B. Nelson, J. Woodruff, M. Barth, C. Murray, A. Kurt, K. Redmill, and Ümit Özgüner. Lane change and merge maneuvers for connected and automated vehicles: A survey. *IEEE Transactions on Intelligent Vehicles*, 2016.

Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. *arXiv:1812.03823*.

Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. The inD dataset: A drone dataset of naturalistic road user trajectories at german intersections. *arXiv:1911.07602*, 2019.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016.

Antonia Breuer, Sven Elflein, Tim Joseph, Jan-Aike Bolte, Silviu Homoceanu, and Tim Fingscheidt. Analysis of the effect of various input representations for LSTM-based trajectory prediction. In *ITSC 2019*, 2019.

Antonia Breuer, Jan-Aike Termöhlen, Silviu Homoceanu, and Tim Fingscheidt. openDD: A large-scale roundabout drone dataset. *arXiv:2007.08463*, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016.

Chen Chen, Jun Qian, Hengshuai Yao, Jun Luo, Hongbo Zhang, and Wulong Liu. Towards comprehensive maneuver decisions for lane change using reinforcement learning. 2018.

Chenyi Chen, Ari Seff, Alain L. Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV 2015*. IEEE Computer Society, 2015.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *CoRL 2019*. PMLR.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, 2019.

Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA 2018*. IEEE.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines. `https://github.com/openai/baselines`, 2017.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CoRL 2017*. PMLR.

Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. 2020. URL `https://www.a2d2.audi`.

John Halkias and James Colyar. NGSIM interstate 80 freeway dataset, 2006.

Mikael Henaff, Alfredo Canziani, and Yann LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. In *ICLR*, 2019.

John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset, 2020.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *arXiv:2002.00444*, 2020a.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *arXiv:2002.00444*, 2020b.

Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highD dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *ITSC 2018*.

Błażej Osiński, Adam Jakubowski, Piotr Miłoś, Paweł Zięcina, Christopher Galias, and Henryk Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. *arXiv:1911.12905*, 2019.

Dean Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan Kaufmann, 1988.

Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In *ECCV 2016*.

Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. *arXiv:1611.04201*, 2016.

Dorsa Sadigh, Shankar Sastry, Sanjit A. Seshia, and Anca D. Dragan. Planning for autonomous cars that leverage effects on human actions. In *Proceedings of Robotics: Science and Systems*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

Shai Shalev-Shwartz, Nir Ben-Zrihem, Aviad Cohen, and Amnon Shashua. Long-term planning by short-term prediction. *CoRR*, abs/1602.01580, 2016a. URL `http://arxiv.org/abs/1602.01580`.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *CoRR*, abs/1610.03295, 2016b. URL `http://arxiv.org/abs/1610.03295`.

Ardi Tampuu, Maksym Semikin, Naveed Muhammad, Dmytro Fishman, and Tambet Matiisen. A survey of end-to-end driving: Architectures and training methods. *arXiv:2003.06404*, 2020.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proc. Rob: Sci and Sys*, 2018.

S. Ullman. Against direct perception. *Behavioral and Brain Sciences*, 3(3):373–381, 1980.

U.S. Department of Transportation Federal Highway Administration. Next generation simulation (NGSIM) vehicle trajectories and supporting data, 2016.

P. Wang, C. Chan, and A. de La Fortelle. A reinforcement learning based approach for automated lane change maneuvers. In *IV 2018*. IEEE.

Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. 2000. URL `http://torcs.sourceforge.net`.

Dongfang Yang, Linhui Li, Keith A. Redmill, and Ümit Özgüner. Top-view trajectories: A pedestrian dataset of vehicle-crowd interaction from controlled experiments and crowded campus. In *IV 2019*. IEEE.

Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 2020.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv:1806.07937*, 2018.

A.1 Benchmark scenarios

A.1.1 Scenario extraction algorithm

In this section we describe the procedure of creating scenarios. In both the cases of NGSIM U.S. Department of Transportation Federal Highway Administration [2016] and openDD Breuer et al. [2020], the datasets are scanned for maneuver events – lane change and entering roundabouts, respectively. The car performing the maneuver is declared to be the ego vehicle and a scenario is formed by mapping all traffic participants except for the ego vehicle to CARLA. A vehicle appearing in the field of view for the first time is spawned within the location and velocity matching the dataset. The model of vehicle is chosen from the CARLA library to match the dimension of the replayed car (measured using the Jaccard similarity). Later, consistency with the dataset is enforced every 100ms (by setting again the locations and velocities).

The initial speed and position of the ego vehicle is determined in the same way and in subsequent frames it is maintained by the CARLA physics engine according to the received actions.

The set of such created scenarios is divided randomly into the train and test set in the 80/20 ratio.

NGSIM The lane change event is declared as changing the lane id corresponding to a given vehicle. The scenarios starts 5 sec before the lane change event. The scenario is considered successful if, at least for 1 sec, the ego vehicle is located less than 30cm of the target lane and its yaw is smaller than 10 degrees. The scenario is unsuccessful if either a collision occurs, the car leaves both the starting and target lane or there is a timeout of 10 sec.

During the scenario the chauffeur commands are issued: `LANE_CHANGE_(`dir`)` if the ego vehicle is on the starting lane, $dir \in \{left, right\}$ denoted the direction to the target lane, otherwise `LANE_FOLLOW` is issued.

For the NGSIM we remap position of the rear and front axle of vehicle (to match the CARLA convention). We also smooth positions using a 1.5 sec window.

openDD The scenarios begin when the ego vehicle is approx. 20 meters before the roundabout entry. The scenario is considered successful if the ego vehicle exits the roundabout via the same exit as the reference driver. The scenario is unsuccessful if either a collision occurs, the car moves away more than 3m from the original trajectory or there is a timeout of 1.5 times of the time of the original drive.

During the whole scenario `LANE_FOLLOW` command is issued until the car passes the last exit before the target one. Then the command changes to `TURN_RIGHT`.

The openDD dataset is recored at 30 fps, which we downsample to match our 10 fps.

ALC Artificial Lane Change (ALC) is a set of synthetic procedurally generated scenarios similar to NGSIM. On one lane of a highway we spawn a column of vehicles. Their position is randomized and their average distance is 8m. All the vehicles in the column move in the the same speed, which is randomly chosen at the beginning of the scenario from the interval $[3, 5.5]$ m/s. The ego-vehicle is spawned on the lane to the right of the column and tasked to change lane to the left. The speed of the ego-vehicle is random in $[3, 5.5]$ m/s. Other details, success conditions, rewards etc are the same as in NGSIM.

A.1.2 Rewards

NGSIM The distance to the target lane is segmented into 10 pieces. A reward of 0.1 is issued each time the car moves a segment closer to the target lane and is penalized with -0.1 if it moves to a more distant segment.

At the end of the scenario a reward of 1 (resp. -1) is issued if the scenario is successful (resp. unsuccessful).

openDD The original trajectory is segmented into 10 pieces. The agent is rewarded with 0.1 for passing each segment. At the end of the scenario a reward of 1 (resp. -1) is issued if the scenario is successful (resp. unsuccessful).

Sparse reward In the above setting ± 0.1 "dense" rewards are issued for partial progress. In the sparse regime these are omitted.

No failure penalty We do not issue -1 for unsuccessful scenarios.

A.2 Training details

We report the results for one seed, though more will be available before publication. We note, however, the training has been stable in the cases we've observed so far.

A.2.1 Observations for bird's-eye view

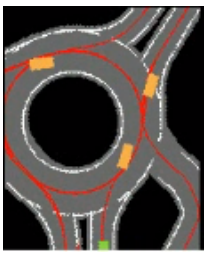


Figure 7: Bird's-eye front view

The observation in all *bird's-eye view* baseline experiments is a tensor of shape $(186, 150, 5)$, which corresponds to a real-world area of approximately $47m \times 38m$. The ego vehicle is located in the center; see Figure 4. There are five channels, each representing a distinct category of CARLA world features: roads (grey area on Figure 4), lanes (solid and dashed white lines), centerlines (red lines), other vehicles (orange), and the ego vehicle (green). In *front only* experiments we use $(186, 150, 5)$ tensors covering only area in front of the ego vehicle, see Figure 7. In *no centerline* experiments the input is $(186, 150, 4)$ as we omit the centerline channel. In *framestack* experiments we stack 4 consecutive observations into $(186, 150, 20)$ tensors.

A.2.2 Observations for RGB cameras

We use 4 RBG cameras with field of view equal to 90 degrees and a resolution of 384 (width) and 160 (height), totaling in $(160, 384, 12)$ tensor inputs. The cameras are facing front, right, left and rear directions, more precisely 0, 90, 180, 270 degrees with respect to the longitudinal axis of the vehicle. Their roll, pitch and yaw are set to 0.

In *front camera* experiments we use only the front-facing camera.

A.2.3 Training algorithm and infrastructure

For training we use the PPO algorithm Schulman et al. [2017], which produced state-of-the-art results for many challenging problems. We base our code on the ppo2 OpenAI Baselines implementation Dhariwal et al. [2017]. Our typical experimental setup consisted of 4 machines with 2 Nvidia K40 GPUs each. One GPU was used for policy training, the other hosted 56 instances[2] of CARLA 0.9.9.4 for experience collection. The nodes were synchronized using MPI. This setup was able to produce 1.3M hours of experience per day when using *bird's-eye view*, 390k with *visual input*, and 800k with *lidar and camera*. The CARLA server was configured to render frames with fixed time-step equal to 10, which means that in the first case, 13M simulation steps were produced. It is important to note that the typical experiment lasted 3 days.

A.2.4 Action space

The ego-vehicle is controlled by steering and throttle. The angle of the steering wheel can be set to within $[-360, 360]$ degrees, which is normalized to $[-1, 1]$. The speed is controlled indirectly by issuing target speed values to a PID controller. The PID directly steers throttle. In both cases the policy is modeled using the Gaussian distribution with the mean and the logarithm of the variance output by a neural network.

[2]The number of CARLA instances on one GPU was 8 due to the memory requirements.

A.2.5 Evaluation details

In order to measure the agent's performance on unseen situations (which come from the validation set) we developed an evaluation script.

Scenarios are sampled from the datasets with fixed seed equal to 777. Model weights are initialized by the most recent checkpoint saved during the training process. In NGSIM there are two maps (I80 and US101), while in openDD there are 7 (RDB1-RDB7). In order to get a comparable number of evaluation episodes for both datasets, we choose 30 episodes per map in NGSIM (60 in total) and 10 per map on openDD (70 in total).

Data from every simulation step is recorded and stored in a JSON file. Gathered metrics contain information about rewards, potential failure causes (collision, timeout, off-road, moved too far from reference trajectory) or successful endings, distance from replayed reference vehicle (replaced by the agent), and much more. Furthermore, we save video files with RGB & bird-eye's view preview and extensive text information.

A.2.6 Network architecture

We used a CNN backbone inspired by Espeholt et al.[3] It consisted of three residual blocks with channel number respectively 16, 32 and 32. In each policy, the same CNN backbone was applied separately to the one or more of the policy inputs (bird's-eye view, 4xRGB camera, lidar & camera). The backbone's output was concatenated with embeddings of car metrics (current speed and acceleration) and of a high-level navigational command.

A.3 Hyperparameters

Parameter	bird's-eye view experiments	visual input experiments	lidar and camera experiments
CNN backbone	IMPALA CNN		
network architecture	backbone applied to 2-D map	backbone applied separately to each camera	backbone applied separately to RGB input and lidar's 2D map
lr	0.0003		
clip range	0.2		
noptepochs	1		
nsteps	64		
nminibatches	8	16	
lam	0.95		
ent_coeff	0.001		
gamma	0.98		

Table 4: Hyperparameters used during training.

A.4 Experimental details

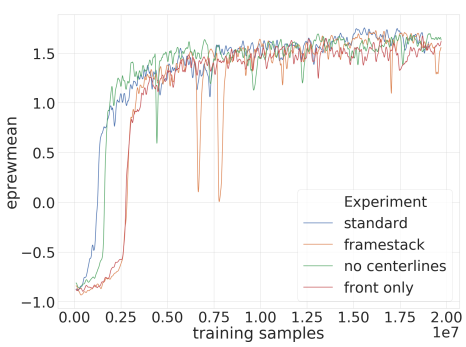
A.4.1 Bird's-eye view experiments

As mentioned in the main paper, we saw little difference during training with bird's-eye view ablations, as further evidenced by Figure 8a. Concerning evaluation (see Table 5 and Table 6), there is no large difference when not using centerlines, which is surprising. The *front only* variant performs slightly worse, which is to be expected. Furthermore, *frame stack* has significantly worse results, especially on openDD, which may have been caused by optimization issues with the larger input.

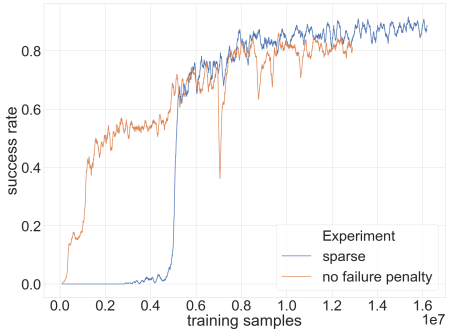
A.4.2 Other modalities experiments

In addition to bird's-eye view, we examined visual input consisting of 4 RBG cameras, as well as a lidar & one front-facing camera setup. The observation spaces for the latter two are described in

[3]Espeholt, Lasse, et al. *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.* International Conference on Machine Learning. 2018.

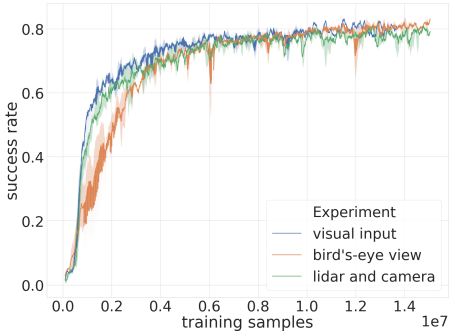


(a) Comparison of bird's-eye view experiments on openDD scenarios.

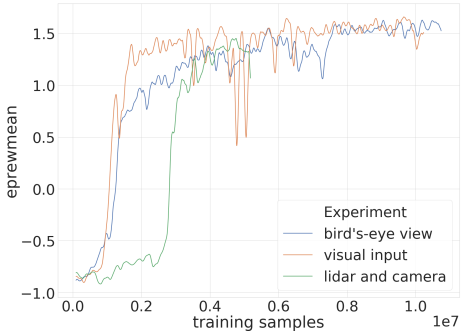


(b) Reward schemes comparison for openDD scenarios with bird's-eye view input.

Figure 8: Comparison of openDD experiments.



(a) For NGSIM scenarios. Three experiments were run in each case.



(b) For openDD scenarios.

Figure 9: Comparison of other input modalities.

Appendix A.2.1 and Appendix A.2.2, respectively. As we can see in Figure 9, all three modalities exhibited similar performance during training.[4] However, we see a difference during evaluation on NGSIM (see Table 6). Also, lidar & camera has slightly worse performance in evaluation on openDD scenarios, as detailed in Table 5, though it's hard to draw any definitive conclusions from this.

A.4.3 Rewards experiments

In these experiments we can see dense rewards performing the best during training – the sparse variants take a long time to get going and reach lower performance overall. Furthermore, removing the failure penalty hinders the training (see Figure 8b). These observations also carry on to evaluation – the dense variant performs best (see Table 5 and Table 6).

A.5 Datasets

All NGSIM and openDD maps were built using RoadRunner by Mathworks. One benefit of using it instead of building maps directly in Unreal Engine Editor is out-of-the-box support for the OpenDRIVE format, which means that roads are fully annotated (e.g, road centerlines, lane markings, junctions). That information is crucial to render the bird's-eye view of a scene without the necessity of rendering a 3D scene, which is computationally expensive. In addition to that, aerial images can be layered underneath which allows for very precise and high-fidelity map construction. Released maps

[4]Due to a mistake in running experiments we didn't report the success rate, thus we compare the mean episode reward. This will be fixed in the final version.

does not contain environmental details such as buildings, trees, or pavements in order to minimize the importance of particular objects to the agent across different maps, and at the same time, maximize simulation performance.

We base on NGSIM U.S. Department of Transportation Federal Highway Administration [2016] and openDD Breuer et al. [2020] datasets, which contain footage of real traffic and processed trajectory data; see Figure 10. We use the trajectory data to produce scenarios as described in Section A.1.1.

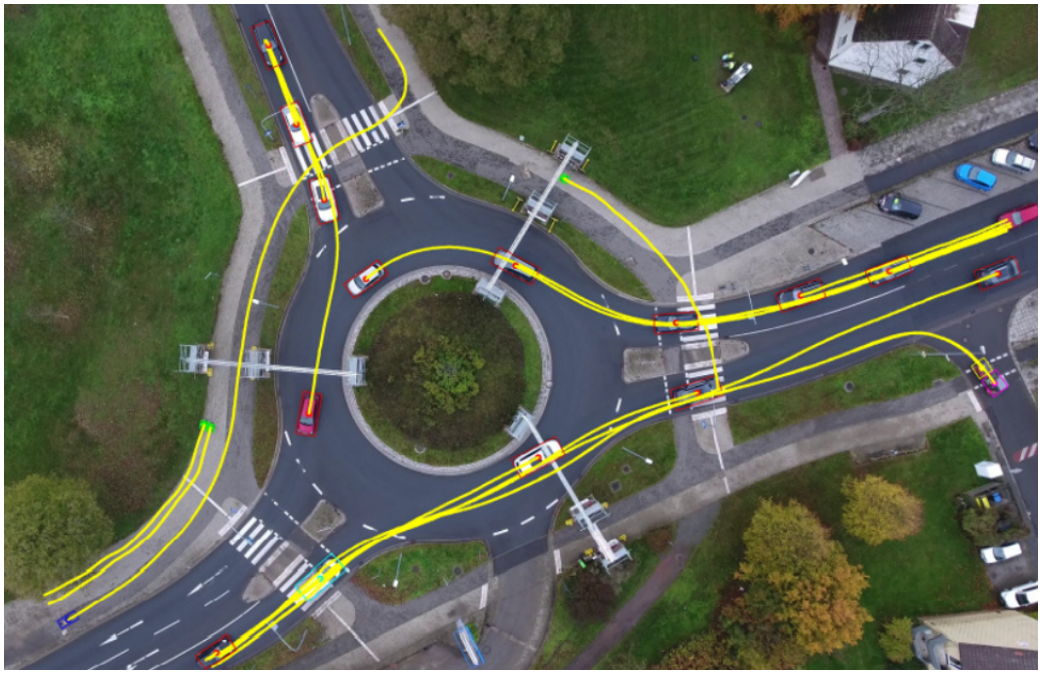


Figure 10: An example visualization from Breuer et al. [2020] of a given traffic scene included in the dataset.

A.6 Analysis of evaluation metrics

As previously mentioned, each evaluation step contains information about whether episode succeeded, table 5 and 6 address that metric. RDB5 seem to be the easiest map for all experiments, the toughest one is RDB4. The *bird's-eye view* has the highest success rate across all the maps.

	experiment	RDB1	RDB2	RDB3	RDB4	RDB5	RDB6	RDB7
0	bird's-eye	0.9	0.9	1.0	0.6	1.0	1.0	1.0
1	visual input	0.7	1.0	1.0	0.6	1.0	0.9	1.0
2	lidar and camera	0.6	0.5	0.7	0.3	1.0	0.7	0.8
3	front only	0.5	0.9	1.0	0.6	1.0	1.0	1.0
4	no centerline	0.8	0.8	0.9	0.6	1.0	1.0	1.0
5	framestack	0.3	0.8	0.7	0.4	1.0	0.8	0.7
6	sparse	0.7	0.9	1.0	0.5	1.0	0.9	0.8
7	no failure penalty	0.4	1.0	0.9	0.4	1.0	0.8	0.8

Table 5: Mean success rate of evaluation episodes runs for openDD maps.

According to Figure 11, *bird's-eye view* performs best for both datasets, having a small advantage over *no centerline* in the number of collisions. With *no centerline* agent tends to move away from the road center more often, which is expected.

From all input types on NGSIM, *front only* has the highest collision rate, but considering the importance of rear vision in a lane change maneuver, the result is not surprising. On openDD *front*

	experiment	I80	US101
0	bird's-eye	0.862069	0.793103
1	visual input	0.827586	0.724138
2	lidar and camera	0.862069	0.689655
3	front only	0.827586	0.724138
4	no centerline	0.862069	0.724138
5	framestack	0.758621	0.793103
6	front camera	0.827586	0.758621
7	sparse	0.827586	0.655172
8	no failure penalty	0.793103	0.620690

Table 6: Mean success rate of evaluation episodes runs for NGSIM maps.

only performs better by this criteria, which might suggest that side or rear vision is less crucial on roundabouts.

A unusual result is observed for *lidar and camera* input on openDD. Seemingly agent has difficulties with lane keeping, probably due to the fact that all roundabout maps are flat - road cannot be distinguished from grass with laser and one front-facing camera does not provide enough view angle to perceive all lane markings. On NGSIM this effect is not observed, because there, the agent is surrounded by other vehicles and barriers, which are perceived by lidar. *Framestack* also struggles on roundabouts, although agent's actions are more smooth, they are also less reactive. This leads to going off-track on such a narrow roads much more likely.

For both datasets, the highest collision rate is reported for experiments trained with *no failure penalty*. This is not surprising as without the negative feedback it tends to risk more often.

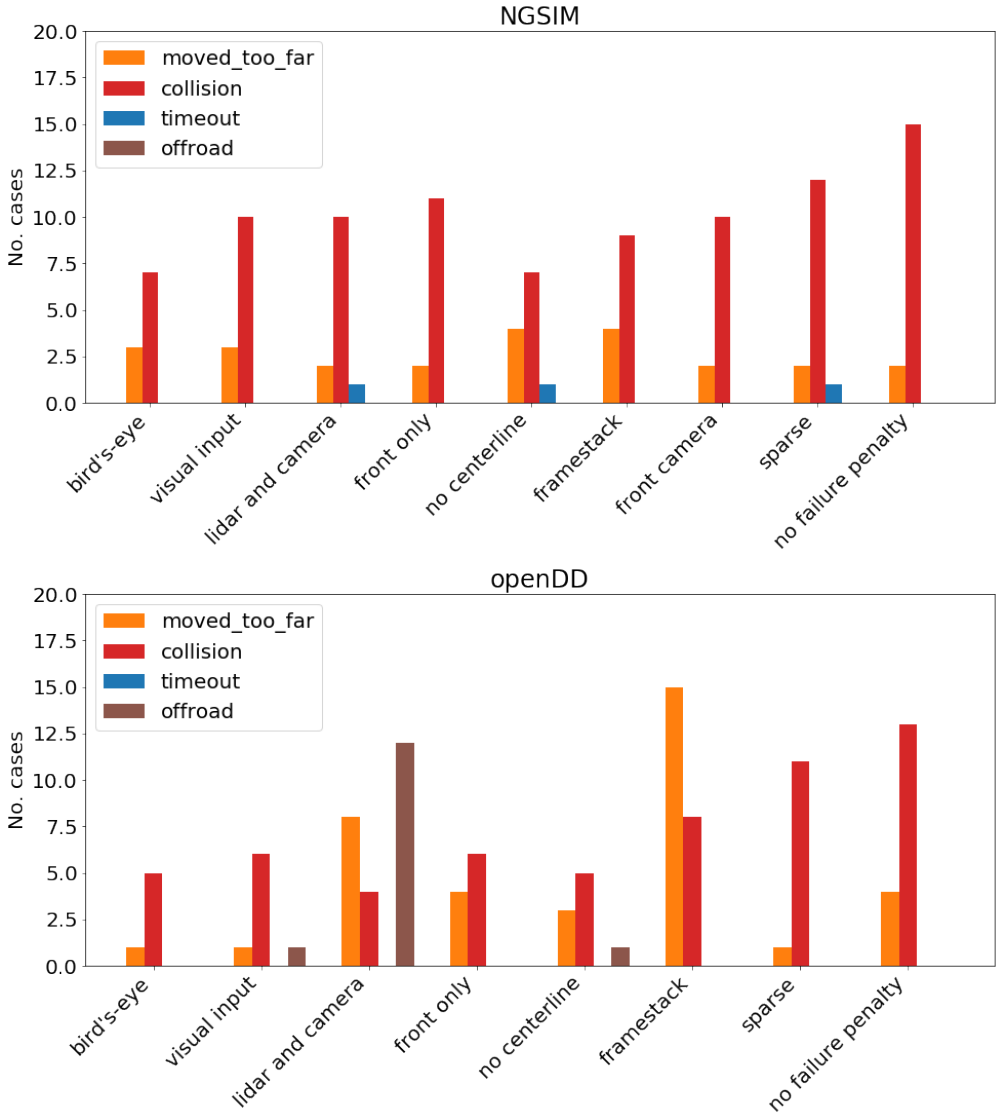


Figure 11: Failure causes during evaluation on NGSIM and openDD validation sets depending on input type.