3DCV - Critical Driving Scenarios

Generate critical driving scenarios in CARLA simulator and apply imitation learning to train a neural network

Christopher Klammt Tobias Richstein Julian Seibel Karl Thyssen

***Abstract*—In this project report, we present an approach for generating new scenarios for supporting autonomous driving tasks in critical situations using the CARLA simulator along with it's Scenario-Runner. Our approach consists of generating new XML-based OPENScenario files which can be interpreted by the Scenario-Runner. Based on already present basic Scenarios, we apply a variety of changes to attributes which are crucial for different sensors of an autonomous car. We considered external factors like weather, road conditions, time of day, color and models of cars, cyclists and pedestrians. In a second step, we transform the scenarios to different places in the same town and to different towns. To ensure the novel generated scenarios are valid, we consider different sanity checks as well as a specific verification to avoid duplicating scenarios. Also, we use the generated scenarios for training a model using imitation learning. TODO..**

***Index Terms*—CARLA simulator, Scenario generation, Critical driving scenarios, Imitation learning, Autonomous driving**

I. Introduction

Autonomous driving vehicles are not just an idea for the more distant future, but rather a very current topic. Not only Tesla, the company that dominates the news in this regard, is showing how far autonomous driving has come in the last years. One crucial issue still lies in providing a stable and safe behavior in critical situations, especially in which traffic participants behave unexpectedly. The problem is that to robustly learn the appropriate behavior for these specific critical scenarios and to generalize using supervised learning a large amount of training data is needed.

In order to tackle this problem, our paper describes the development of a generator for such critical driving scenarios in the CARLA simulator. These critical driving scenarios are mainly inspired by the report *Pre-Crash Scenario Typology for Crash Avoidance Research* [1] for the National Highway Traffic Safety Administration in the United States. These contain scenarios such as avoiding an obstacle, lane changing with oncoming traffic or crossing traffic running a red light at an intersection (as shown in Figure 1).

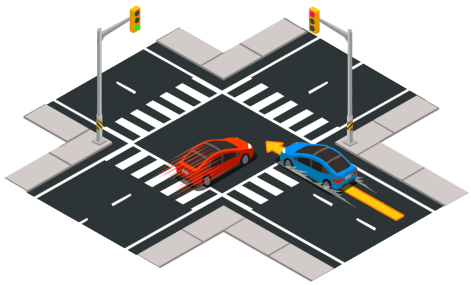

Fig. 1: NHTSA scenario: Crossing traffic running a red light at an intersection [2]

Furthermore, after generating these different critical driving scenarios we utilize them to learn a robust model. To do so imitation learning is used, in which an expert demonstrates the desired behavior in each critical situation and is then adopted by the model.

II. Related Work

A. CARLA

As presented by Dosovitskiy et al., CARLA is an open-source urban driving simulator for autonomous driving research [3]. It enables handling different use cases within the general problem of driving, such as learning driving policies or training perception algorithms. To control the simulation, e.g. changing weather, adding cars or pedestrians, an API is available in Python and C++. CARLA consists of the simulator responsible for rendering etc. as well as multiple components, for example a traffic manager controlling the vehicles or a component handling the sensors.

B. Critical driving scenarios

Another component that works in unison with the CARLA simulator is the ScenarioRunner [4]. This is a module that makes it possible to define various traffic scenarios and execute them in the CARLA simulator.

These scenarios can be defined using Python or the OpenSCENARIO [5] standard. The ScenarioRunner already contains some predefined Scenarios which are based on the critical driving scenarios as described in *Pre-Crash Scenario Typology for Crash Avoidance Research* [1].

C. Generating data in simulators

In order to generate data it first is necessary to identify parameters that are to be adjusted and that should vary across the different entities. The data generation can then be approached in different ways. One possibility is to use a random statistical distribution of these parameters.

Another way to go about data generation is to use learning-based methods to adjust the parameters of the simulator. Such an approach was chosen by Ruiz, Schulter, and Chandraker and published in "Learning To Simulate" [6]. They proposed a reinforcement learning-based method to adjust the parameters of the synthesized data to maximize the accuracy of a model trained on that data. A quite similar approach was chosen by Kar et al. [7] called Meta-Sim, which learns a generative model of synthetic scenes and modifies the attributes using a neural network.

D. Imitation learning

To learn a model based on labeled data some form of supervised learning is generally applied. Chen et al. propose a method called "Learning by Cheating" [8]. This is a two-stage method, in which first an agent is trained using privileged information. In the second stage, the privileged agent acts as a teacher that trains a purely vision-based agent.

Another approach is "End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances" [9]. Here reinforcement learning is utilized to learn an optimal behavior policy based on a reward-function, effectively punishing wrong behavior such as leaving the track or running over pedestrians.

To train and manage the trainings of imitation learning networks jointly with evaluations on the CARLA simulator *COiLTRAiNE: Conditional Imitation Learning Training Framework* [10] can be used.

III. Method

A. Generating new scenarios through ScenarioRunner

In general, we identified the two possible ways to create new scenarios that are supported by the ScenarioRunner. The first option consists of using the Python based API of the ScenarioRunner library to define the dynamic behavior of every actor in combination with a `xml`-based configuration file, which can be used to initialize different parameters for a predefined python based scenario.

The second option is to utilize OpenSCENARIO-based configuration files supported by the ScenarioRunner. OpenScenario is standard `xml`-based file format for the description of dynamic contents in driving simulation applications [5]. These configuration files differ from the former approach mainly in one aspect. For defining OpenSCENARIO based scenarios, there is no need to create the python-based scenario description. Therefore OpenSCENARIO can be used to generate all configurations for executing a scenario through the ScenarioRunner library at once.

Both the introduced options come with advantages and disadvantages. For our decision process, we identified three criteria which we would expect for the approach we want to work with. The first criteria is *generalization*. We would expect that the approach can be well generalized by extracting an abstract process which can be applied to all the different formats a new scenarios could possibly have. The second criteria is *the number of directly accessible and adjustable parameters*. Since we aim to provide a scenario generation process which is capable of generating as many scenarios as possible, we want to achieve a high number of adjustable parameters for providing a large combinatoric basis. The third and last criteria we considered is *simplicity* of the overall concept. Reducing the complexity and following the KISS (Keep it simple, stupid)-Principle for achieving different advantages in further developments like extensions and maintenance.

Finally after evaluating both the options for generating new scenarios with respect to the defined criteria, we concluded to use the OpenSCENARIO-based approach. Overall we saw only advantages by using the openSCENARIO standard as our final way for generating new scenarios. In comparison to the Python-based scenario generation, openSCENARIO can be better generalized in terms of generating the files itself. It is pretty uncommon and connected with more effort to generate Python-files for setting up scenarios. In addition, the `xml` configuration needs to be generated as well. OpenSCENARIO-based files are including all the necessary data combined in one file, which is additionally `xml`-based. Therefore it provides an easy file-generation while maintaining all the information for one scenario. Considering the second criteria of direct accessible and adjustable parameters, with

OpenSCENARIO-files, all possible parameters which are crucial for a scenario are direct accessible and values can be changed easily. All the described differences can be transferred to the last and third criteria of following the KISS-principle. It might be that a OpenSCENARIO file alone has a more complex structure, but nevertheless, it would be much more complicated to generate generic Python-files for defining new Scenarios.

B. Design and Implementation

With the decision of using OpenSCENARIO-files, we designed a generator tool which is capable of changing a variety of parameters that are decisive for the underlying problem of critical driving scenarios. Our approach utilizes the already present basic scenarios for critical driving situations of the ScenarioRunner. Since it is of tremendous effort to generate scenarios which state a completely new critical situation by changing the overall behavior of all actors, we first decided to use the five predefined scenarios, which will be called *basic scenarios* in the further course of this report. Those include situations of a crossing bicycle, changing weather conditions, following a leading vehicle, pedestrian crossing as well as a lane changing scenario.

We consider a scenario as new, if a single parameter value differs from all of the already generated scenarios. With this definition we can generate thousands of new scenarios by providing one base scenario.

To generate new values for selected parameters, we use the schema file for openSCENARIO-files of CARLA which comes along with the ScenarioRunner tool. We extract mainly attribute names of tags we considered changeable in the context of this project as well as their data-types and valid values if the data-type is categorical.

Therefore we extract available information for example of the `<weather>`-tag, which has an attribute `cloudState` that only accepts values of a predefined enum type. Our generator would extract all possible values which is in the mentioned case a set of [”`free`”, ”`cloudy`”, ”`overcast`”, ”`rainy`”, ”`skyOff`”]. With the information of all attributes and their data-types and the basic scenario as template, new scenarios are generated. We also extract the information about the map, the scenario plays in. This is a prerequisite for extracting values of pedestrian and vehicle types from the CARLA Python API, since the valid and possible values of those attributes depend on the actual map. For all the tags which include attributes of categorical values we apply a random choice on all possible values. In the case of numeric attributes, we change the values based on the set values in a range of [-100%, +100%]. For every base scenario which is accessible, the user can specify how many additional scenarios should be created. After every new scenario, we use a hash value based comparison on the document to execute a duplication check such that we avoid the generation of the exact the same scenario twice. The overall generator-flow is shown in figure 2.

TBD: Attribute tree

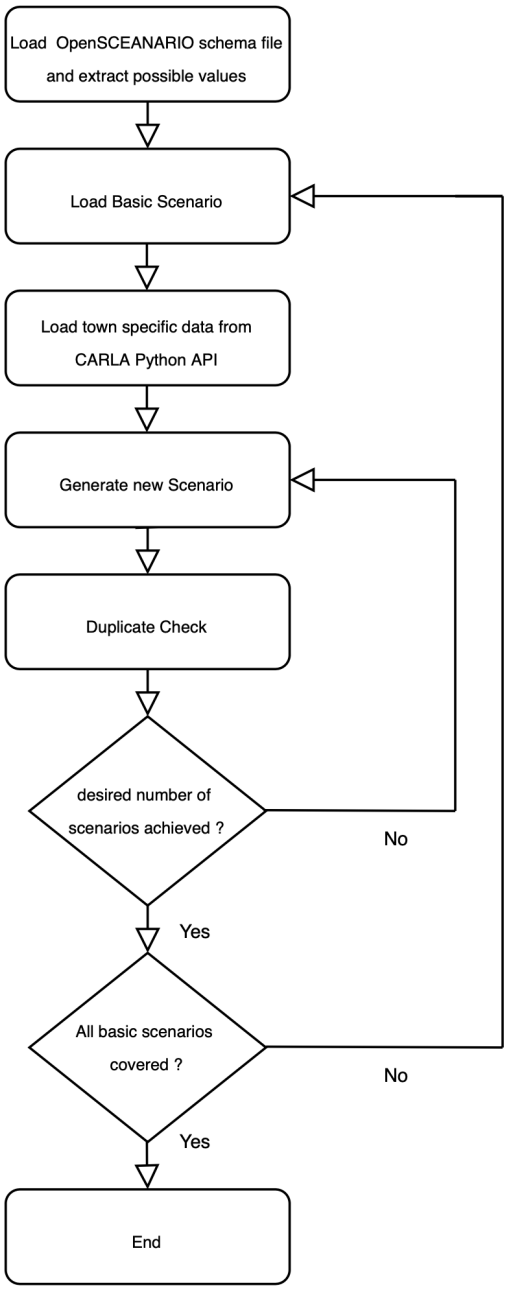

Fig. 2: Flow-Chart of the scenario-generator.

C. Adjustable parameters

CARLA and the ScenarioRunner enable us to adjust quite a few parameters, so that the generator can create a lot of diverse settings. In theory we have access to all possible parameters by the ScenarioRunner tool which are contributing to a single scenario. Since most of them are interventions into the semantics of a scenario, we propose to first change parameters which are not changing the overall actor behavior. This includes the type and model of the car, which is steered by the learned agent, but more importantly of each and every other car. This is important so that the model generalizes well, even if cars of different size, shape and color are in it's field of vision resulting in different RGB patterns perceived by the cars sensors. Another parameter, that can be changed is the speed of the cars, which also has an impact on the behavior in critical situations. This setting is not limited to cars, also available bicycle and pedestrian types are used in similar manner.

Moreover the weather plays an important role, as it drastically changes what is perceived by sensors and also alters the effects of a certain behavior. For example precipitation does not only impair the video of the camera, but additionally lengthens the breaking distance. These variables can all be adjusted rather easily, by just randomly picking a value from the set of possible values. Similar, we are feasible of adjusting all available attributes for sun and fog tags.

In addition, we generate a random time of day for every new scenario to cover also an increasing danger in situations for example in night, where pedestrians and bicycles are harder to detect.

To also simulate a variety of different road conditions, we include a random generated friction scale factor for every new scenario. Those factors can be influenced by many external effects like oil spilled on the road.

However, these attributes are independent from the original spawn-position of all actors. As an additional requirement, the generation of new scenarios should include a relocation of the scene to another suitable place in the same map or even in a different map. For this, we first analyze the initial situation by fetching the closest town specific waypoints based on the initial actor position. To get information about the scenario circumstances we use a simple approach to detect whether the scenario takes place on a intersection by calculating and comparing the distances between the actors and the hero (ego-vehicle) actor to the next intersection in driving direction. If the latter distance is less, we conclude that the scenario must take place at an intersection. This information is valuable for shifting the scenario to another position in the same or different town. We do this with respect to all the conditions given by the initial scenario for instance retaining distances between all actors.

If we conclude that the scenario takes place at an intersection, we fetch all possible junctions using the Python API of CARLA for a specific map. The closest junction in driving direction is used as the scenario anchor.

For some scenarios, it is typical that actor positions like for pedestrians or cyclists are not considered as waypoints since they spawn off-road. If we detect such a deviation we calculate a offset between the actual actor position and the nearest waypoint. To apply the correct orientation we rotate the position of the actor with respect to the new orientation of the road.

$$M = \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

tbd. picture of bike scenario + mit distancen

Not all positions calculated will work on different maps. Therefore we include two sanity checks when calculating new actor positions.

The part of the configuration, which is more difficult to vary is with regards to further actors, such as pedestrians and other vehicles. These have to be spawned somewhere and a behavior needs to be defined. CARLA provides a list of possible spawn points across a specific map, but if we just choose these randomly, we cannot be sure, that the spawned car has an impact on our scenario. Most importantly, the behavior of the car, which characterizes the critical scenario needs to well defined. But apart from that, we also want vehicles in the background, that drive around somewhat automated.

D. Limitations

Kein Einfluss auf Autopilot route, nicht deterministisch. Pro intersection gibt es Wahrscheinlichkeiten die zum Moment nicht beeinflussbar sind. Eigentliche Route kann vom Agent geplant werden.

+ problems faced:

- Keine semantische informationen bei scenarios
- nur 5 basis scenarien, um komplett neue situationen zu definieren, müssten neue basis scenarien generiert werden.
- ...

E. Imitation Learning Approach

tbd,.

IV. Experiments

tbd.

V. Conclusion

To conclude this report, we try to answer three research questions based on the outcome of our experiments.

• Can we generate critical scenarios such that a model trained on these scenarios achieves robustness against critical scenarios?

• How many critical scenarios have to be represented in the training distribution in order to achieve robustness against critical scenarios?

• Does robustness against a certain type of critical scenario transfer to a different type of critical scenario?

References

[1] W. G. Najm, J. D. Smith, and M. Yanagisawa. *Pre-Crash Scenario Typology for Crash Avoidance Research*. Tech. rep. U.S. Department of Transportation, 2007.

[2] CARLA. *NHTSA-inspired pre-crash scenarios*. URL: https://carlachallenge.org/challenge/nhtsa/.

[3] A. Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[4] CARLA. *Scenario Runner*. URL: https://carla-scenariorunner.readthedocs.io/en/latest/.

[5] V. S. GmbH. *OpenSCENARIO*. URL: http://www.openscenario.org/.

[6] N. Ruiz, S. Schulter, and M. Chandraker. "Learning To Simulate". In: *CoRR* abs/1810.02513 (2018). arXiv: 1810.02513. URL: http://arxiv.org/abs/1810.02513.

[7] A. Kar et al. "Meta-Sim: Learning to Generate Synthetic Datasets". In: *CoRR* abs/1904.11621 (2019). arXiv: 1904.11621. URL: http://arxiv.org/abs/1904.11621.

[8] D. Chen et al. "Learning by Cheating". In: *Conference on Robot Learning (CoRL)*. 2019.

[9] M. Toromanoff, E. Wirbel, and F. Moutarde. "End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 06/2020.

[10] F. Codevilla. *COiLTRAiNE: Conditional Imitation Learning Training Framework*. URL: https://github.com/felipecode/coiltraine.