i4Driving Software for automatically increasing the criticality of driving scenarios

Deliverable No. D3.3 | October 2023

i4Driving
integrated 4D driver modelling under uncertainty

D3.3 – Software for automatically increasing the criticality of scenarios

Project Acronym	Grant Agreement #	Project Title	Deliverable Reference #	Deliverable Title
i4Driving	101076165	Integrated 4D Driver Modelling under Uncertainty	D3.3	Software for automatically increasing the criticality of scenarios

AUTHORS

Tobias Mascetta	TUM – Cyber-Physical Systems
Florian Finkeldei	TUM – Cyber-Physical Systems
Matthias Althoff	TUM – Cyber-Physical Systems

DISSEMINATION LEVEL

X	P	PUBLIC
	C	CONFIDENTIAL

Funded by the European Union

This project has received funding from the European Union's Horizon Europe programme, under grant agreement No 101076165.

Disclaimer

Version History

Revision	Date	Authors	Organisaton	Description
V0.1	29.09.2023	Florian Finkeldei, Tobias Mascetta and Matthias Althoff	TUM	First draft
V0.2	13.10.2023	Ali Nadi Najafabadi	TUD	Review
V0.3	17.10.2023	Florian Finkeldei, Tobias Mascetta and Matthias Althoff	TUM	Second draft
V0.4	24.10.2023	Lucia Schlemmer, arnaud Burgess, Menno Menist	PAN	Review

Executive Summary

When identifying human driver models, one is especially interested in how humans react in edge-case situation with high criticality. However, these traffic situations seldom occur in real-world traffic, making it hard to use recorded datasets for this purpose. The approach implemented in this deliverable therefore aims at generating artificial traffic scenarios based on real-world situations by steering the surround vehicles in a way that continuously increases the criticality of traffic situations. So far, similar approaches have not been applicable to human driver studies as there were far away from being real-time capable. Our approach solves this by leveraging reachability analysis in combination with heuristic guiding of the attackers. The results of our approach are shown for several examples and instructions to use our open source software are provided.

Contents

List of Figures

1 Introduction

For the determination of precise human driver models, it is useful to be able to investigate human behaviour in exceptional situations in a targeted manner. However, in reality dangerous situations rarely occur and, therefore, are hard to find in recorded traffic data.

To tackle this problem, within the scope of i4Driving a method and software implementation are being developed to automatically increase the criticality of traffic situations with respect to the vehicle-under-test (VUT) by controlling the surrounding road users.

Since human subjects require delay-free driving simulator experiments (DSE) for realistic examination, existing methods are not suitable for this purpose [1]. The aim of the present innovation is to control the surrounding vehicles in near real-time in such a way that the resulting traffic situation is continuously more difficult for the VUT to solve. To avoid trivial and inadmissible solutions, the road users must comply with the traffic rules.

In the following sections, the reasoning behind the developed method is explained and subsequently instructions for the use of the associated software are given.

Please note all the code can be found at the following link:
https://bitbucket.org/data_ecosysteem/i4driving/src/master/

2 Developed Methodology

This section describes the developed methodology by first giving on overview of the concept. Thereafter, examples are presented and lastly a high-level description of the algorithms are given.

2.1 Concept Overview

The basic approach of this innovation is to control other vehicles, called 'attackers' from hereon, to increase the criticality of the scenario dynamically by shrinking the drivable area of the VUT. A predictive scenario analysis is used as the basis for the criticality enhancement and the planning of the attackers' trajectories. Both the replanning and the re-computation of the prediction are executed in periodic cycles.

The developed methodology combines **reachability analysis with numerical optimisation** to increase the criticality of situations via a predictive approach:

In a first step, the traffic rule compliant reachable sets of all vehicles are determined by applying reachability analysis. Reachability analysis determines the states a dynamic system can reach over a time horizon. The states are defined as the combinations of longitudinal and lateral position and velocity. The drivable area is the projection of the reachable set onto the position domain. During each replanning cycle, the reachable sets are computed for the entire cycle time horizon.

However, not all these reachable states comply with the traffic rules. Those states that violate the traffic rules are removed from the reachable sets by applying automata-based model checking techniques. For an introduction to this approach in a single-agent context, please refer to [2] [1].

The existing approaches can be used for a multi-agent problem by temporarily reducing it to a single agent problem with respect to the VUT. Herein, it is assumed that the attackers follow their reference path with a constant velocity for a short period of time. This assumption is valid since the multi-agent trajectories have a fast replanning-cycle. An approach for multi-agent traffic control conformity is given, e.g., in [3].

[1] TUM-Althoff submitted a paper to ICRA 2023 that describes the approach used in our methodology. The final admission result will be published in January 2024.

Once the traffic rule compliant reachable sets for all traffic participants have been computed, the trajectories for all attackers are determined. Firstly, the intersections of the VUT's drivable area with the drivable areas of the attackers are examined and suitable target states are chosen via a heuristic algorithm. Secondly, the trajectories of the attackers are planned via convex optimization of a Quadratic Program (QP).

A common approach that significantly eases planning in autonomous driving is to switch between a global cartesian coordinate frame and a vehicle-dependent curvilinear coordinate frame (also known as Frenet-Frame) created along the reference path, usually given by the centreline of the lanelet, see, e.g. [3]. The software of Innovation 4 also uses this approach, since one of the core benefits is that the reachable sets are convex polygons in the curvilinear frame. This is a necessary precondition for applying convex optimisation, as used in the QP described above.

At the end of the planning cycle, the inputs of the VUT are processed and collisions between the VUT and attackers are detected.

2.2 Algorithmic Overview

The reachable sets for each attacker and the VUT are computed using their initial state and the admissible inputs. Considering the system dynamics, the reachable set is propagated one time step. After the propagation step, the reachable sets are cut to comply with road boundaries and obstacles. Moreover, the reachable sets are restricted both in position and velocity domain to comply with the traffic rules by evaluating automatically generated or user-defined traffic rules in Linear Temporal Logic (LTL). The reachable sets of the entire planning and prediction horizon are stored in a reachability graph that allows easy extraction of driving corridors and predictive queries.

The algorithm for choosing a target state for each attacker aims to minimise the remaining drivable area of the VUT. At the same time, the algorithm asserts that the attacker does not move into a state from which it cannot stay on the road network in subsequent planning cycles. If one or more intersections exist, the centre of the largest one of these is used as the target state. The reasoning behind this heuristic is that by steering towards this position, the other agents are likely to put the VUT under as much "pressure" as possible. If no intersection is found, the agents follow their reference path with constant velocity.

The QP planner can decide to slightly deviate from the target state via constraint relaxation and penalty cost terms, if it means that the trajectory remains feasible in subsequent cycles. This results in a good compromise between forcing the attackers to increase the criticality for the VUT whilst still allowing them to drive towards their own goal, resulting in a more natural behaviour of the attackers. More details on this can be found in [3, Section III.D].

As the default option, all vehicles apart from the VUT are controlled as attackers. However, the software also allows only choosing specific vehicles. Since we use the CommonRoad format and the dataset converters from Deliverable 1.2, most of the data is generated from real-world traffic. The start and goal region of all vehicles, therefore, are derived from real-world traffic participants in most use cases of this project. This supports our decision to find a compromise for the attackers between reaching their goal and increasing the criticality: In real-world traffic, other traffic participants usually do not create critical situations out of malicious intent towards a specific vehicle but as a (likely undesired and unplanned) side effect while following their own goal.

The time-steps for the replanning cycles can be adjusted by the user. However, it must be considered that the choice of the first time-step is crucial for increasing the criticality. If this step is too low, no intersection of drivable areas occurs, and the attackers only follow their reference path. Once the first intersection is detected, the vehicles will typically move closer to the VUT, thereby making it more likely that there will be an intersection of drivable areas in the next cycle. Unfortunately, this first time-step heavily depends on the specific scenario. To tackle this issue, a fast but heavily over-approximative reachability analysis was developed to determine a suitable value. The problem can best be understood by examining the following

example (see Figure 1). Both (A) and (B) show the same scenario with the same time-step size of 1s for replanning and prediction. They only differ in their first time-step. In (A) the automatic time-step generation chooses 2.5.

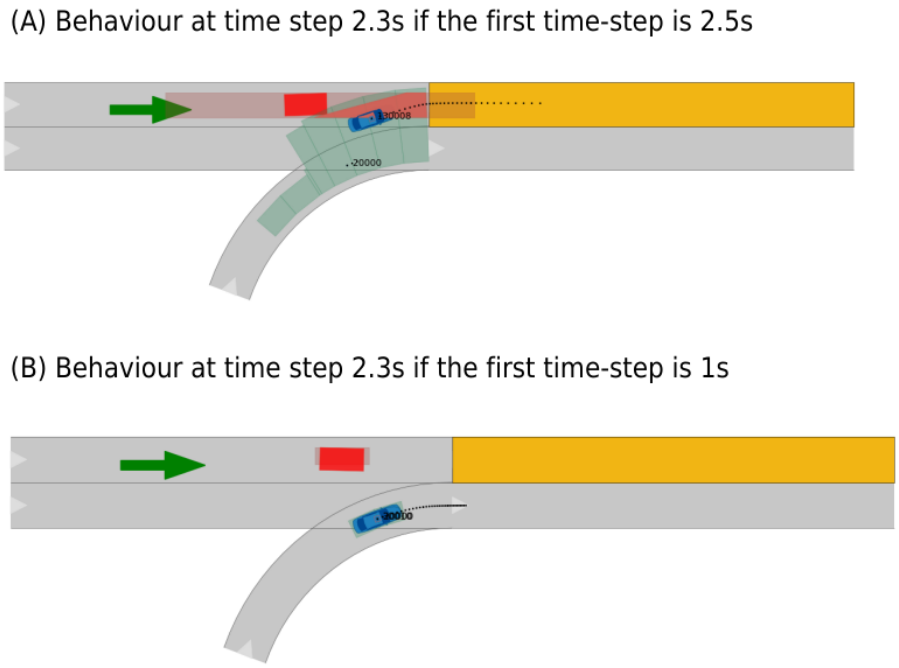


Figure 1: Example for the Influence of the first Time-Step.

2.3 Possible Enhancements

Currently, the i4Driving software meets real-time the requirements, depending on the scenario specification. However, so far, the software relies on other packages that are implemented in Python and, therefore, negatively affect the runtime. The current runtime is in the range of real-time but does not support fast replanning cycles. Migrating the code to C++ will improve the runtime, usually by a factor of around 20. Additionally, once faster replanning cycles are facilitated, the prediction time horizon and the replanning time horizon could be made independent of each other. This could result in even better performance of the attackers since their decision remains predictive over a larger time horizon but is updated more often.

As another point for improvement, the attackers currently only consider the VUT in their decision for the target state and not each other. Future work could develop a method to coordinate the attackers. Furthermore, the attackers can currently drive through each other. Developing methods for negotiating the drivable area in a multi-agent scenario is an active research topic [4, 5]. Future work could incorporate such algorithms in our software.

Another possible enhancement is the incorporation of existing works on robustness metrics [6] and criticality measures [7] for single-agent scenarios to both increase the performance of the attackers and to evaluate the performance of the VUT.

Furthermore, at present only automatic traffic rule generation is supported for the highway merge scenarios, although other rules can be specified in the configuration file manually. A promising future work would be to incorporate automatic LTL rule generation for more of the common rules, e.g., the ones described in [8].

3 Examples

In the following examples, the VUT is always depicted in red, and the attackers are always depicted in blue. The goal regions are depicted as orange polygons. Both the original trajectory without criticality enhancement and the enhanced trajectories of all attacking vehicles are depicted as doted black lines. If collisions occur, they are shown as yellow circles. The drivable areas are translucent polygons.

For detailed examples with animated GIFs, please refer to the */examples* folder of the repository included with this deliverable.

3.1 Merge Scenario with One Attacker

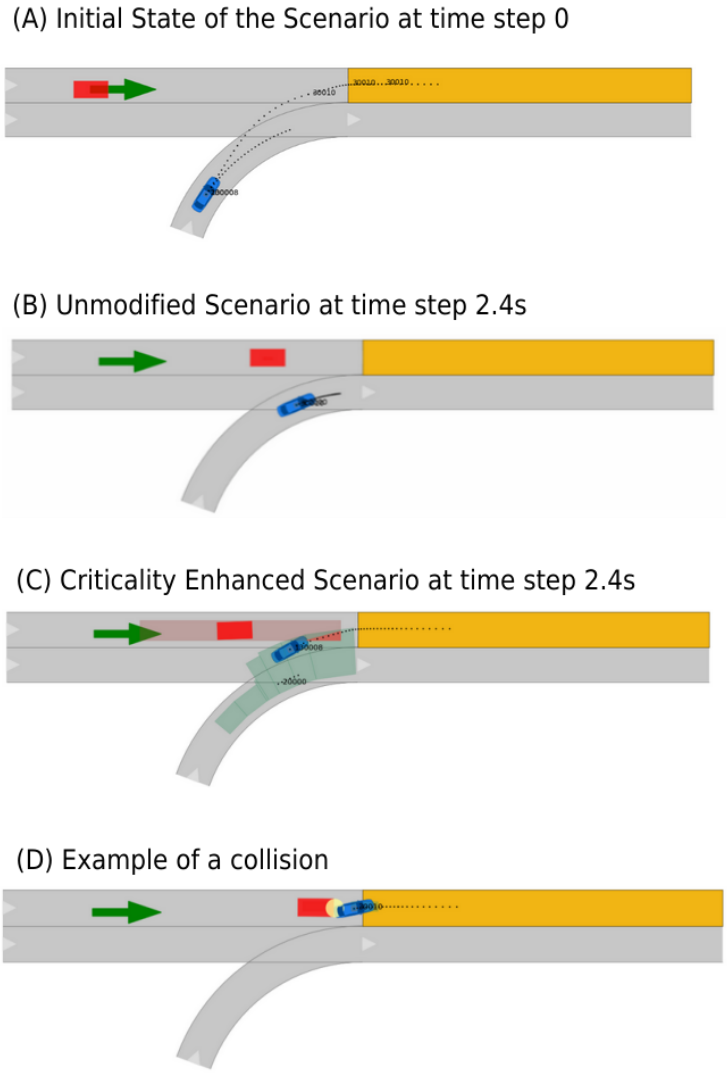


Figure 2: Merge scenario with one attacker.

In Figure 2, (A) shows the initial time step, (B) the unmodified scenario, and (C) and (D) scenarios with increased criticality. The algorithm increases the criticality by choosing a more dangerous entry maneuver, which cuts the VUT. Note that the VUT has an activated Merge Rule; therefore, only states are considered where the VUT does not violate the rule by moving to the right lane. In (D), the VUT planner is deliberately chosen poorly to show collision detection.

3.2 Merge Scenario with Two Attackers

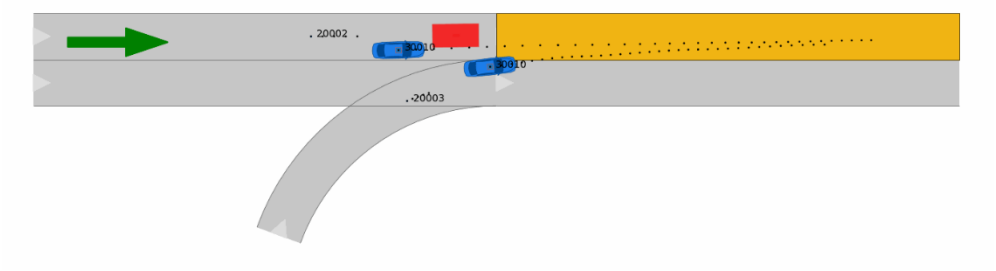


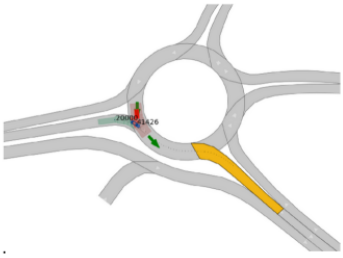
Figure 3: A Merge-Scenario with two Attackers.

In Figure 3, a similar basic scenario to the one above is used, but an additional attacker is placed behind the VUT. The entering vehicle also cuts the VUT, whilst the attacker from behind accelerates. Note that the attacker behind the VUT respects the Merge Rule by not entering the right lane. This example shows that the attackers also consider activated traffic rules.

3.3 Real-world Roundabout Scenario

(A) Unmodified Scenario

(B) Criticality-Increased Scenario with upper vehicle as VUT



(C) Criticality-Increased Scenario with left vehicle as VUT

Figure 4: A Real-World Roundabout-Scenario.

In Figure 4, (A) shows the unmodified scenario, whilst (B) and (C) show criticality-increased scenarios for different VUTs. In (B), the algorithm makes the attacker cut the VUT. In (C), the algorithm chooses to first pass the VUT safely and then suddenly decelerate in the roundabout, since the starting velocities of the VUT

and the attacker were purposefully chosen so that it is impossible for the attacker to just produce criticality by cutting the VUT. This shows the predictive capabilities of the algorithm. This scenario was recorded on the German street B741 near Munich.

3.4 Passing Vehicles

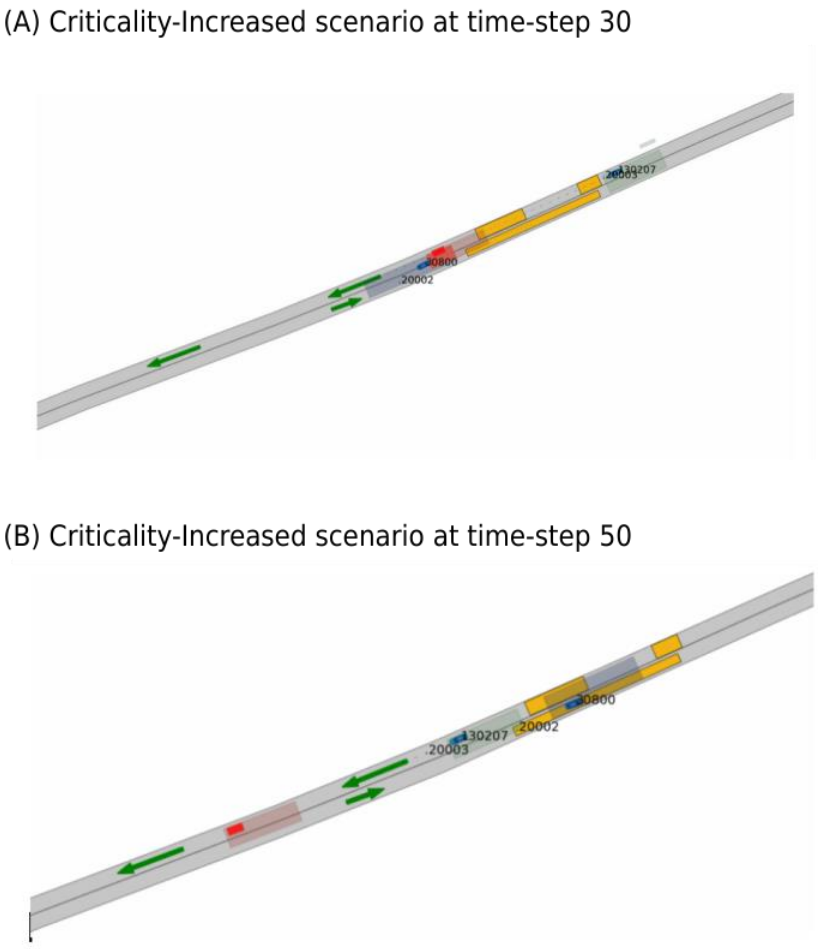


Figure 5: Scenario with Passing Vehicles.

Regarding Figure 5, in (A) the algorithm chooses the attacker coming from the left to move into the middle of both lanes to increase criticality to a near-crash scenario. (B) shows the same criticality-enhanced scenarios a few time-steps later, where the previously attacking vehicle is back on its rightful lane and reaching its goal region. This exemplifies that the algorithm generates trajectories for the attackers that increase the criticality whilst still pursuing the goal from the unmodified scenario.

3.5 Real-world Merge Scenario

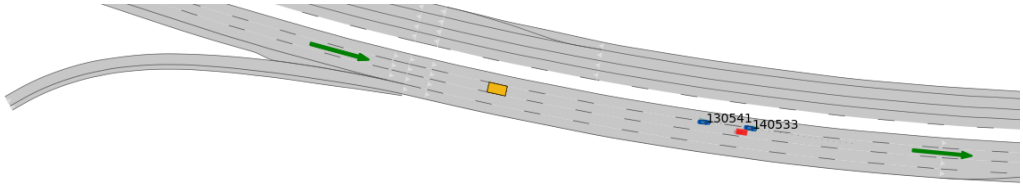


Figure 6: Criticality-Enhanced Real-World Merge Scenario.

In this scenario (Figure 6), the attackers chose to make a risky, near crash (rule compliant) merge to the right. The real-world scenario that this enhancement is based on was recorded on the German Autobahn between Aachen and Cologne. Note that although the original goal-region (in orange) was surpassed, the attackers continue trying to increase the criticality.

4 Software

This section gives an overview of the developed software architecture and describes its basic usage.

4.1 Architecture Overview

The main class of the software architecture is the *ScenarioManager*, which runs the entire scenario, including prediction, planning and input of the VUT on a high abstraction level.

Each vehicle, including the VUT, is represented by an *AgentVehicle* object. Each agent vehicle has its own *ReachInterface* object that handles the entire generation of the reachability graph.

Furthermore, each agent vehicle, including the VUT, has a *TargetTrajectory* object for each planning cycle that generates the trajectory with the aforementioned QP approach. Each state of the vehicle is either a *GeneralState* or a *TargetState* object that saves all state information in both the cartesian and the curvilinear coordinate frame.

Each intersection of reachable sets from an attacker and the VUT is saved as an *OverlappingRegion* object. Each Collision between an attacker and the VUT is saved as a *Collision* object. Both *OverlappingRegion* and *Collision* save and update all information in both the cartesian and the curvilinear coordinate frame.

The entire scenario, including the lanelet network, the obstacles, and the planning problems, is saved in a *ScenarioConfiguration* object. The *ConfigurationBuilder* class is used to build the configuration from the user specifications, the configuration files, and the scenario. The *VirtualPlanningProblemFactory* is used in the *ConfigurationBuilder* for creating virtual dynamic obstacles for evaluating the traffic rule compliance.

For a detailed UML diagram and a visualisation of the architecture, please refer to the attached PDF files.

4.2 Software Usage

The *ScenarioManager* class is also the main software interface for the user. Additionally, the *TimeStepGenerator* class can be used to automatically generate time steps. The *MultiAgentScenarioGenerator* automatically generates a scenario representation in CommonRoad format for the enhanced scenario with multiple planning problems and provides several optional features for selecting appropriate attackers out of a real-world scenario, if desired.

Please refer to the example script *generate_critical_scenarios.py* for an exemplary usage.

The following sections provide more detailed information on the individual aspects of the software.

4.2.1 Scenario Manager

The *ScenarioManager* provides the following steps as methods:

1. *perform_reachability_analysis()*: performs reachability analysis for all agent vehicles
2. *find_overlap_in_reachable_sets():* finds the overlaps in reachable sets
3. *perform_planning():* facilitates the predictive planning of the attackers
4. *get_input_of_vut_vehicle():* exemplary implementation of a VUT trajectory planner
5. *check_collision_of_vut()*: checks collision of VUT with attackers

4.2.2 Using your own VUT / a driving simulator as inputs

The method *get_input_of_vut_vehicle()* gives an exemplary implementation of the interface. The main steps are as follows:

1. Define initial state of VUT and instantiate a *GeneralState* object for it
2. Choose the target state and instantiate a *TargetState* object for it
3. Compute the trajectory between initial state and target state and instantiate a *GeneralState* object for each state.
4. From the generated states, instantiate a *TargetTrajectory* object

Use the *set_trajectory_for_start_step()* method of your VUT vehicle's *AgentVehicle* object and update the *ScenarioConfiguration* and the *ReachInterface* as shown in the example.

4.2.3 Installation

Included in this deliverable is the software repository with the source code and detailed installation instructions in the readme file. Moreover, we provide a prebuilt docker image that allows the execution within the docker container without further installation of all dependencies and submodules. To use the docker container, execute the following steps.

1. Download and install the docker software as described on the website [2].
2. Navigate the terminal to the folder that contains the file "i4driving-inno4-docker.tar" which is included in this deliverable.
3. To load the docker image execute the terminal prompt "docker load --input i4driving-inno4-docker.tar".

Alternatively, you can also build the docker image yourself:

1. Download and install the docker software as described on the website [2].
2. Navigate the terminal to the folder "i4driving-inno4" that is included in this deliverable.
3. Build the docker container with the terminal prompt "docker build –t inno4:latest .".

Once the docker image is available on your system (using one of the ways specified above), you can run the software:

1. Run the software with the terminal prompt "docker run --it inno4:latest bash".
2. In the docker image, the source code is provided in "/home/inno4". Note that you must manually activate the conda environment before running it inside the docker with "conda activate inno4". Use "python3 FILENAME" to run the code, e.g., the example script "generate_critical_scenarios.py".
3. We recommend using the well-known open-source IDE VSCode with its open-source extensions *Remote Development* and *Dev-Containers.* After running a container as described in step 1, you can attach via VSCode to it (blue button in the bottom left corner → attach to running container) and test / use / adapt to code as you desire.
4. Per default, the results of "generate_critical_scenarios.py" are visualized in "/home/inno4/i4crit/output".

Please refer to the docker documentation for further information. We recommend mounting both the input data folder as well as the output data folder via "–v HOSTPATH:CONTAINERPATH" *flag*.

Inside the Docker container, you find a pre-installed Linux operating system and the installed software described in this document in "/home/inno4/i4crit", where you will enter the docker by default.

[2] https://docs.docker.com/desktop/

You can use the software as if you installed it following the readme file installation procedure. You can mount your own scenarios into "/home/inno4/i4crit/scenario_data_root/single_agent_scenarios". The software will automatically convert it into a scenario with attackers, stored in "/home/inno4/i4crit/scenario_data_root/scenarios".

5 References

[1] M. Klischat and M. Althoff, "Falsifying Motion Plans of Autonomous Vehicles With Abstractly Specified Traffic Scenarios," in *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 2, pp. 1717-1730, Feb. 2023, doi: 10.1109/TIV.2022.3191179.

[2] E. Irani Liu and M. Althoff, "Specification-Compliant Driving Corridors for Motion Planning of Automated Vehicles," in *IEEE Transactions on Intelligent Vehicles*, doi: 10.1109/TIV.2023.3289580.

[3] F. Finkeldei and M. Althoff, “Synthesizing Traffic Scenarios from Formal Specifications Using Reachability Analysis”. *2023 IEEE International Conference on Intelligent Transportation Systems Conference (ITSC)*, Bilbao, Spain.

[4] S. Manzinger and M. Althoff, "Negotiation of drivable areas of cooperative vehicles for conflict resolution," *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, 2017, pp. 1-8, doi: 10.1109/ITSC.2017.8317765.

[5] S. Manzinger and M. Althoff, "Tactical Decision Making for Cooperative Vehicles Using Reachable Sets," *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, USA, 2018, pp. 444-451, doi: 10.1109/ITSC.2018.8569560.

[6] L. Gressenbuch and M. Althoff, "Predictive Monitoring of Traffic Rules," *2021 IEEE International Conference on Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, IN, USA, 2021, pp. 915-922, doi: 10.1109/ITSC48978.2021.9564432.

[7] Lin, Y., & Althoff, M. (2023). CommonRoad-CriMe: A toolbox for criticality measures of autonomous vehicles. In *2023 IEEE Intelligent Vehicles Symposium (IV)*.

[8] S. Maierhofer, A. -K. Rettinger, E. C. Mayer and M. Althoff, "Formalization of Interstate Traffic Rules in Temporal Logic," *2020 IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas, NV, USA, 2020, pp. 752-759, doi: 10.1109/IV47402.2020.9304549.