OSCCAR:
FUTURE OCCUPANT SAFETY FOR CRASHES IN CARS

openPASS framework for integrated safety assessment

Document Type	Deliverable
Document Number	D1.2
Primary Author(s)	Uwe Wössner | USTUTT/HLRS
Document Version / Status	1.4 | Final
Distribution Level	PU (public)

Project Acronym	OSCCAR
Project Title	FUTURE OCCUPANT SAFETY FOR CRASHES IN CARS
Project Website	www.osccarproject.eu
Project Coordinator	Werner Leitgeb | VIF | werner.leitgeb@v2c2.at
Grant Agreement Number	768947
Date of latest version of Annex I against which the assessment will be made	2020-09-09
Upload by coordinator:	First submitted: 2021-11-02 Re-submitted: 2022-04-11

OSCCAR has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 768947.

CONTRIBUTORS

Name	Organization	Name	Organization
Uwe Wössner	USTUTT/HLRS		
Jan Dobberstein	MERCEDES BENZ AG		
Daniel Schmidt	Robert Bosch GmbH		

FORMAL REVIEWERS

Name	Organization	Date
Ernst Tomasch	TU Graz	2020-11-05
Ernst Tomasch	TU Graz	2020-11-23
Peter Wimmer	Virtual Vehicle	2020-11-13
Peter Wimmer	Virtual Vehicle	2020-11-24

DOCUMENT HISTORY

Revision	Date	Author / Organization	Description
0.9	2020-08-01	Uwe Wössner/USTUTT	Initial draft
1.0	2020-10-01	Jan Dobberstein/DAIMLER	Scenarios
1.1	2020-10-20	Uwe Wössner/USTUTT	Installation
1.2	2020-11-09	Uwe Wössner/USTUTT	Modifications after review
1.3	2020-11-25	Uwe Wössner/USTUTT	Adjusted to latest version
1.4	2022-03-15	Uwe Wössner/USTUTT	Adjust to EU review comments

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS AND DEFINITIONS

Term	Definition
AD	Automated Driving
ADAS	Advanced Driver Assistance System
ASAM	Association for Standardization of Automation and Measuring Systems
DEM	Digital Elevation Model
DSL	Domain Specific Language
EPL	Eclipse Public License
FE	Finite Element
GUI	Graphical User Interface
LGPL	GNU Lesser General Public License
USB	Universal Serial Bus
WG	Working Group
VR	Virtual Reality
XML	eXtensible Markup Language

1 EXECUTIVE SUMMARY

The OSCCAR deliverable “D1.2 openPASS framework for integrated safety assessment” consists mainly of a software development and is accompanied by this written document. Here, we give an overview of the software framework itself including our additional developments, an installation guide and exemplary test scenarios. With the software developments in D1.2, the user is now able to describe, simulate and visualize crash scenarios. Please note, this deliverable is neither motivating relevant test scenarios nor describing simulations or simulation results. All these topics are addressed in great detail in the OSCCAR deliverable “D1.3 Future collision type matrix”. This documents gives a brief destription of the software stack that has been developed within WP1, especially the OpenDRIVE designer OddLot to interactively design OpenDRIVE roads, extensions to the openPASS accident simulation framework which allow integrating openPASS into other applications such as driving simulators and finally extensions to OpenCOVER in order to visualize openPASS traffic and accidents.

All components are developed Open Source under either LGPL or EPL license and are available for commercial and non-commercial use to both the OSCCAR consortium and the general public.

Keywords: OpenPASS, OddLot, COVISE, OpenCOVER, OpenDRIVE, OpenSCENARIO

2 OVERVIEW

As outlined in the OSCCAR deliverable “D1.1 Accident data analysis - remaining accidents and crash configurations of automated vehicles in mixed traffic”[1] in full detail, the goal of OSCCAR WP1 is to use current accident data and predict traffic scenarios as well as crash configurations which automated vehicles will be exposed to in the future. The work reported in D1.1 [1] incorporated the development of a methodological framework for integrated safety assessment, accident data analysis and literature review – as well as the application of the methods to provide baseline results. While the work in D1.1 is mainly an interpretation of results for the urban AD model (selection of relevant single crash configurations) and how to process them (e.g., in a generic crash pulse estimation), all other work on future accidents is covered in D1.3 serving as a final report of the WP1 work. The work in “D1.2 openPASS framework for integrated safety assessment” describes the open-source accident simulation tool openPASS as well as the OSCCAR specific software developments as an enabler for a harmonized, transparent simulation approach which was the aim of OSCCAR from the start. This goal, including the decision to use the openPASS framework in the first place, was explicitly outlined in the Description of Work in the OSCCAR project proposal. This activity was aligned with the needs identified during the project as described in D1.1 [see Chapter 8.2.3] and the capabilities of the available open-source software.

The role of the “openPASS approach” in the work on “prediction of what is expected to remain” beyond D1.1 is illustrated in Figure 1. In D1.1, accident re-simulation led to relevant crash configurations to derive “future test cases” for urban AD concepts. Taking into account the safety impact of AD models, the remaining crashes motivated new relevant crash configurations, which were provided to the subsequent work packages in terms of generic crash pulses. For an AD model in motorway scenarios, it was discussed in D1.1 to not rely on accident re-simulation only, but to investigate a traffic simulation approach to consider potential crashes from stochastic multi-agent simulations.

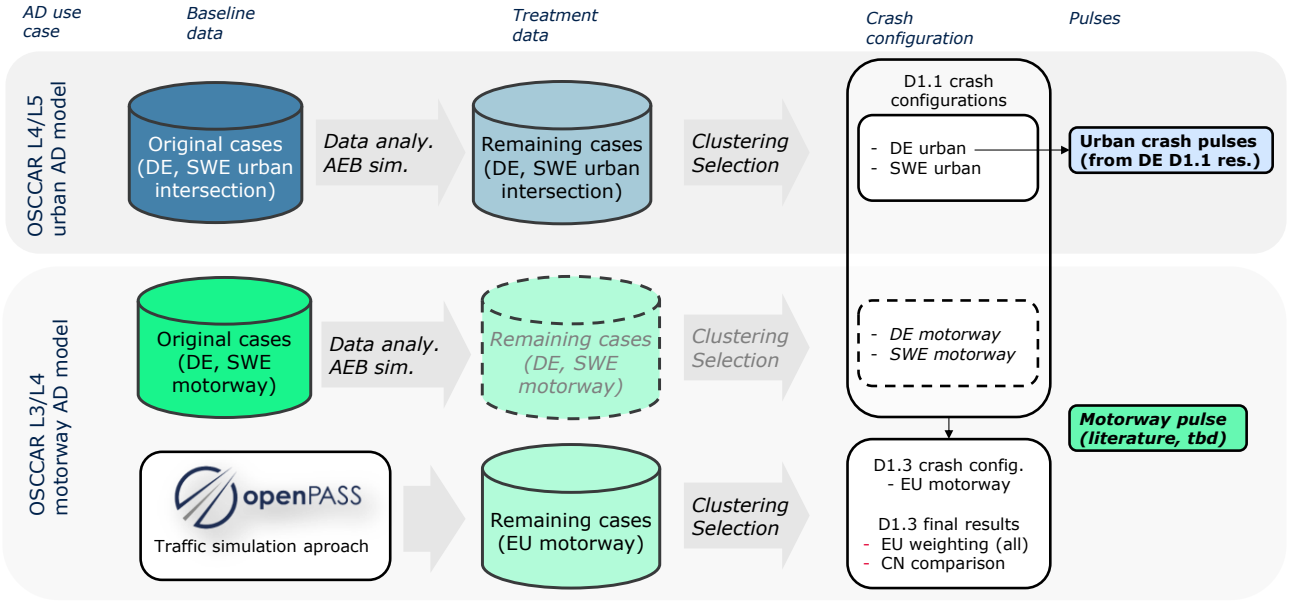


Figure 1. Schematic integration of the openPASS framework in the OSCCAR project

At this point, the benefits of traffic scenario simulation approaches from previous projects, e. g., AdaptIVe [9] and the open-source approach of HLRS at USTUTT were taken up in OSCCAR. For D1.2, the openPASS simulation core (v0.6) was equipped with additional features (advanced driver, crash configuration calculation) and combined with further tools (OddLot, see 3.4, OpenCOVER, see

3.6) to a comprehensive open-source toolset This toolset is able to address relevant future crash configurations especially for human / AD mixed traffic on motorways. The results of these openPASS simulation studies with the “OSCCAR use case” will be described in D1.3.

The requirements for D1.2 can be summarized in the following aspects:

- Full tool chain, incorporating the openPASS simulation platform and components needed to run the traffic simulations as well as tools for pre-processing (e. g. changing road or scenario parameters) and post-processing (3D visualization of openPASS traffic)
- Simulation framework capable of large scale, stochastic traffic simulations, accounting for accidents being rare events in realistic traffic (about in the order of one accident per million km driven)
- D1.1/D1.2 alignment: crash configurations, collision speeds and delta-v interpretation (filtering for severity)
- Integration of openPASS agent components needed for setting up simulation scenarios such as the driver model “ModularDriver”, developed alongside, but outside of OSCCAR by openPASS open-source committers from in-tech, ITK Engineering GmbH and AMFD GmbH.
- Exemplary configurations demonstrating the experimental setup used for providing traffic simulation results for AD on motorways (results in D1.3)
- Installer including all tools the user needs (OddLot, OpenCOVER; openPASS)

This deliverable serves as technical report providing an overview of the technology, but also as a “user guide” for the implementation of the “OSCCAR use case”, built upon openPASS as platform. The actual implementation and application of the openPASS traffic models aligned to the openPASS assessment toolchain and the results from these simulations will be described and discussed in the final WP1 deliverable D1.3. This incorporates e. g. the detailed description of the features of the ModularDriver and the steps taken for validating the baseline traffic characteristics.

3 FRAMEWORK DESCRIPTION

The following paragraphs give a brief overview of the collection of software systems which have been developed and extended within OSCCAR to simulate pre-crash scenarios.

3.1 Framework overview

To simulate pre-crash behaviour several boundary conditions need to be defined. Most important is a detailed description of the road geometry which includes road surface properties, lane boundaries, road markings, road signals and signs, sidewalks, parking places but also guard rails, vegetations, buildings and much more. All of these have and influence on the sensor performance and the behaviour of future driver assistance- and automated driving systems.

The generic traffic scenarios investigated in OSCCAR most often must be modelled by hand from scratch to exactly fit the requirement of the study case. To support the definition of these simulation scenarios we developed a complete tool chain consisting of OddLot, an editor for Road systems and Traffic scenarios, openPASS, a traffic and accident simulation platform, and OpenCOVER, an interactive visualization system which allows visualizing traffic- and crash simulations.

All of these components are described in the following paragraphs in the order of their dependency.

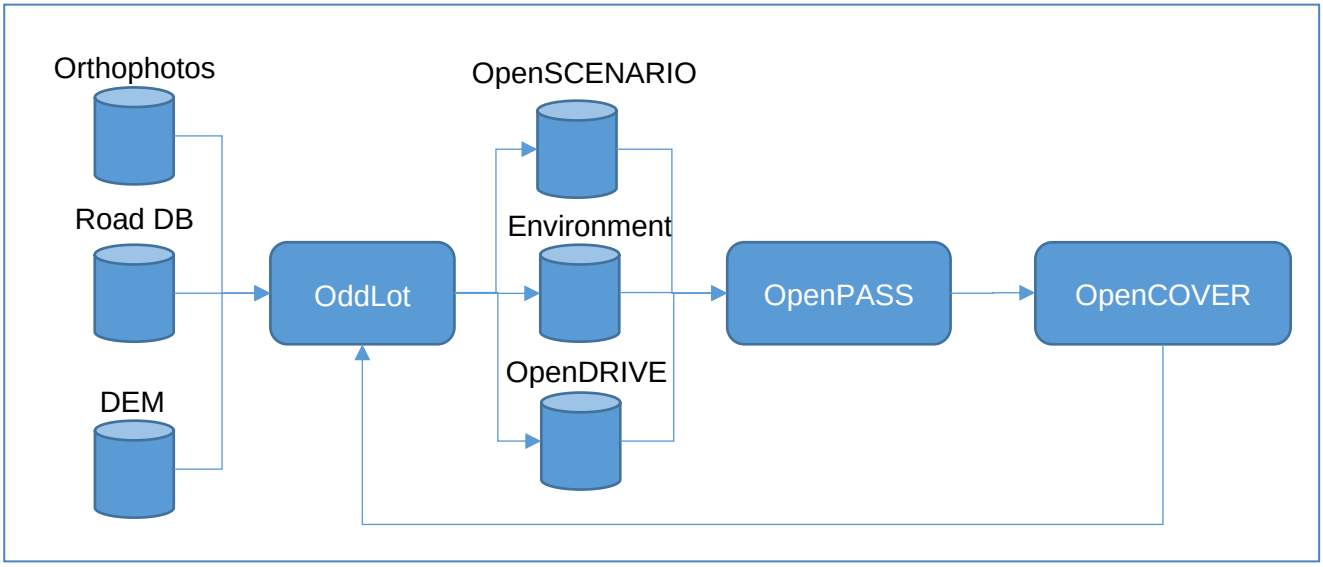


Figure 2. Schematic overview of the final D1.2 openPASS ecosystem

3.2 OpenDRIVE®

OpenDRIVE®[1] [4] is a format to describe road networks using the Extensible Markup Language (XML). The geometry of the roads can be defined, as well as the lane layout, the elevation, and the lateral profile. The connection between roads is specified by the road linkage, junctions by a combination of connecting with incoming roads. Elements, which are relevant to the traffic flow, like signals, can be integrated and assigned with a country code, to match regional characteristics.

[1] OpenDRIVE is registered as a trademark by VIRES Simulationstechnologie GmbH while „ASAM OpenDRIVE“ is registered by ASAM

Objects which have a close relation to the road, e.g., bridges and tunnels, bushes, barriers, and poles, can also be described. OpenDRIVE also supports the definition of rail roads. Most elements of OpenDrive are static, only signals and objects can be dynamic.

The standard was initiated by Daimler AG, DLR e.V. and VIRES Simulationstechnologie GmbH. Further development is driven by ASAM (Association for Standardization of Automation and Measuring Systems). In March 2020 Version 1.6.0 was released.

3.3 OpenSCENARIO®

OpenSCENARIO®[2] [5] is a format to describe the dynamic behaviour of driving scenarios. Given a road network, the actions of actors, such as pedestrians, cyclists, and cars, can be triggered by time dependent events or by matching relative conditions. Actors can follow routes and trajectories. Catalogs help to administrate and reuse objects. OpenSCENARIO is used in driving simulation, traffic simulation, virtual development, test and validation of driving assistance functions, automated and autonomous driving.

The standard was initiated by VIRES Simulationstechnologie GmbH. At an early stage, HLRS contributed to the standard by evaluating the necessary elements of the language and resolving inconsistencies. Since 2018 ASAM (Association for Standardization of Automation and Measuring Systems) administers the ongoing process to improve the open standard to the needs of the automotive industry. While OpenSCENARIO 1.0 is serialized in an XML format, OpenSCENARIO 2.0 will be based on a DSL, allowing to define abstract and concrete scenarios, flexible scenario composition, run-time parametrization and embedded script execution. Both standards will be merged in the future. HLRS has been participating in the concept phase of both OpenSCENARIO 1.0 and 2.0 [6] and will further contribute to their implementation.

In order to create a generic parser for the OpenSCENARIO 1.0 file format a number of modifications of the initial OpenSCENARIO draft have been proposed and accepted by the consortium. They are now part of the 1.0 version of the standard. The parser implementation is published under LGPLV3 as part of the COVISE git repository [2][1] in the directory src/OpenCOVER/DrivingSim/OpenScenario. It consists of a tool to automatically generate the parser source code from the Standard XML Schema files as well as a working parser for the 1.0 version of OpenSCENARIO. The parser is generic and thus not depending on any other part of the COVISE system, thus it can also be used in other projects without modification.

In OSCCAR, the parser has been used in OddLot to load and save OpenSCENARIO files as well as in the OpenSCENARIO plugin in OpenCOVER, see 3.6. openPASS uses its own scenario importing system to implement the dynamic behaviour described in the Scenario.xosc files in the openPASS simulation environment (“the World”).

3.4 OddLot

In order to create high resolution road surfaces for virtual crash scenarios in OSCCAR, OddLot (see Figure 3, an OpenDRIVE editor has been extended to support editing OpenSCENARIO files. Furthermore, a link to the OpenCOVER visualization system (see Chapter 3.6) has been developed which allows modelling of 3D road systems after artificial 3D models of test environments or after digital elevation models and 3D city models, to recreate existing roads and their environment. OddLot

[2] OpenSCENARIO is registered as a trademark by VIRES Simulationstechnologie GmbH while „ASAM OpenSCENARIO“ is registered by ASAM

has also been extended to load background maps from Google Maps or Bing Maps in order to quickly create realistic street networks whenever there is no high-resolution 3D environment available that could be loaded through OpenCOVER.

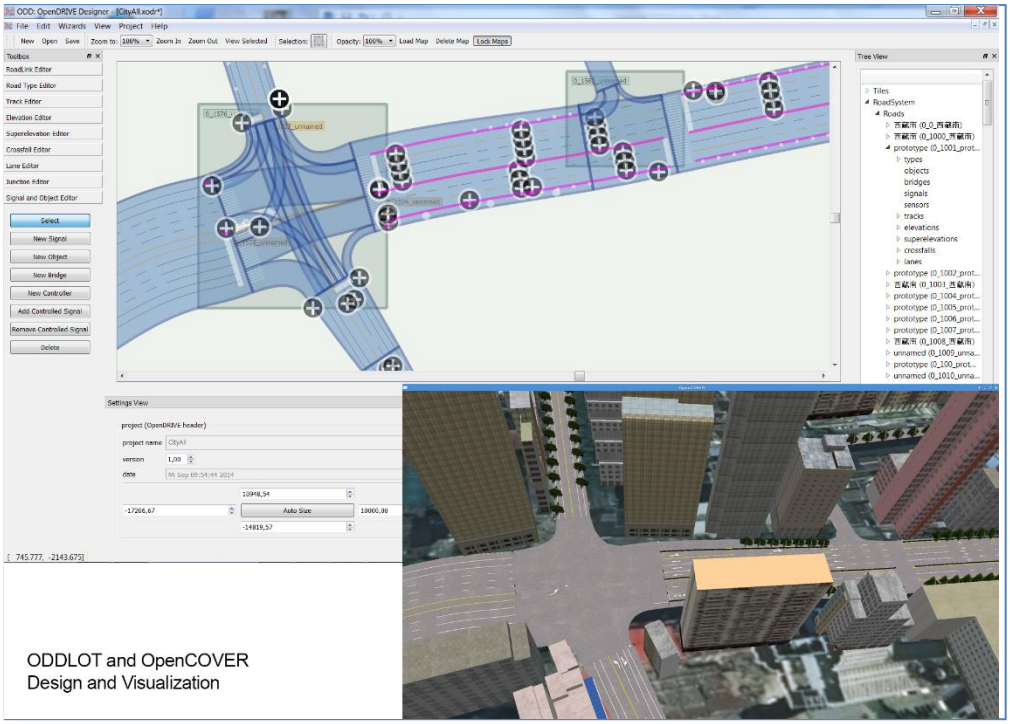


Figure 3. OddLot OpenDRIVE/OpenSCENARIO editor

3.5 openPASS

openPASS is an open-source platform for the prospective safety assessment of ADAS and AD technologies. Originally, the term “openPASS” formed a backronym for “Open Platform for the Assessment of Safety Systems”, but openPASS has been expanded beyond the limitations of safety systems towards any kind of ADAS and AD functions. Thus, one of the main agreed target objectives is to become a broadly accepted effectiveness assessment platform for ADAS and AD functions. In line with our target objective, the Eclipse working group behind openPASS aims to develop a trustworthy, reliable, and transparent platform.

The openPASS platform concept is shown in the following picture.

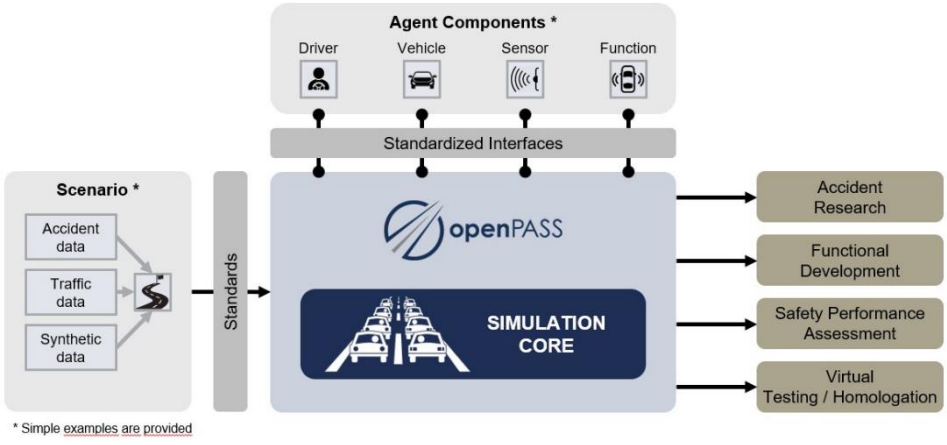


Figure 4. openPASS Platform

The platform is built up in a modular manner. The simulation core calculates different simulation runs. Scenarios and agent components can connect to the core over standardized interfaces like OpenSCENARIO, OpenDRIVE, OSI and FMI. The openPASS open-source project provides simple examples of scenarios and agent components. Consequently, the user can freely connect their own scenarios and agent components to the simulation core. Possible use cases of openPASS are accident research, functional development, safety performance assessment, virtual testing, and virtual homologation.

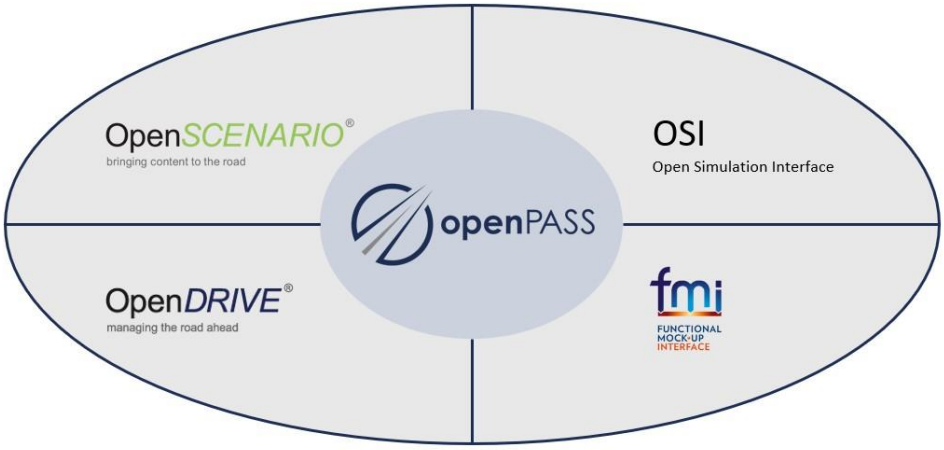


Figure 5. Top-level architecture

3.5.1 Overview of openPASS architecture

OpenPASS has two main parts: the graphical user interface (GUI) and the simulation framework, see [8]. The GUI is meant to give the user easy access to configure experiments prior to simulation. To perform a traffic simulation, input must be available, which can be prepared and offered by one of the GUI plugins. In the GUI, the users will conveniently select traffic scenarios, traffic participants (including vehicles, drivers, and systems to be tested) as well as post-processing of the simulation results. openPASS simulation accepts open formats such as openDRIVE, openSCENARIO and several openPASS-specific formats. Each file defines either simulation parameters, traffic situation, traffic composition, events, or other key data. It is built as modular software allowing the attachment of independent or dependent plugins. A plugin can read input data (e.g., from a PCM database), process this data and prepare an experiment configuration, which can be understood by the simulation.

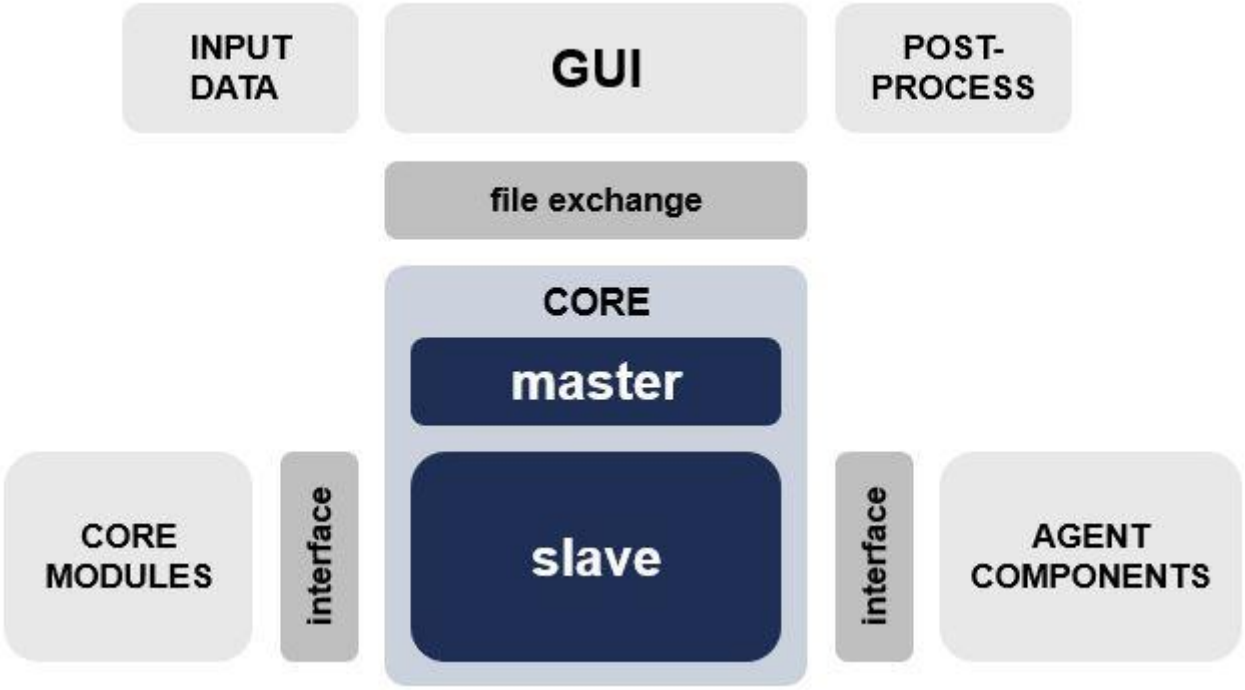


Figure 6. Top level system architecture diagram

The simulation framework is a standalone executable (to be precise, two of them). It consists of:

- the simulation core (master and slave)
- the core modules
- and the agent components.

There is a motion to rename the “master” and “slave” components to a less controversial name. Within this document we refer to the current version of openPASS and would like to stick to the naming of that version to reduce confusion.

The core is a mostly generic assembly of data I/O routines, scheduler, and a collection of interfaces. The master executable is meant to only collect and organize input data. The slave executable is the one performing the actual simulation. A master can, if configured so, start several experiments simultaneously by triggering several slaves.

The core modules and agent components belong to the slave application. Both communicate with the slave via interfaces and bound dynamically at runtime, see Figure 7.

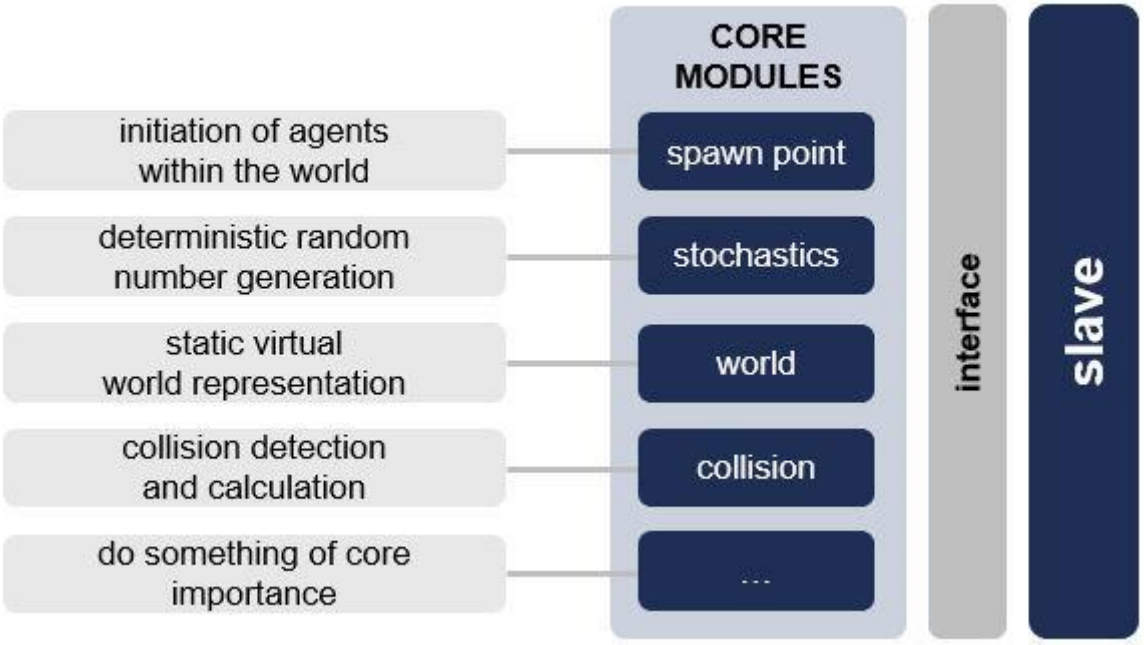


Figure 7. Core module diagram

Core modules are all singletons and used by the slave and/or agent components. These are necessary for every simulation to run and cover basic needs like e.g.:

- contain world representation
- initiate agents within the world
- perform random number operations
- detect events
- log and output simulation results

On the other hand, agent components (Figure 8) represent a composition of participants or agents. They define the behaviour and dynamics of each agent. A set of such components is then called “system”.

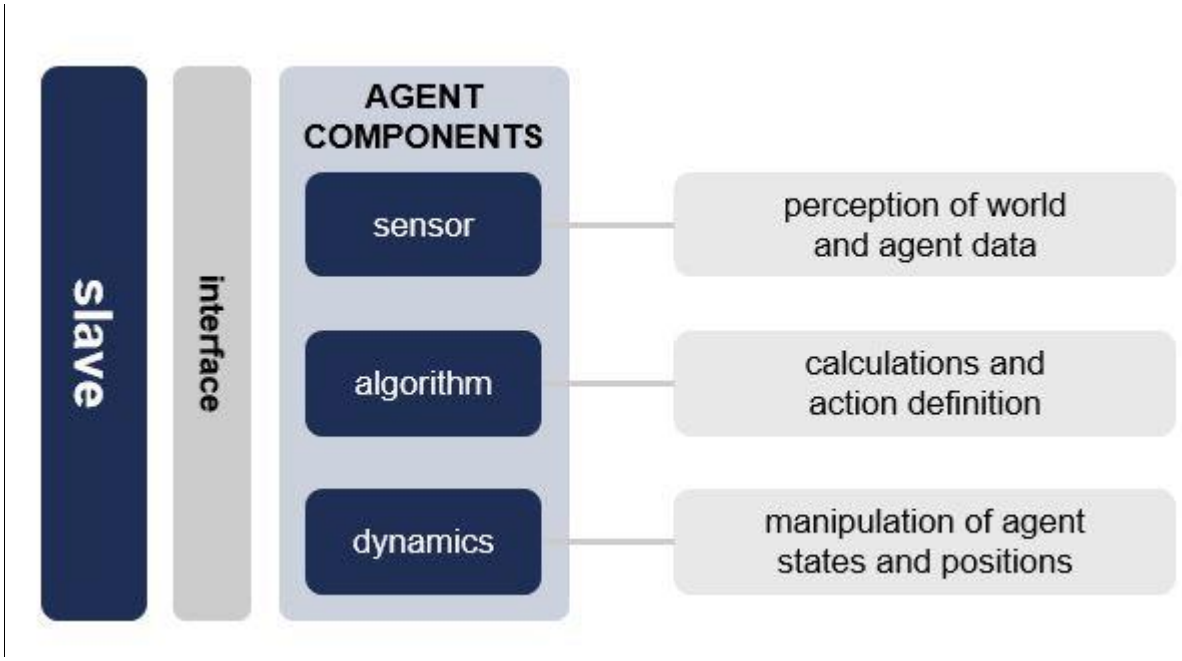


Figure 8. Agent component diagram

Unlike core modules, components can be freely selected and assembled by the user according to the scope of application. A typical system would consist of a sensor perceiving the environment, an algorithm performing analysis of this environment and making decisions and a dynamics algorithm receiving directives from the algorithm and calculating the actual physical simulation step. When assembling a system, the user shall of course care about connecting the chosen component via signals, which are understood by the sender and the receiver.

Completing the loop back to the graphical UI, a plugin can await the completion of one or several experiments and collect the output produced by the simulation in order to evaluate and/or visualize the results.

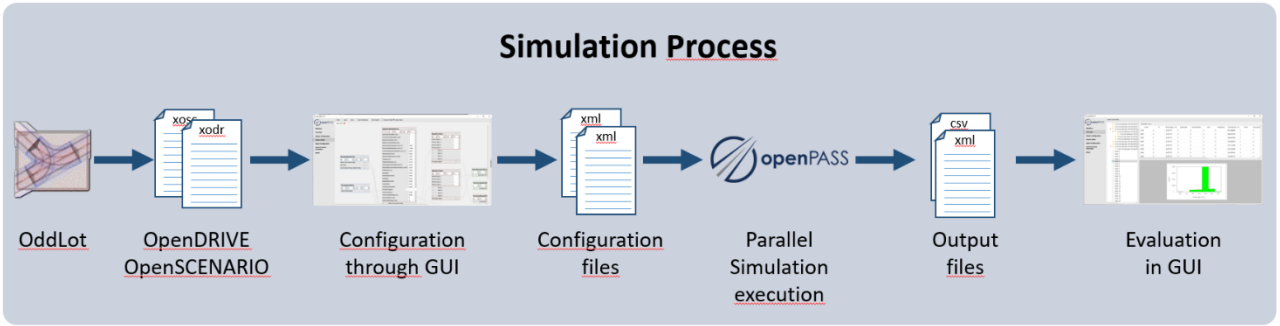


Figure 9. User perspective of the simulation process

Figure 9 shows the simulation process in openPASS. The user sets up the desired simulation run via configurations in the GUI. After the setup of the configuration files, the simulation can be executed. As an output .csv and .xml files are generated. In the future, visualization and evaluation of the simulation results should be possible from within the GUI. This feature is still under development.

3.5.2 Alignment of requirements from OSCCAR towards openPASS

The open-source development in openPASS implementing openSCENARIO and openDRIVE was aligned to the OSCCAR use case for traffic simulation as described in D1.1. The support of these standards is an overall goal of openPASS, independent of the OSCCAR project. Hence, the "standard functionality" of openPASS v0.6 already provided the main capabilities to configure and to run traffic simulation of motorway scenarios. Furthermore, the requirements from OSCCAR towards openPASS were matched with the simulation capabilities.

However, two major features were defined as specific requirements towards openPASS from the OSCCAR perspective and subsequently developed alongside OSCCAR, but outside, i.e., not by OSCCAR partners:

- Driver model "modular driver": the openPASS traffic model had to be enhanced to be used as a "accident prediction tool" in OSCCAR which enables the user to set up more than regular traffic, with agents adapting longitudinally to each other and creating realistic traffic flow as in the exemplary modules in v0.6. Additionally, the "modular driver" was developed, an intelligent, yet accident-causing modular driver model. This incorporates wrapping of SUMO driver sub-models car following and lane change behavior, additional sub-models of cognitive information flow and mechanisms to deteriorate this flow e. g. by inserting avert view or reaction times. For more information, see the preliminary documentation of the driver model concept and its implementation on the openPASS wiki [8].
- Crash configuration interpretation: Furthermore, an enhanced impact model that detects and writes out crash configurations has been implemented. The simulationOutput.xml files (see Figure 10) are written according to the definition in [9]. They are used in OSCCAR to come from pre-crash simulation results (mitigated, remaining accidents) to a selection of exact constellations to be used in FE simulation. This allows integrating openPASS in the OSCCAR assessment tool chain as described in D2.1 (Test Case Matrix ➔ crash configuration dimension) and D1.1 (WP1 pre-process: how to process, filter, cluster etc. remaining accidents for relevant crash configurations).

```
<RunResults>
    <RunResult RunId="0">
        <RunStatistics>
            <RandomSeed>56478</RandomSeed>
            <VisibilityDistance>125</VisibilityDistance>
            <StopReason>Due to time out</StopReason>
            <StopTime>-1</StopTime>
            <EgoAccident>False</EgoAccident>
            <NumberOfAccidentsInFollowers>0</NumberOfAccidentsInFollowers>
            <NumberOfArbitraryAccidents>4</NumberOfArbitraryAccidents>
            <TotalDistanceTraveled>29651.5</TotalDistanceTraveled>
            <EgoDistanceTraveled>755.79</EgoDistanceTraveled>
        </RunStatistics>
        <Events>
            <Event Id="0" Time="71300" Source="EventDetector" Type="Collision">
                <EventParameter Key="CollisionWithAgent" Value="true"/>
                <EventParameter Key="CollisionAgentId" Value="8"/>
                <EventParameter Key="CollisionOpponentId" Value="35"/>
                <EventParameter Key="AgentVelocity" Value="2.019567"/>
                <EventParameter Key="AgentVelocityChange" Value="0.375800"/>
                <EventParameter Key="AgentVelocityDirection" Value="-0.053793"/>
                <EventParameter Key="AgentPointOfContactLocalX" Value="-2.621945"/>
                <EventParameter Key="AgentPointOfContactLocalY" Value="0.983547"/>
                <EventParameter Key="AgentCollisionVelocity" Value="1.656875"/>
                <EventParameter Key="AgentSliding" Value="0"/>
                <EventParameter Key="OpponentVelocity" Value="1.896731"/>
                <EventParameter Key="OpponentVelocityChange" Value="0.375800"/>
                <EventParameter Key="OpponentVelocityDirection" Value="0.030821"/>
                <EventParameter Key="OYA" Value="-1.273328"/>
                <EventParameter Key="HCPAo" Value="157.625155"/>
                <EventParameter Key="OCPAo" Value="-18.415437"/>
                <EventParameter Key="HCPA" Value="135.193639"/>
                <EventParameter Key="OCPA" Value="-38.777298"/>
```

Figure 10. Exemplary crash configuration description of an openPASS agent collision

3.6 COVISE/OpenCOVER

COVISE stands for COllaborative VIsualization and Simulation Environment. It is an extendable distributed software environment to integrate simulations, postprocessing and visualisation functionalities in a seamless manner. From the beginning COVISE was designed for collaborative work allowing engineers and scientists to work together over the Internet.

Through VR and AR support, the users can analyse their datasets intuitively in a fully immersive environment through state-of-the-art visualization techniques including GP-GPU post processing, Volume rendering and fast sphere rendering. Physical prototypes or experiments can be included in the analysis process through Augmented Reality techniques.

OpenCOVER is the COVISE renderer for immersive virtual environments, supporting Virtual environments ranging from workbenches over powerwalls, curved screens up to full domes or CAVEs. It is based on OpenSceneGraph, an open-source high performance 3D graphics toolkit, to make optimal use of rendering hardware. OpenCOVER stands for "Open COVISE Virtual Environment" and is an integral part of the COVISE visualization and simulation environment.

Figure 11. OpenCOVER used for driving simulation scenario visualisation

OpenCOVER has been extended to support driving simulation applications. Plugins provide access to several different driving simulation hardware systems ranging from simple USB steering wheels over motion platforms to full driving simulators such as the one at FKFS, University of Stuttgart or Porsche in Weissach. Endless terrains are supported through a high-performance tiled database system based on Virtual Planet Builder. An OpenDRIVE plugin provides support for high-definition street geometry and street accessories and Driving Scenarios are supported through OpenSCENARIO and a proprietary scenario description based on OpenDRIVE. A live interface to SUMO enables the visualization of large-scale traffic simulations.

Within OSCCAR, OpenCOVER has been extended by a live interface to OddLot and openPASS. OpenSCENARIO and OpenDRIVE support has also been extended substantially.

4 SOFTWARE INSTALLATION AND TEST SCENARIOS

4.1 Installation

The demonstrator consists of latest (2020/11/25) binary Windows builds of COVISE and openPASS, available on the following HLRS server:

- openPASS: https://fs.hlrs.de/projects/covise/support/download/openPASS/
- COVISE: https://fs.hlrs.de/projects/covise/support/download/

The source code incorporated in these builds are online here (version as of 2020/11/25):

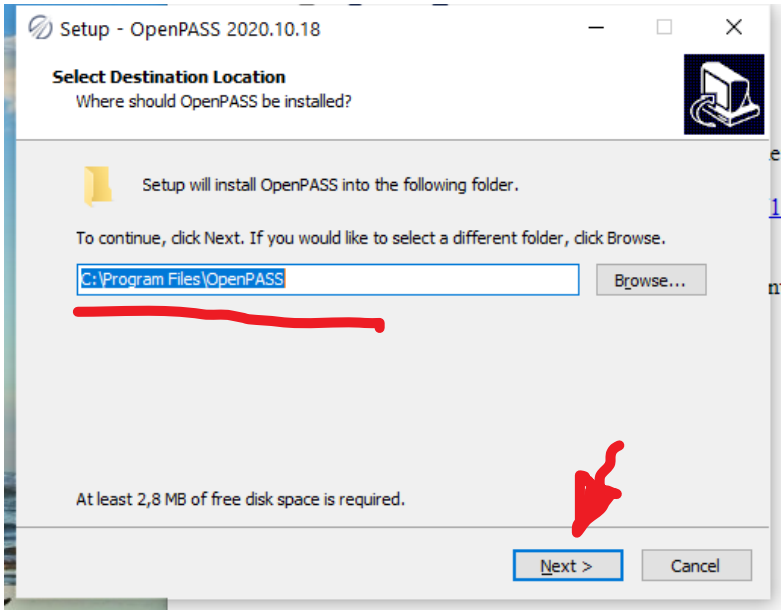
- openPASS (hlrs brach): https://git.eclipse.org/c/simopenpass/simopenpass.git/tree/?h=hlrs
- COVISE: https://github.com/hlrs-vis/covise

Pitfalls during installation

The installation should be straightforward: download both OpenPASS and COVISE installers and run them as Administrator. Default values for the installation should be applicable in most cases.

When running the installer for the first time, Windows wants to protect you and refuses to run the installer:

After you click on additional information, a new button “Install anyway” will appear.



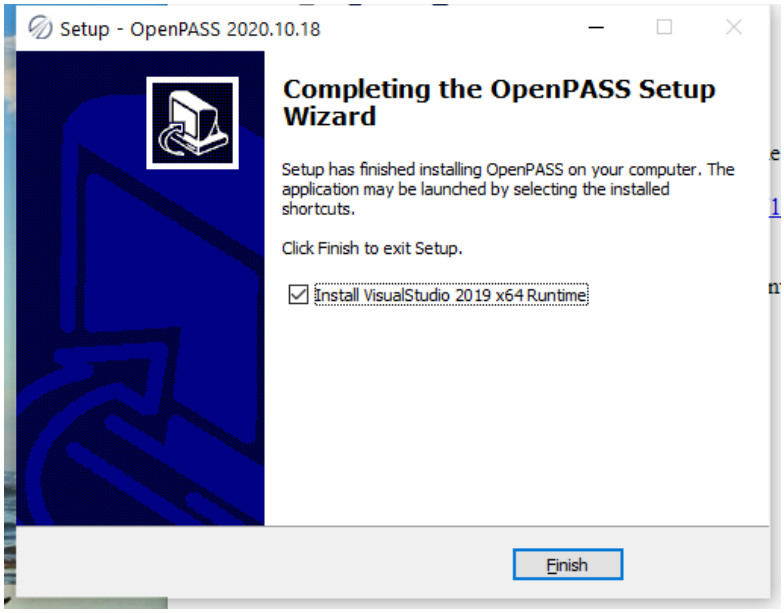
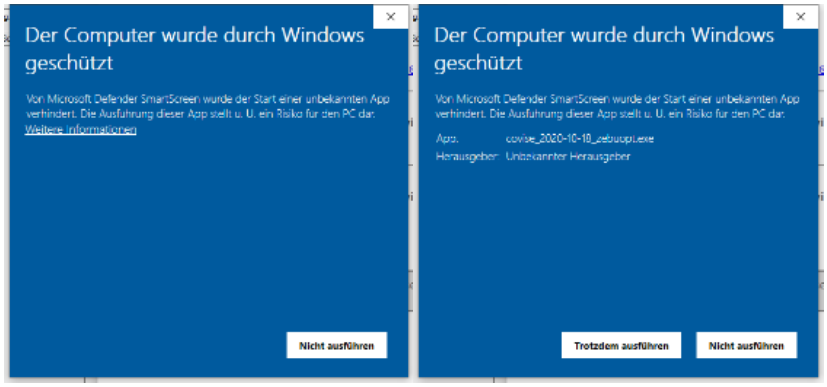


Figure 12. openPASS installation process

Following the installation accepting default values, you are prompted to install the Visual C++ runtime environment. On some computers the system will automatically reboot after installation of the Visual C++ runtime, thus be prepared and save all your work before starting the installation.

Please do not miss the security shield in the task bar. You have to confirm the installation of all dependencies (if they are not already installed).

You can now proceed with installing COVISE.



After running the installer as Administrator, you might also have to confirm the installation by clicking on “additional information” and “install anyway”.

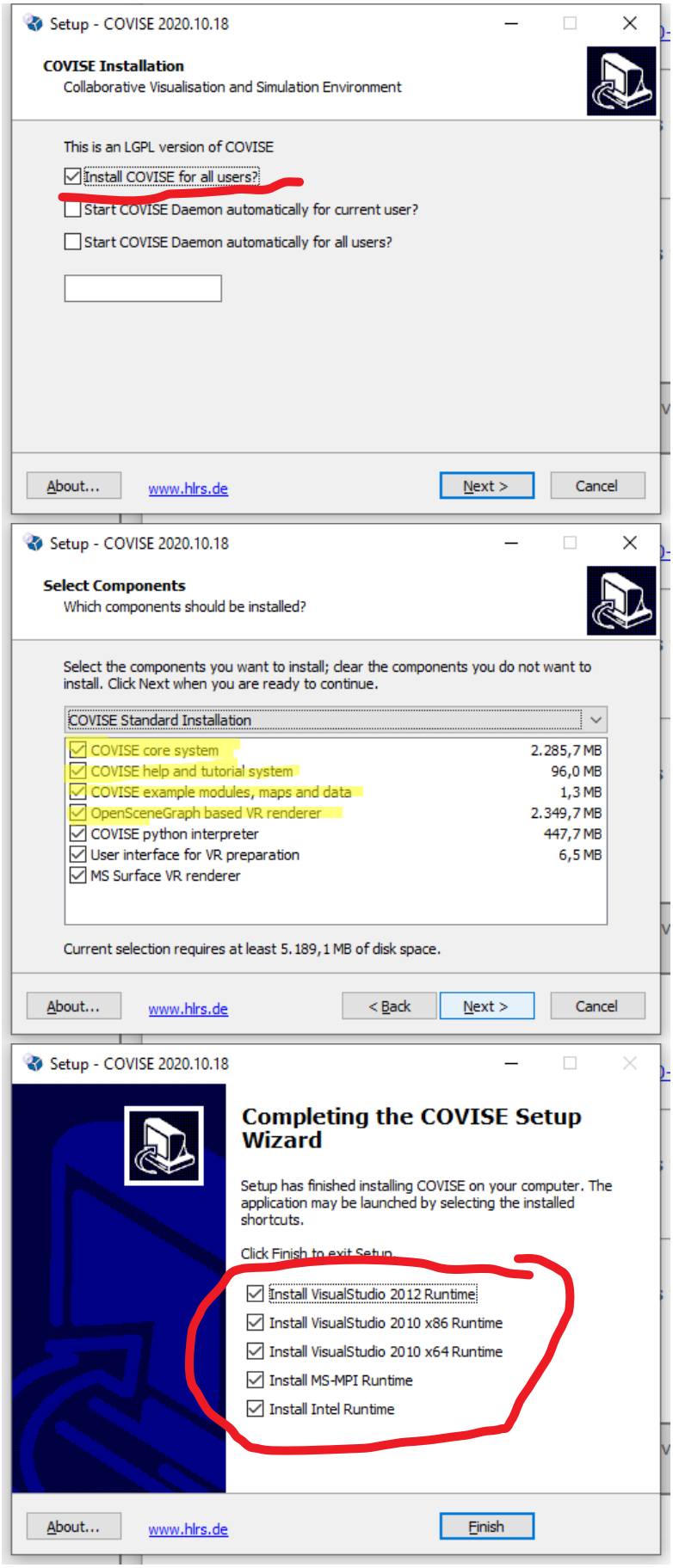
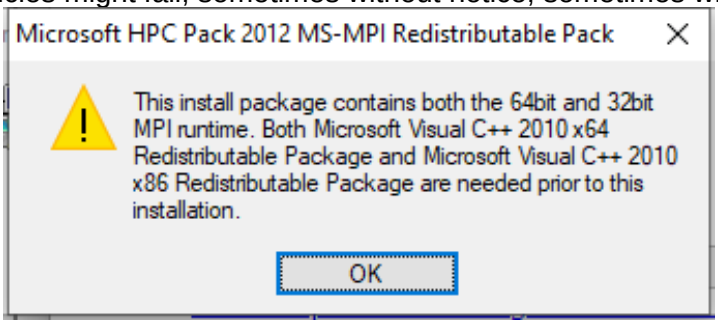


Figure 13 COVISE installation process

Again, you can accept default settings and click “Next” to follow through with the installation. Please install all five dependencies.

Remember to confirm all dependency installations.

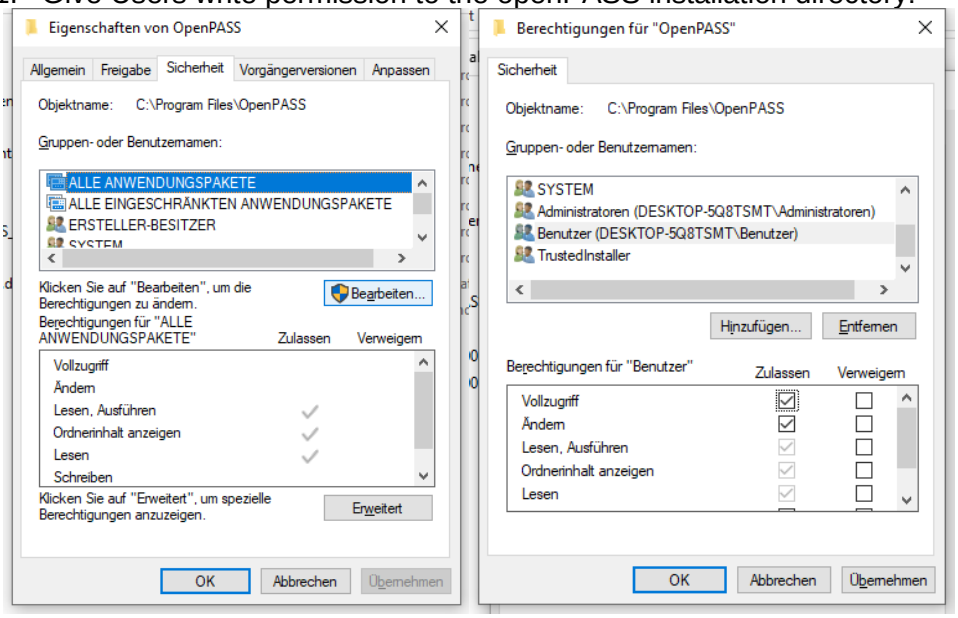
Depending on the state of your Windows installation, automatic installation of some of these dependencies might fail, sometimes without notice, sometimes with an error message such as the following:



In such a case: please navigate to “C:\Program Files\COVISE\zebuopt\lib\bin” and install the missing dependencies manually. We are trying to fix this situation for future releases.

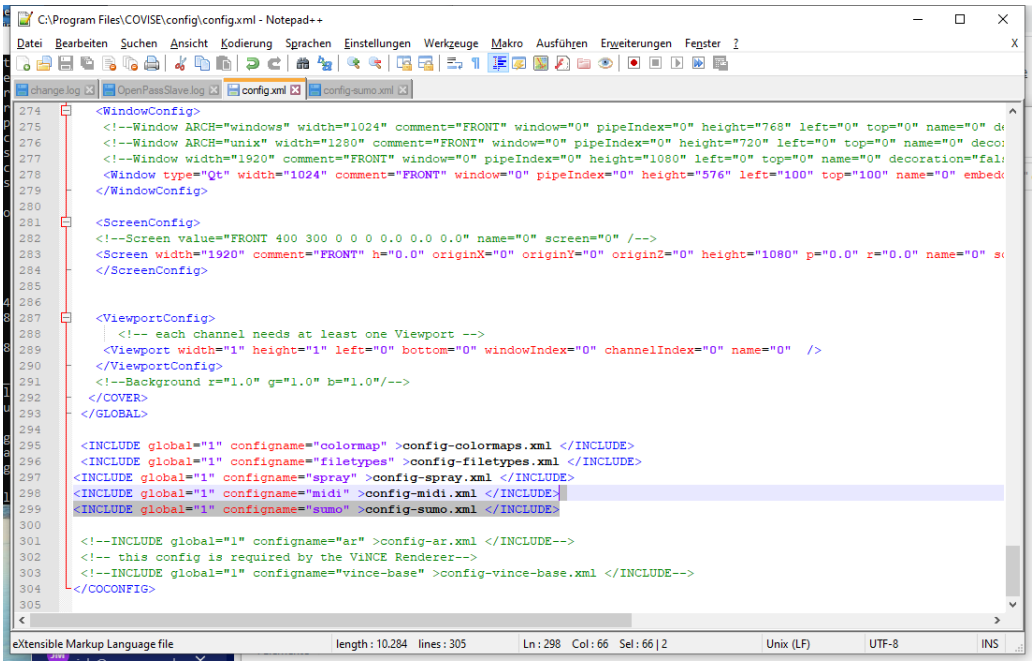
For the examples to work, you have to do three more manual configurations after the installation.

1. Give Users write permission to the openPASS installation directory:



Right-click on C:\Program Files\openPASS and select the security tab. There click on modify, confirm the modification, check all permissions for users of your computer and apply these changes.

2. Download and install sample data sets in OpenPASS.zip .zip from the HLRS OpenPASS download page: Please extract this file to C:\data
3. Add the line
   “<INCLUDE global="1" configname="sumo" >config-sumo.xml </INCLUDE>”
   to “C:\Program Files\COVISE\config\config.xml”



if you copied the OpenPASS data to a different directory than c:\data, you have to modify config-sumo.xml accordingly.

4.2 Exemplary test scenarios

After installing openPASS as described in the previous chapter, the simulation is ready to start: The simulation tools within OSCCAR provide all necessary configurations for motorway traffic in three different scenarios.

To generate motorway traffic, a pre-defined driver parametrization is provided. In previous projects, the driver parametrization was tested against real world traffic, especially against the highD dataset [7]. In addition, the driver can miss or forget information and has configurable reaction times. Thereby, the OSCCAR driver parametrization generates traffic and, at the same time, accidents occur naturally. Therefore, inducing critical situations by a specific scenario (defined by an OpenSCENARIO configuration) is possible but not necessary.

The OSCCAR demonstrator provides simulation configurations for three different scenarios, see Figure 14:

a) A motorway with three lanes and a speed limit,

b) An onramp at a motorway with two lanes,

c) A traffic jam on a motorway with three lanes.

As a specific openPASS simulation is defined by an OpenDrive road (see Chapter 3.2; .xodr-file) and an OpenSCENARIO scenario (see Chapter 3.3; .xosc-file), all three scenarios differ fundamentally. While scenarios 1 and 3 have simple .xodr-road environments (a straight three-lane motorway), scenario 2 has a more complex .xodr-road including an onramp. In addition, scenarios 1 and 2 have only simple .xosc-scenarios (defining only the end condition) and scenario 3 has a more complex .xosc-scenario including several scenario agents slowing down to generate a traffic jam.

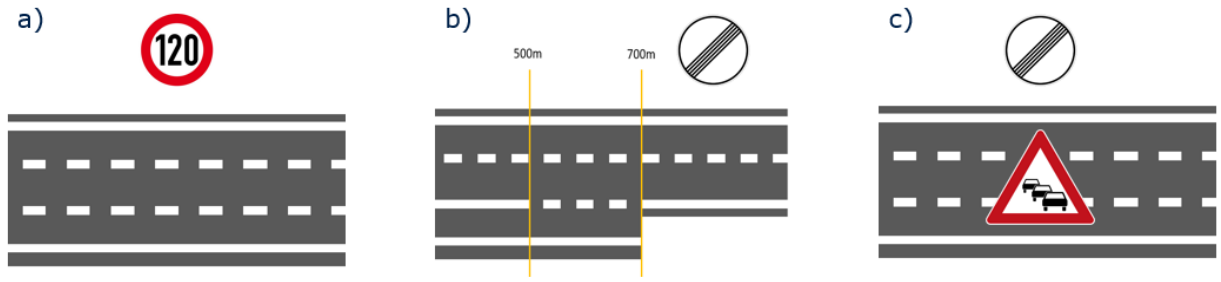


Figure 14. OSCCAR Demonstrator scenarios

The openPASS simulation starts by executing OpenPassMaster.exe, automatically distributing the simulation tasks on available CPUs. After finishing the simulations, each simulation has a simulationOutput.xml with all output information on the simulation, agents, and events. Information on collisions is stored as events in the simulationOutput.xml and, among other information, provides the time of collision, involved agents, collision velocities and collision angles (cf. OSCCAR deliverable D1.1). Upon activation, a file containing all trajectory information of all agents in the individual simulation runs is available in .csv-format, too. Analysing traffic characteristics is possible by using this trajectory .csv-file.

4.3 GUI workflow

Simulations can now be fully prepared through a graphical workflow. To work with one of the provided OSCCAR examples, the traffic jam in this case, please copy all configuration files from “C:\Program Files\OpenPASS\openPASS_Resource\Osccar_UseCase_traffic_jam*” to the main openPASS Directory “C:\Program Files\OpenPASS”. The main openPASS directory will be omitted in the following guide for simplicity reasons.

You can then start OddLot (see Figure 15) from the windows Start menu in the COVISE folder. After opening “configs\SceneryConfiguration.xodr” you will see the highway in the Road Link editor. In the Track editor, you can modify the street center line, add or modify lanes in the lane editor, or add street signs in the Signals and Objects editor.

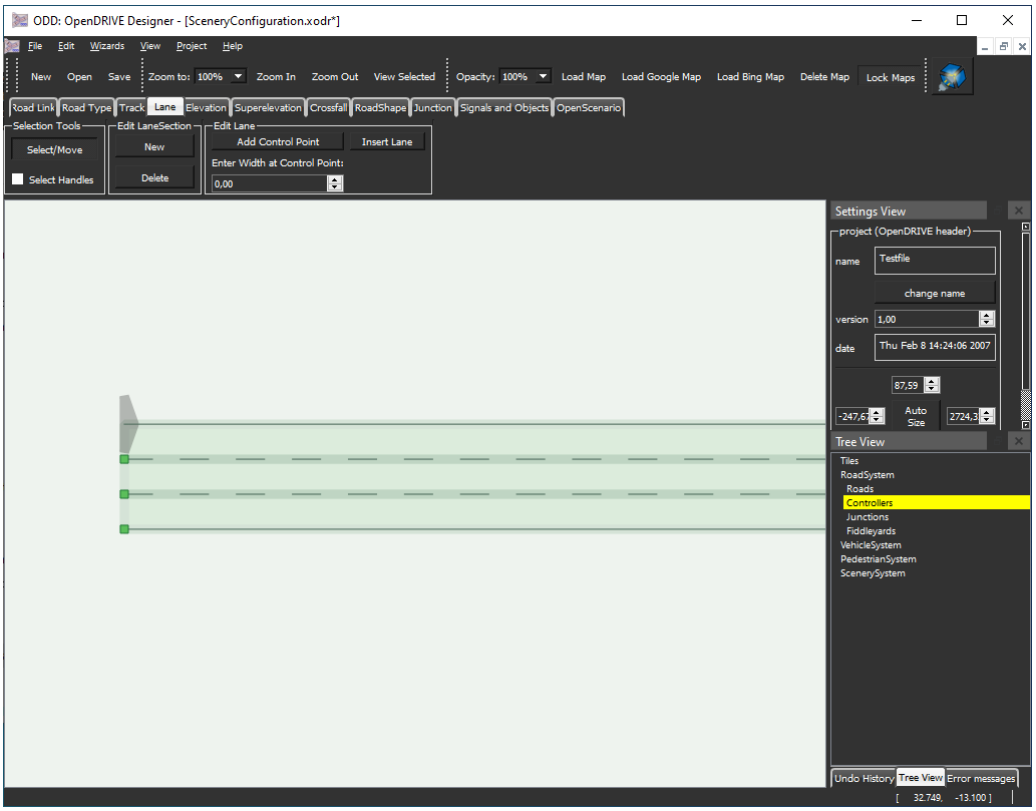


Figure 15. OddLot GUI

After creating or modifying the street network in OddLot, you can define the Scenario and simulation boundary conditions in the openPASS GUI (see Figure 16). The openPASS GUI can be started from the windows Start menu or a desktop icon. On the left panel select “Scenario-based Simulation” and Load “slaveConfig.xml” from the top menu bar.

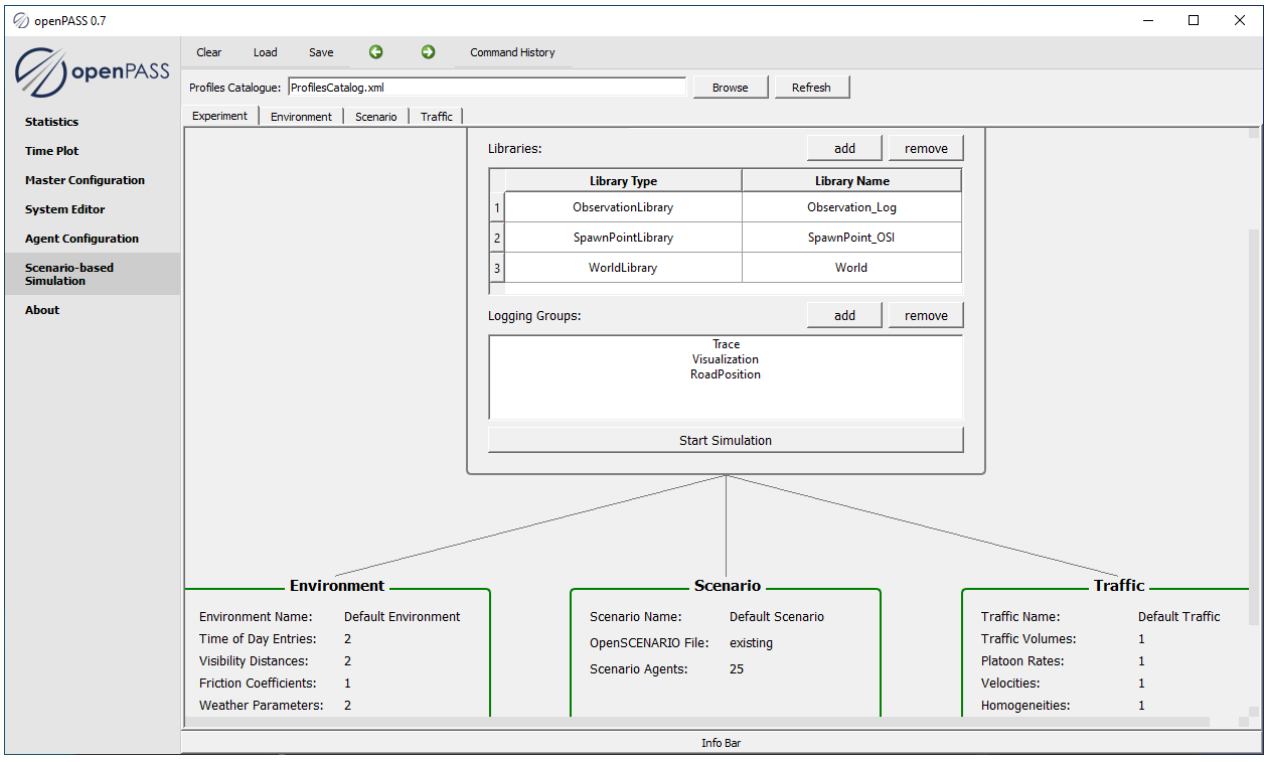


Figure 16. OpenPASS GUI

This will load the environment, scenario and traffic settings and allow you to run a simulation.

Once a simulation is completely defined, you can either run the simulation from within the openPASS GUI, in parallel from OpenPASSMaster or interactively in OpenCOVER.

In the Start menu you will find a “Covise Shell” where you can navigate to “C:\Program Files\OpenPASS”. By starting “opencover opencover://plugin/OpenPASS” you can load the OpenPASS simulation and watch its progress in 3D (see Figure 17).

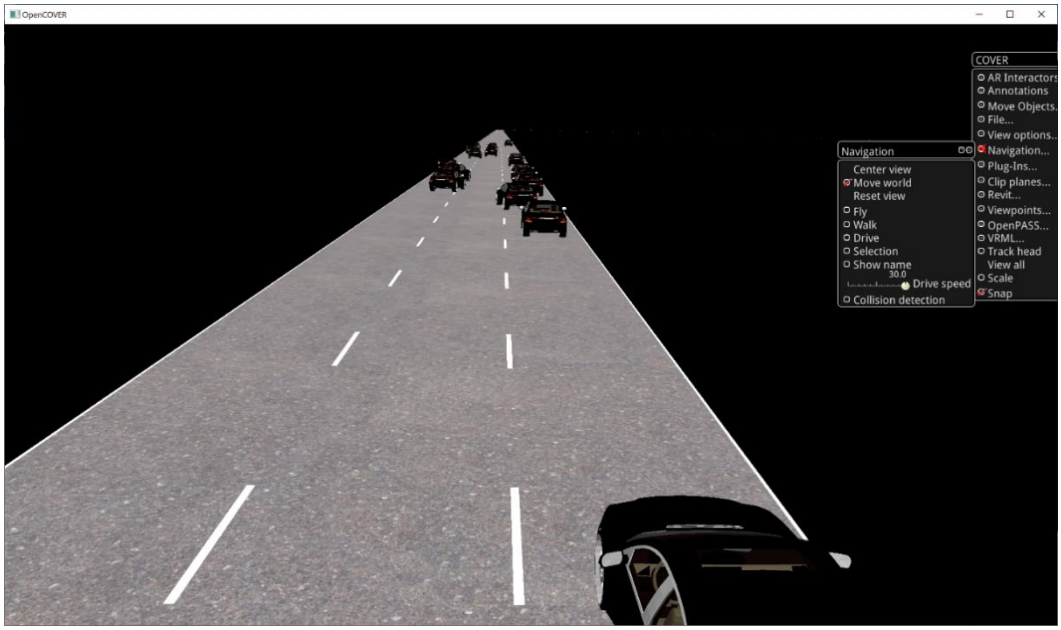


Figure 17. Traffic jam simulation in OpenCOVER

5 DISSEMINATION AND STANDARDISATION

5.1 Dissemination

The traffic simulation use case implemented with this demonstrator was aligned and discussed within the openPASS WG, comprising the following steps:

- Evaluation of OpenDrive / OpenSCENARIO functionalities of the current state of openPASS (mid 2019: openPASS minor release v0.6)
- Requirements towards applicable driver model and framework features (impact model, logging) as well as contributions to a new Cmake build system.
- Release planning considering what features are part of the “master” branch (see impact model: part of v0.7) and which COVISE/OSCCAR specific content is developed on the “hlrs” branch
- Integration of openPASS state on “hlrs” branch with the COVISE framework

The work towards this deliverable incorporates further development of open-source software, hence HLRS of USTUTT contributed to the open-source projects sim@openPASS and COVISE within the scope of OSCCAR under the respective licenses.

5.2 Standardization

This demonstrator enhances the role of openPASS and COVISE as reference implementations of OpenDrive and OpenSCENARIO and will support the further development of these standards and their goals of harmonizing the file formats used for describing scenarios in traffic simulation environments.

6 CONLUSION

D1.2 (demonstrator) comprises the openPASS framework for integrated safety assessment needed in OSCCAR. Requirements towards the traffic simulation environment were aligned with the openPASS open-source development during the project and exemplary use cases were defined. D1.2 includes the openPASS simulation platform and tools for pre-processing (e. g. changing road or scenario parameters) and post-processing (e. g. 3D visualisation of openPASS traffic). All code is open-source (licensed EPL or LGPL) and thus available for anyone to use both in research but also commercial development. Binary builds are available on the HLRS website. All executables are code-signed by now which should soon eliminate the security warnings during installation as described in this document. Within OSCCAR, the toolchain developed here was used throughout the rest of WP1 and results from this then fed into the other workpackages as initial- and boundary conditions for detailed crash and HBM simulations.

The configuration examples are aligned to the ongoing simulation studies for D1.3: regular lane motorway, motorway with on ramp, traffic jam. The methodology for using openPASS as an accident prediction tool as well as validation and input data will be documented in D1.3.

A. REFERENCES

[1] OSCCAR Deliverable D1.1 Accident data analysis - remaining accidents and crash configurations of automated vehicles in mixed traffic, v1.0, 2019-05-09

[2] COVISE Source code https://github.com/hlrs-vis/covise (last accessed: November 2020)

[3] OddLot https://www.hlrs.de/solutions-services/service-portfolio/visualization/driving-simulator/oddlot/ (last accessed: November 2020)

[4] OpenDRIVE https://www.asam.net/standards/detail/opendrive/ (last accessed: November 2020)

[5] OpenSCENARIO 1.x https://www.asam.net/project-detail/asam-openscenario-v1x/ (last accessed: November 2020)

[6] OpenSCENARIO 2.0 https://www.asam.net/project-detail/asam-openscenario-v20/ (last accessed: November 2020)

[7] Krajewski, Robert and Bock, Julian and Kloeker, Laurent and Eckstein, Lutz; The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems; 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp 2118-2125.

[8] Eclipse OpenPASS working group wiki https://wiki.eclipse.org/OpenPASS-WG; “Modular driver” presentations by Konstantin Blenz, TU Dresden: “Modular driver architecture and demands on the framework” https://wiki.eclipse.org/images/a/a9/20190704_AC_ModularDriverArchitecture_TUD.pdf; https://wiki.eclipse.org/images/a/ae/20191204_openPASS_ModularDriverArchitectur.pdf

[9] Fahrenkrog, F. et al., "Prospective Effectiveness Safety Assessment Of Automated Driving Functions – From The Method To The Results", 26th ESV, 19-0166 (2019)

[10] Wagström, L. et al., "Integrated safety: establishing links for a comprehensive virtual tool chain", 26th ESV, 19-0177 (2019)

B. ABBREVIATIONS AND DEFINITIONS

Term	Definition
AD	Automated Driving
ADAS	Advanced Driver Assistance System
ASAM	Association for Standardization of Automation and Measuring Systems
DEM	Digital Elevation Model
DSL	Domain Specific Language
FE	Finite Element
WG	Working Group
VR	Virtual Reality
XML	eXtensible Markup Language