Article

Design and Implementation of a Co-Simulation Framework for Testing of Automated Driving Systems

Demin Nalic[1,†]*, Aleksa Pandurevic [2,†], Arno Eichberger[3,†] and Branko Rogic[4,‡]

1 Institute of Automotive Engineering, TU Graz; demin.nalic@tugraz.at
2 Institute of Automotive Engineering, TU Graz; pandurevic@tugraz.at
3 Institute of Automotive Engineering, TU Graz; arno.eichberger@tugraz.at
4 MAGNA Steyr Fahrzeugtechnik AG Co & KG; branko.rogic@magna.com
† Current address: Inffeldgasse 11/2, 8010 Graz, Austria
‡ Current address: Liebenauer Hauptstraße 317, 8041 Graz, Austria

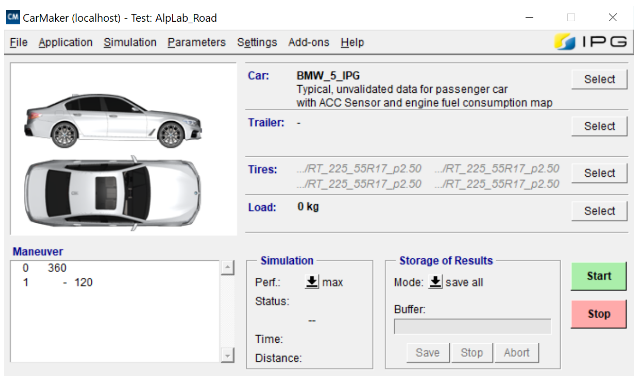
Abstract: The increasingly used approach of combining different simulation software for testing of automated driving systems (ADS) increases the need for potential and convenient software designs. Recently developed co-simulation platforms (CSP) provide the possibility to cover the high demand on testing kilometres for ADS by combining vehicle simulation with traffic flow simulation software (TFSS) environments. Having chosen a suitable CSP rises up the question how the test procedures should be defined and constructed and what are the relevant test scenarios. Parameters of the ADS in vehicle simulation, traffic parameter in TFSS and combination of all these can be used for the definition of test scenarios. Thus the automation of a process, consisting of vehicle and traffic parameters and a suitable CSP, a test procedure for ADS should be well designed and implemented. This paper presents the design and implementation of a complex co-simulation framework for virtual ADS testing combining IPG CarMaker and PTV Vissim.

Keywords: ADAS Simulation; Scenario Generation; Automated Driving; Testing

1. Introduction

Using suitable virtual platforms for generation of test scenarios for ADS is an essential part of the development process of automated driving, [1] [2] and [3]. The high demand on testing kilometres ([4]), the complexity of test procedures ([5], [6]) for Level 3+ ([7]) ADS in real world testing is time and resource consuming. Besides costs, many interesting traffic scenarios involve critical situations. Having a configurable model vehicle used for Model-in-the-Loop testing introduces a lot of advantages compared to the physical test drives. It provides automated, efficient and extensive testing tool with no risk and low costs regarding necessary equipment. In order to create such an environment, the approach of using two components, one for modelling of the ego vehicle and ADS and another for modelling the traffic. For both components, the widely used software in automotive engineering community, IPG CarMaker for modelling ego vehicle and PTV Vissim for microscopic traffic flow simulation, [8], was used. CarMaker, which can be seen in Fig. 1, provides a virtual multi-body simulation vehicle model which is a computer-modelled representation of an vehicle, see [9].

PTV Vissim, which can be seen in 2, is used to create realistic and accurate traffic conditions for testing different traffic scenarios [8]. The traffic flow model in Vissim is calibrated by using measured data from an official test road in Austria [10] and presented in the work of [1]. The concept and the modelling part of co-simulation framework, the vehicle under test and the traffic environment is already presented in [1]. The main goal of this work is the developed and design of the presented co-simulation framework and its maximal utilization of the co-simulation between CarMaker and PTV Vissim and the generation of simulation data for extraction of concrete scenarios (e.g. [11] and [12]).


File Application Simulation Parameters Settings Add-ons Help
IPG
Select
Select
Select
Select
Start
Stop
Save Stop Abort

CarMaker (localhost) - Test: AlpLab_Road

Car: BMW_5_IPG
Typical, unvalidated data for passenger car
with ACC Sensor and engine fuel consumption map

Trailer: -

Tires: ../RT_225_55R17_p2.50 ../RT_225_55R17_p2.50
../RT_225_55R17_p2.50 ../RT_225_55R17_p2.50

Load: 0 kg

Maneuver
0 360
1 - 120

Simulation
Perf.: max
Status: --
Time:
Distance:

Storage of Results
Mode: save all
Buffer:

Figure 1. User interface of CarMaker from where vehicle parameters can be specified.

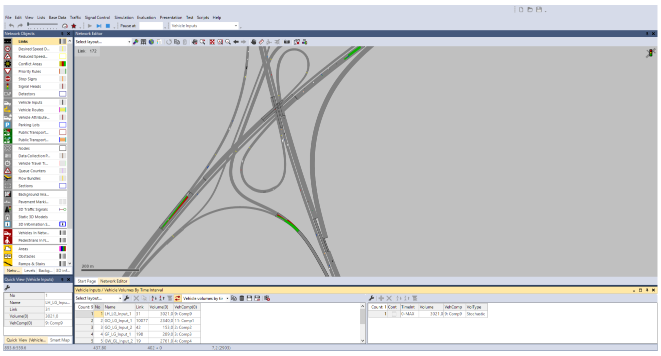


Figure 2. User interface of PTV Vissim.

2. Co-Simulation Framework

The high level overview of the co-simulation framework (CSF) can be seen as a group of three components interacting with each other, as represented in Figure 3 In a broad sense, this diagram represents a rough overview of the architecture and since CarMaker and Vissim are existing components, remaining tasks are the development of the co-simulation controller (CSC) and communication between CSC, CarMaker and Vissim. Communication between Vissim and CarMaker is handled through the Vissim Interface for CarMaker, see [13]. Vissim provides an useful COM interface through which the communication between CSC and Vissim is handled, see [14] and [15]. Communication between CSC and CarMaker is done in two ways, by sending *Tcl* commands through the *cmguicmd* interface and setting the Simulink model parameters pragmatically. Considering the whole framework, the most development is done regarding CSC. Using object-oriented concepts, data and the behaviour related to that data are coupled together. Another goal of introducing object-oriented concepts was a separation of logic from the user interface related code.

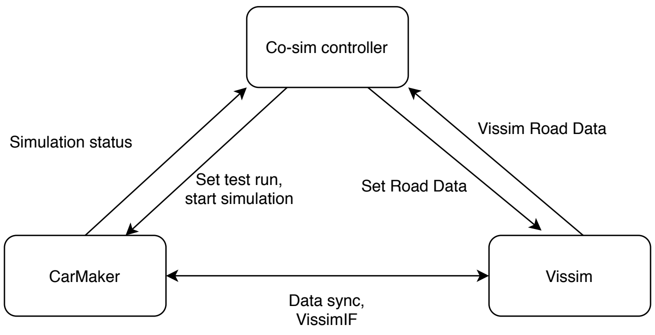


Figure 3. A high-level overview of the software architecture presenting the three components of the co-simulation framework.

Requirement 1	Simulation
Requirement 2	Data Storage
Requirement 3	Road File
Requirement 4	Simulation Control
Requirement 5	Input Validation

Table 1. Functional requirements on the co-simulation framework.

3. Software Design

Software Design is the essential part of this work. Here, CSC is broken into the multiple components where each is described with at least one class without violating any of the functional requirements.

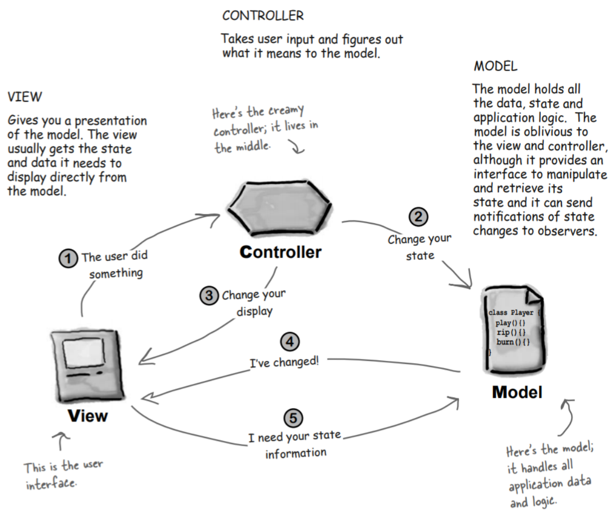

CONTROLLER
Takes user input and figures out what it means to the model.
MODEL
The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.
VIEW
Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.
Here's the creamy controller; it lives in the middle.
Controller
1 The user did something
2 Change your state
3 Change your display
4 I've changed!
5 I need your state information
View
Model
class Player play(){} rip(){} burn(){}
This is the user interface.
Here's the model; it handles all application data and logic.

Figure 4. Representation between components of MVC (image taken from [16]).

3.1. Defining Functional Requirements

The functional requirements for the software design are shown in Tab. 1 The main functional requirements from Tab. 1 is the first one. It is divided in additional parts.

First Part: Simulation of one specific scenario over many kilometers

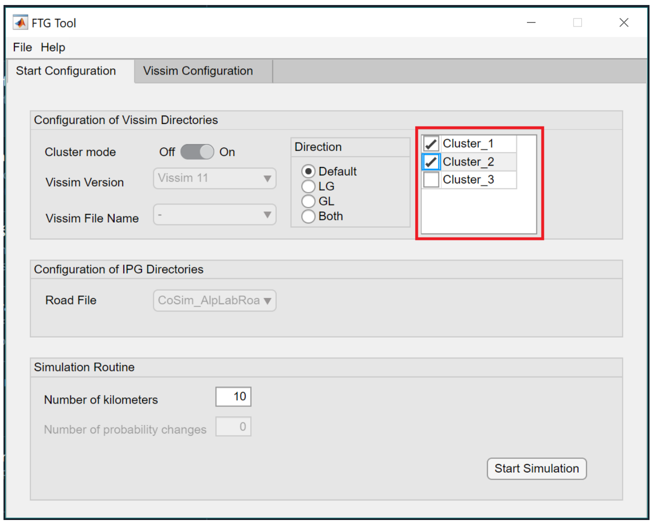
The co-simulation framework provides the possibility to choose the test road, the Vissim version, the testing kilometers and the specific scenario testrun for scenario generation purposes. In CarMaker a scenario testrun is a specified simulation run which contains the vehicle under test with all functionalities, a defined test road, environment objects and test definition. All given parameters shall be checked by the system for validity before starting the simulation. A scenario testrun is testrun defined with a configured vehicle model and traffic simulation.

Second Part: Simulation of multiple scenarios over many kilometers

The co-simulation framework provides the possibility to choose the root folder of containing scenarios and a selection of those which should be be simulated. Each scenario testrun is simulated over a given number of kilometers. All given parameters are checked by the system for validity before starting the simulation.

3.2. Design

For the design of the co-simulation framework a GUI is implemented using a model-view-control (MVC) pattern ([17]). Applying MVC pattern, elements of view component are decoupled from the framework logic and the rest of the code is broken into smaller yet meaningful components which


FTG Tool

File Help

Start Configuration | Vissim Configuration

Configuration of Vissim Directories

Cluster mode: Off [✓ On]

Vissim Version: Vissim 11

Vissim File Name: -

Direction: [✓ Default] LG GL Both

- ✓ Cluster_1
- ✓ Cluster_2
- Cluster_3

Configuration of IPG Directories

Road File: CoSim_AlpLabRoa

Simulation Routine

Number of kilometers: 10

Number of probability changes: 0

Start Simulation

Figure 5. Selecting clusters loaded from the cluster's root path in GUI.

interact with each other. For implementing MVC Design Pattern guidelines from [16] were used. The interaction between components can be seen in Figure 4. In co-simulation framework these three components can be distinguished as *View*, *Controller* and *Model*. The *View* component is *singleton View* class which describes the appearance of the user interface. All user interface elements are defined there. The *Controller* component is a *singleton Controller* class which acts as an interface between View and Model components. It updates the state of the model component, handles return values of the model functions and based on them generate adequate graphical response using view elements. Compared to the previous two components, the *Model* component is not a single class. There is a *singleton Model* class but it is just the main part of the model component. The *Model* component is composed of the additional classes which are responsible for the framework logic and for holding all necessary data for the simulation process. The control logic of framework is located in *StartSimulation* function. It differentiates between two different modes of operation, the manual and the cluster mode. The model class is also responsible for the inter-process communication with PTV Vissim and IPG CarMaker. When the *StartSimulation* function is called, model configures both Vissim and CarMaker and executes simulations.

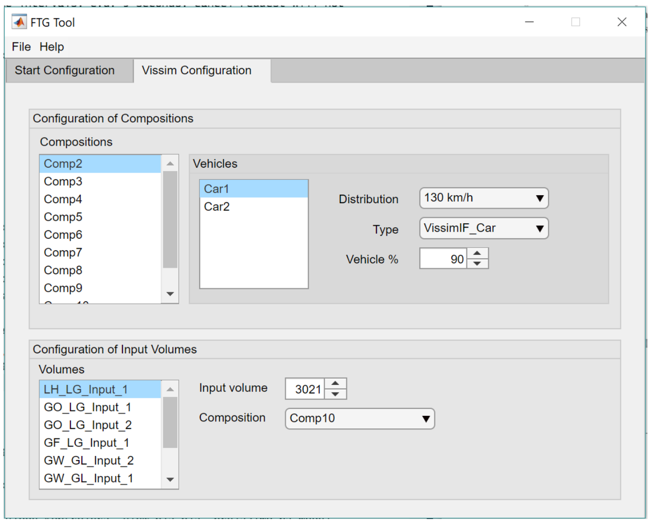
4. Application

From functional requirements, two separate modes of simulations can be distinguished, manual and automatic or cluster mode, see [1]. In manual mode, the user is allowed to generate a very specific scenario where he can vary different parameters, such as distributions, compositions of vehicles and traffic in the the tool. Automatic or cluster mode is meant for exhaustive fully automated testing of a model over many kilometers.

4.1. Cluster mode

Automatic or cluster mode of the co-simulation is a simulation of already predefined scenarios created using real traffic measurements of some specific road, see [1]. With this data provided, the user wants to observe the behavior of the modeled vehicle over varying driving distances. In this mode, the user can choose the path to the root folder where clusters containing predefined scenarios are located. All clusters found in the selected root folder are displayed to the user in GUI and made available for the selection as presented in Figure 5.

Execution of cluster mode can be separated into two phases, initialization phase and simulation phase. Initialization phase is composed of:



Configuration of Compositions

Compositions: Comp2, Comp3, Comp4, Comp5, Comp6, Comp7, Comp8, Comp9

Vehicles: Car1, Car2

Distribution	130 km/h
Type	VissimIF_Car
Vehicle %	90

Configuration of Input Volumes

Volumes: LH_LG_Input_1, GO_LG_Input_1, GO_LG_Input_2, GF_LG_Input_1, GW_GL_Input_2, GW_GL_Input_1

Input volume	3021
Composition	Comp10

Figure 6. Vissim configuration tab.

1. **Loading CarMaker testrun file**
   File with specified direction and Vissim road file is loaded into the CarMaker.
2. **Loading Vissim road file**
   Road from the selected cluster is loaded into the Vissim and it is modified by changing the random seed value. The random seed value changes the traffic procedure without changing the parametrisation. This value represents the stochastic nature of the road traffic.
3. **Setting the number of kilometers to be covered**
   Number of kilometers specified by the user is set in the Simulink model using *set_param* function.
4. **Starting simulation**
   Simulation is started by setting *SimulationCommand* parameter of the Simulink model to *start*. That is also done using *set_param* function.

From this point, *simulation phase* begins and therefore the status of the running simulation has to be checked. For those purposes, timers are used as described in section 4.3. If simulation is stopped, the covered kilometers are checked. If the simulation status reached a given number of kilometers, the CSC can proceed in three following ways:

1. Restart the simulation on the same road in opposite direction.
2. Switch to the following cluster.
3. Finish if no more clusters are left.

During the simulation phase, the progress bar is presented to the user which has three purposes; the first one is that the user aware of the running simulation status. The second purpose is the possibility of cancelling the simulation via the stop button, which is part of the progress bar. Displayed progress bar deactivates the whole user interface so that no changes of the simulation settings can be made until the stop button is pressed, and that is its third purpose. The obvious advantage of this mode is the automation of a process which manually would consume a lot more time. All the data generated by Vissim and CarMaker are stored in such a way, so they can be accessed by date, mode or creation time stamp. Post-processing is done at the end, after data for all clusters are acquired. With all generated data, the configuration file is also stored in which the configuration for the latest simulation run can be found, including cluster root folder path, selected clusters, number of kilometers per cluster and random seeds used for every simulation step.

4.2. Manual mode

The main difference between the Manual and Cluster mode is that in manual one user does not use predefined clusters for the simulation, but rather individual Vissim road files which can be easily modified over the GUI. All the data related to the chosen Vissim road file are loaded into the internal data structure using COM interface and displayed in the separate tab in the GUI as presented in Figure 6. Vissim related data is held by the *VissimData* class which when initialized, reads the data from Vissim and has the responsibility of filling that data structure. The process of loading the data is done in the *VissimData* constructor which is called upon the selection of the Vissim file. All operations on the data are handled by *VissimData* class. Having this class provides convenient abstraction of Vissim data. It also enables having multiple Vissim files loaded into the memory. Switching between them is a lot faster than loading all the data all over again. When all the data from Vissim is retrieved, the data is placed into the structures. Lists of compositions and volumes are created, where both lists are contained within *VissimData* class. Each volume has a reference to the corresponding composition, so it can keep the track of the current composition which is assigned to that particular volume. From this point, the CSF is ready for the simulation run, and again, code execution after pressing Start Simulation button is separated into two phases, initialization and simulation phase.

At the beginning of the *initialization phase*, all changes of compositions and volumes are sent back to Vissim which is also the main difference between cluster and manual mode regarding this phase. Changes which user is able to perform are following:

1. Change vehicle input volume value
2. Change vehicle input composition
3. Modify composition by modifying vehicles of that composition
4. Change the speed distribution of selected vehicle
5. Change the type of selected vehicle
6. Change the vehicle percentage within selected composition

Rest of the initialization phase of manual mode is similar to the one in the cluster mode.

In the *simulation phase* the main difference to the cluster mode happens when all given kilometers are covered. Simulation does not continue with the different road file since there are no clusters to follow. *The only case when the simulation continues after a selected number of kilometers is when the direction option is set to both*.

All performed changes are immediately reflected into the internal data structure which is upon simulation start loaded into the Vissim road file using COM interface. All the data generated by Vissim and CarMaker are stored in the same way as described in Cluster Mode.

4.3. Simulation status

Since MATLAB lacks conventional multi-threading capabilities, timers are used as a workaround for making the GUI responsive while the simulation is running. After the simulation is initialized, the timer is started. It schedules the execution of some MATLAB code, in this case, code which checks for the current simulation status. It checks whether simulation is stopped or still running. In the case that simulation is stopped, the number of covered kilometers is checked and it is decided if more kilometers have to be simulated on this configuration or not. The rest of the simulation flow is mode specific.

Since scheduling of the code which checks simulation status is not time-critical, the *fixedSpacing* mode for timer is chosen. This mode avoids any possibility of interrupting the execution of a function during its execution. No new execution is scheduled until the function is executed, and that plays out as an advantage, since the goal is to avoid frequent checks of the status which is time-consuming and interrupts the simulation. It is also not desirable to fill a queue with scheduled function calls since the simulation can be stopped at any time. As a result of using timers, the CSF has responsive GUI allowing the user to stop the simulation, and even more important, it enables the checking of simulation status efficiently, since there is no reliable option for triggering events upon simulation end, and repeatedly checking for status with delays in between keeps the system busy when not needed.

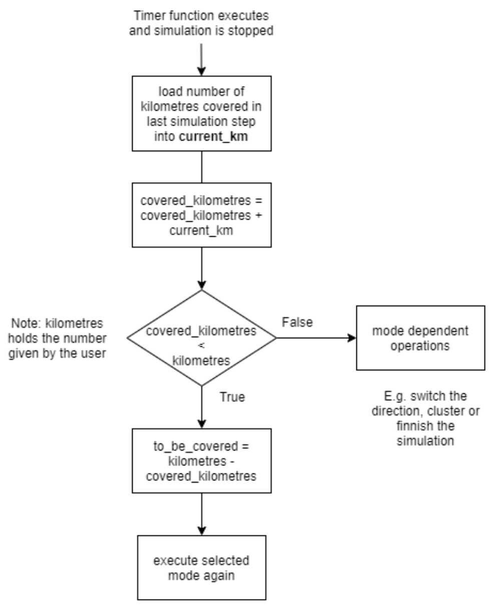

Timer function executes and simulation is stopped
load number of kilometres covered in last simulation step into current_km
covered_kilometres = covered_kilometres + current_km
Note: kilometres holds the number given by the user
covered_kilometres < kilometres
False
mode dependent operations
E.g. switch the direction, cluster or finnish the simulation
True
to_be_covered = kilometres - covered_kilometres
execute selected mode again

Figure 7. Control flow diagram showing the calculation of remaining kilometers.

Method	Context
cmguicmd	Setting distance to be covered
set_param	Starting and stopping simulation

Table 2. Different approaches of interacting with CarMaker.

4.4. Covered kilometers

Checking the covered kilometers is a common mechanism for both modes. It enables the simulation to run for any given length even though the road file used is limited to some length. As it can be seen in the simulation status subsection, the covered kilometers are checked periodically in the function executed by the timer. Out of the number of currently covered kilometers, new number of remaining kilometers is calculated. The workflowof this procedure can be best seen in Figure 7. There is no way to capture a route length which will be driven, therefore a number of kilometers is given as the initial distance to be covered. If the remaining distance is shorter than the current route length, vehicle will drive the remaining number of kilometers and then stop. In case that the remaining distance is longer than the route, the simulation will stop when the end of the route is reached and covered distance will be saved in the work-space. That addition in Simulink is necessary in order to obtain precise number of so far covered kilometers by the model vehicle.

4.5. Communication with CarMaker

Communication with CarMaker is done in two ways, through the *cmguicmd* interface, provided by CarMaker in order to send *Tcl* commands [18] and through setting the Simulink model properties using the *set_param* function, see Tab. 2.

Besides these two options, certain parameters are set using additional scripts, such as driving direction in the simulation, path in which generated data will be stored and which Vissim road path should be applied. Since there is no clear interface for modifying any of these parameters, scripts which modify necessary configuration files are introduced. They are simple C++ scripts which match given attribute name and assign the new value to it.

Task 1	Setting the newly generated random seed value.
Task 2	Loading compositions, vehicles, vehicle distributions, vehicle types, vehicle percentages, vehicle input volumes.
Task 3	Setting modified compositions, volumes and vehicles.

Table 3. Task list for the interaction with Vissim.

4.6. Communication with Vissim

Interaction with PTV Vissim is completely executed by the COM interface [19]. It is used for the tasks described in Tab. 3. Before any of these operations is done, the creation of a new *actxserver* is needed which as an argument requires programmatic identifier (ProgID). It creates a COM server through which it is possible to communicate with Vissim. Depending on the Vissim version, different programmatic identifiers are passed. After starting *actxserver* and therefore starting Vissim, the desired road network has to be loaded in order to make any changes.

4.7. Direction selection

The direction in which the modeled vehicle will move is specified as an attribute in the testrun file. Since this is specific to the selected road, it is meant to be used only with predefined roads and clusters. Differences between the same testruns in different directions are not huge, therefore the scripts mentioned in 4.5 are used to set certain attributes to the specified value. Besides two directions in which simulation can take place, there is also possibility of choosing both directions. In this case, the simulation is running in both directions and it will cover the same specified number of kilometers in both of them.

4.8. Data Storage

The result of the successful simulation is generated data which is used for later post-processing and data extraction. For every simulation step, one **.erg* the file is created by CarMaker which contains data from various ego vehicle sensors. Besides *erg* files, one configuration file is created containing all parameters used for simulation.

Upon simulation start of either of two modes, the folder named by current date is created, e.g. 12-Feb-2020. Inside of it, another folder named by current mode is created, either Automatic or Manual. All the data created by simulations on that day will be stored in the same root folder named by that date with the exception of the user changing the data output path. Assuming that user is not changing data output path, all automatic mode simulations would be stored in *../12-Feb-2020/automatic/* and all manual mode simulations would be stored in *../12-Feb-2020/manual/*. To avoid mixing all the data, every simulation run gets its own folder named by the time of simulation start, e.g. 14-30-24 in which the erg files and the configuration file is placed. All erg files and corresponding info files are simply named by the number which defines the order of its creation.

5. Conclusion

In paper we presented the design and implementation of the virtual test co-simulation framework for testing of ADS using MATLAB/Simulink, IPG CarMaker and PTV Vissim. As described in 3, all functional requirements are defined and implemented. The implemented co-simulation framework is capable of generating data for scenario generation for ADS. Since the framework highly relies on CarMaker and Vissim which allows a bride usage for testing automotive systems. Using CarMaker for developing procedures allows the implementation of ADS algorithms, sensors and a wide number of automotive systems (e.g. powertrain, engines, transmission etc.) and Vissim as traffic flow simulator provides the opportunity for testing automotive system in dynamic and realistic traffic environments.

Author Contributions: Conceptualization, D.N.; methodology, D.N. and A.P.; software, D.N. and A.P.; validation, A.E., B.R. and D.N.; formal analysis, D.N. and B.R.; investigation, D.N.; resources, D.N. and B.R.; data curation, D.N.; writing–original draft preparation, D.N. and A.P.; writing–review and editing, A.E. and B.R.; visualization, D.N.; supervision, A.E. and B.R.;All authors have read and agreed to the published version of the manuscript

Funding: This work is founded by the Austrian Federal Ministry of Transport, Innovation and Technology as part of the FFG Program "EFREtop"

Conflicts of Interest: The authors declare no conflict of interest.

1. Nalic, D.; Eichberger, A.; Hanzl, G.; Fellendorf, M.; Rogic, B. Development of a Co-Simulation Framework for Systematic Generation of Scenarios for Testing and Validation of Automated Driving Systems*. 2019 IEEE Intelligent Transportation Systems Conference (ITSC), 2019, pp. 1895–1901. doi:10.1109/ITSC.2019.8916839.
2. Hallerbach, S.; Xia, Y.; Eberle, U.; Köster, F. Simulation-Based Identification of Critical Scenarios for Cooperative and Automated Vehicles. *SAE Technical Papers* **2018**, *2018-01-1066*. doi:10.4271/2018-01-1066.
3. Aparow, V.R.; Choudary, A.; Kulandaivelu, G.; Webster, T.; Dauwels, J.; d. Boer, N. A Comprehensive Simulation Platform for Testing Autonomous Vehicles in 3D Virtual Environment. 2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR), 2019, pp. 115–119. doi:10.1109/ICMSR.2019.8835477.
4. Kalra, N.; Paddock, S. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* **2016**, *94*, 182–193. doi:10.1016/j.tra.2016.09.010.
5. Broggi, A.; Buzzoni, M.; Debattisti, S.; Grisleri, P.; Laghi, M.C.; Medici, P.; Versari, P. Extensive Tests of Autonomous Driving Technologies. *IEEE Transactions on Intelligent Transportation Systems* **2013**, *14*, 1403–1415. doi:10.1109/TITS.2013.2262331.
6. Stellet, J.E.; Zofka, M.R.; Schumacher, J.; Schamm, T.; Niewels, F.; Zöllner, J.M. Testing of Advanced Driver Assistance Towards Automated Driving: A Survey and Taxonomy on Existing Approaches and Open Questions. 2015 IEEE 18th International Conference on Intelligent Transportation Systems, 2015, pp. 1455–1462. doi:10.1109/ITSC.2015.236.
7. SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 2019.
8. Fellendorf, M.; Vortisch, P., Microscopic Traffc Flow Simulator VISSIM. In *Fundamentals of Traffic Simulation*, 1 ed.; Springer Science+Business Media, 2010; Vol. 145, *International Series in Operations Research & amp; Management Science*, pp. 63–94.
9. Varga, B.O.; Moldovanu, D.; Mariaşiu, F.; Iclodean, C.D. Simulation in the Loop of Electric Vehicles. In *Modeling and Simulation for Electric Vehicle Applications*; Fakhfakh, M.A., Ed.; IntechOpen: Rijeka, 2016; chapter 1.
10. Seebacher, S.; Datler, B.; Erhart, J.; Greiner, G.; Harrer, M.; Hrassnig, P.; Präsent, A.; Schwarzl, C.; Ullrich, M. Infrastructure data fusion for validation and future enhancements of autonomous vehicles' perception on Austrian motorways. 2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE), 2019, pp. 1–7. doi:10.1109/ICCVE45908.2019.8965142.
11. Weber, N.; Frerichs, D.; Eberle, U. A simulation-based, statistical approach for the derivation of concrete scenarios for the release of highly automated driving functions. AmE 2020 - Automotive meets Electronics; 11th GMM-Symposium, 2020, pp. 1–6.
12. Menzel, T.; Bagschik, G.; Maurer, M. Scenarios for Development, Test and Validation of Automated Vehicles. 2018 IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 1821–1827. doi:10.1109/IVS.2018.8500406.
13. Siebel, T. IPG | PTV | New Interface. *MTZ Worldwide* **2017**, pp. 36–39.
14. Tettamanti, T.; Horváth, M. A practical manual for Vissim COM programming in Matlab - 3rd edition for Vissim 9 and 10. 2018.

15. Joelianto, E.; Sutarto, H.; antariksa, s. Simulation of Traffic Control Using VissimCOM Interface. *Internetworking Indonesia Journal* **2019**, *11*, 55.
16. Freeman, E.; Freeman, E.; Bates, B.; Sierra, K. Head First Design Patterns **2004**.
17. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.M. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley Professional, 1994.
18. IPG. CarMaker Programmer's Guide, 2019.
19. PTV AG. PTV Vissim User Manual, 2018.