Simulation Software for Autonomous Driving

Yu Huang
yu.huang07@gmail.com
Sunnyvale, California

Outline

- Open Source of Simulators
- ADAS and Functions for Autonomous Driving
- Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing
- Simulation in development and testing of autonomous vehicles at Daimler
- Simulation Framework for Executing Component and Connector Models of Self-Driving Vehicles
- A multi-domain simulation approach to validate advanced driver assistance systems
- Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios
- Software architecture for an autonomous car simulation using ROS, MORSE & a QT based software for control and monitoring
- An Integrated Architecture for Autonomous Vehicles Simulation

Open Source of Simulators

- **Stage** is a 2D multi-robot simulator, developed within **Player/Stage** project.
  - Stage can simulate simplified robots in flat indoor environments.
  - **Player** can be also used with **Gazebo**, it is a 3D open-source multi-robot simulator.
- Gazebo was developed as an outdoors simulator.
  - Gazebo uses ODE (”Open Dynamics Engine”) as physics engine, also switched to **Bullet** (a SoA physics engine developed in part of Blender 3D), **Simbody** or to **DART** (”Dynamic Animation and Robotics Toolkit”).
  - Gazebo uses OpenGL as a 3D visualization engine.
  - Gazebo can accommodate around ten robots, whose bodies are constructed from boxes, spheres, cylinders, planes, and lines, that are joined with joints.
  - Gazebo is operated through GUI, where the user can see and edit simulation environment and robot model.

Open Source of Simulators

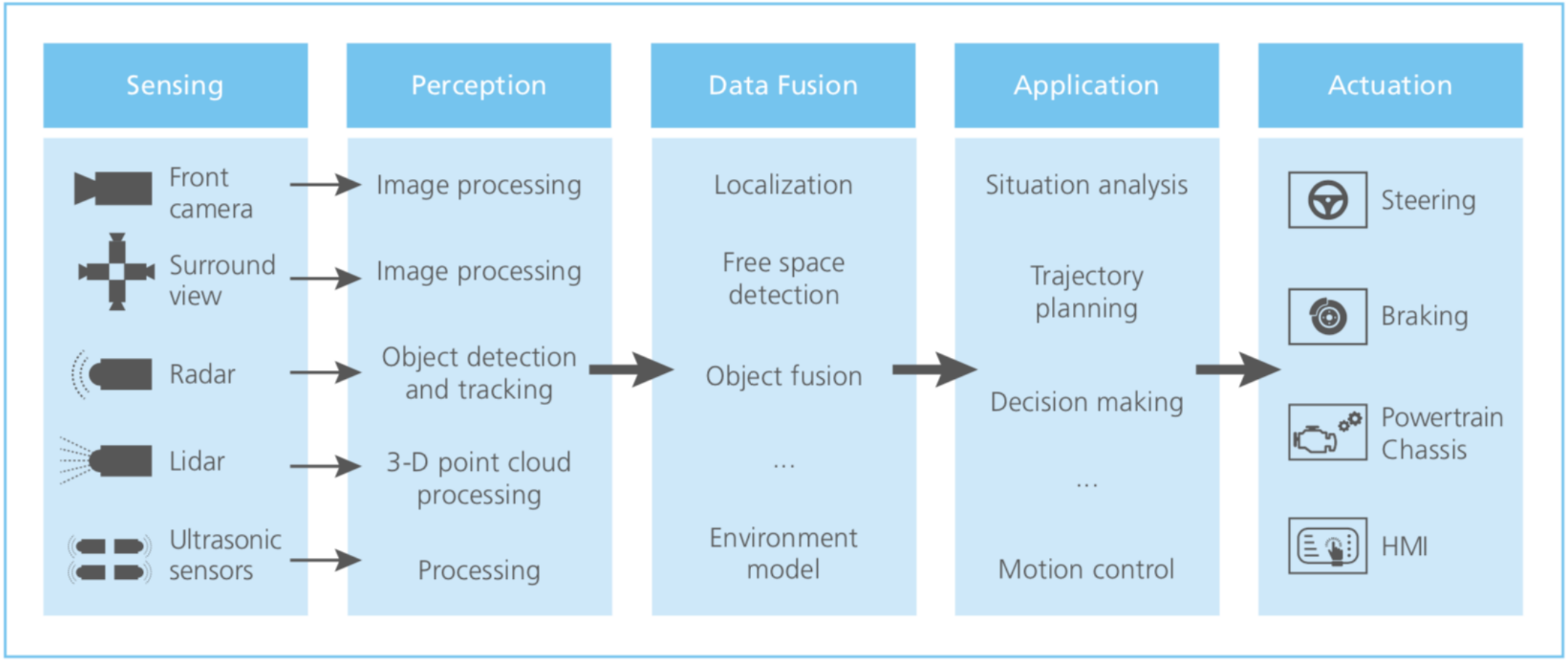
- While **V-REP** (”Virtual Robot Experimentation Platform”) is free to use for students and universities, it is not an open-source project, not grant 100% control over the simulation.
  - V-REP is created to be portable, flexible and versatile.
  - V-REP uses the ODE physics engine as Gazebo, Webots, but switched to Bullet or Vortex.
  - V-REP is operated from GUI, which is more user-friendly than Gazebo’s control.
  - Because V-REP uses its own graphic engine, it could run smoother on some machines, but it does not provide such photo-realistic visualization as MORSE.
- **Blender** is an open-source multi-platform application focused on creating 3D models, rendering, computer generated animation, post-production activities and creating interactive applications.
- **MORSE** (”Modular Open Robots Simulation Engine”) is based on Blender 3D software using its **Blender Game Engine** (BGE) for photo-realistic 3D visualization and **Bullet** for simulating physics.
  - MORSE does not need GUI and it is only operated from a command line.
  - MORSE provides the ability to select a degree of realism of the simulation, for example, the user can choose what data is measured by a laser rangefinder and whether data contain a signal noise.

ADAS and Functions for Autonomous Driving

- The **dSPACE** for autonomous driving is an end-to-end **tool chain** from a single source.
- **Rapid prototyping** solutions of high-performance platforms and a tailored SW environment allow for the development of complete multi-sensor applications in the vehicle, from perception and fusion algorithms to real-time controls.
- Increase in testing effort is handled by moving to **software-in-the-loop** (**SIL**) simulation.
- For release tests, the **hardware-in-the-loop** (**HIL**) simulation remains indispensable.
  - Integrating real environ. sensors such as camera, radar or Lidar and sensor fusion.
- dSPACE offers from restbus **simulation** and raw data feed to **OTA** stimulation.
- dSPACE supports SIL and HIL testing with realistic GPU-based sensor models and detailed real-time vehicle and environment simulations.
- The dSPACE tool chain ensures that **data** volumes generated is **recorded** in a time-correlated manner and **played back** synchronously and jitter-free in the test lab later.

ADAS and Functions for Autonomous Driving

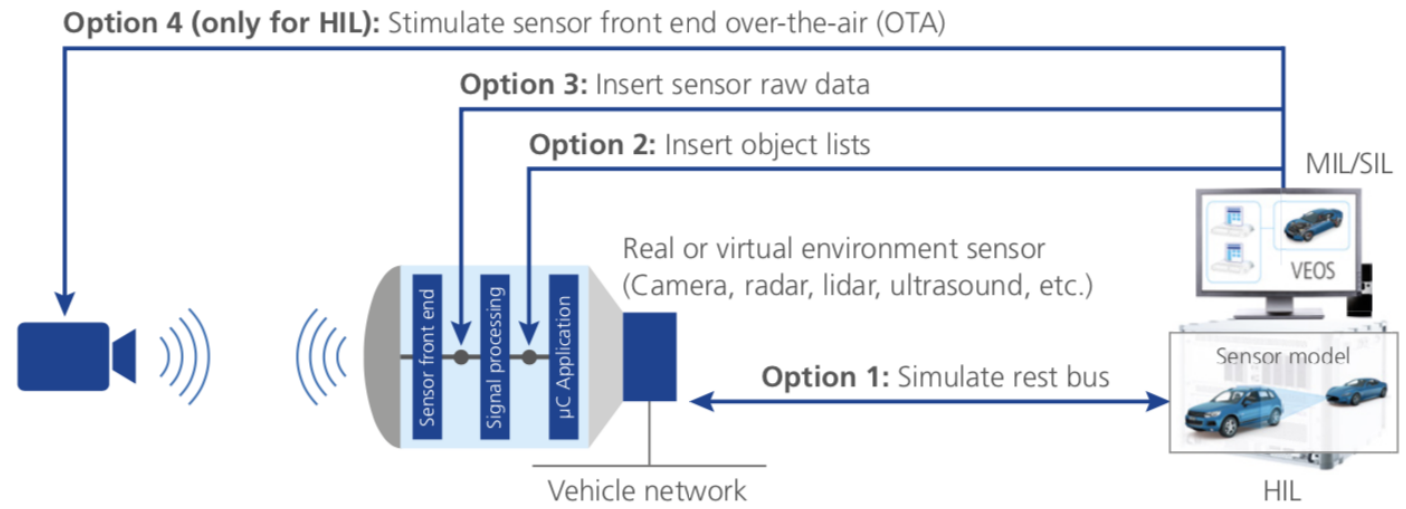
dSPACE



Chain of Effects in Autonomous Driving

ADAS and Functions for Autonomous Driving

dSPACE

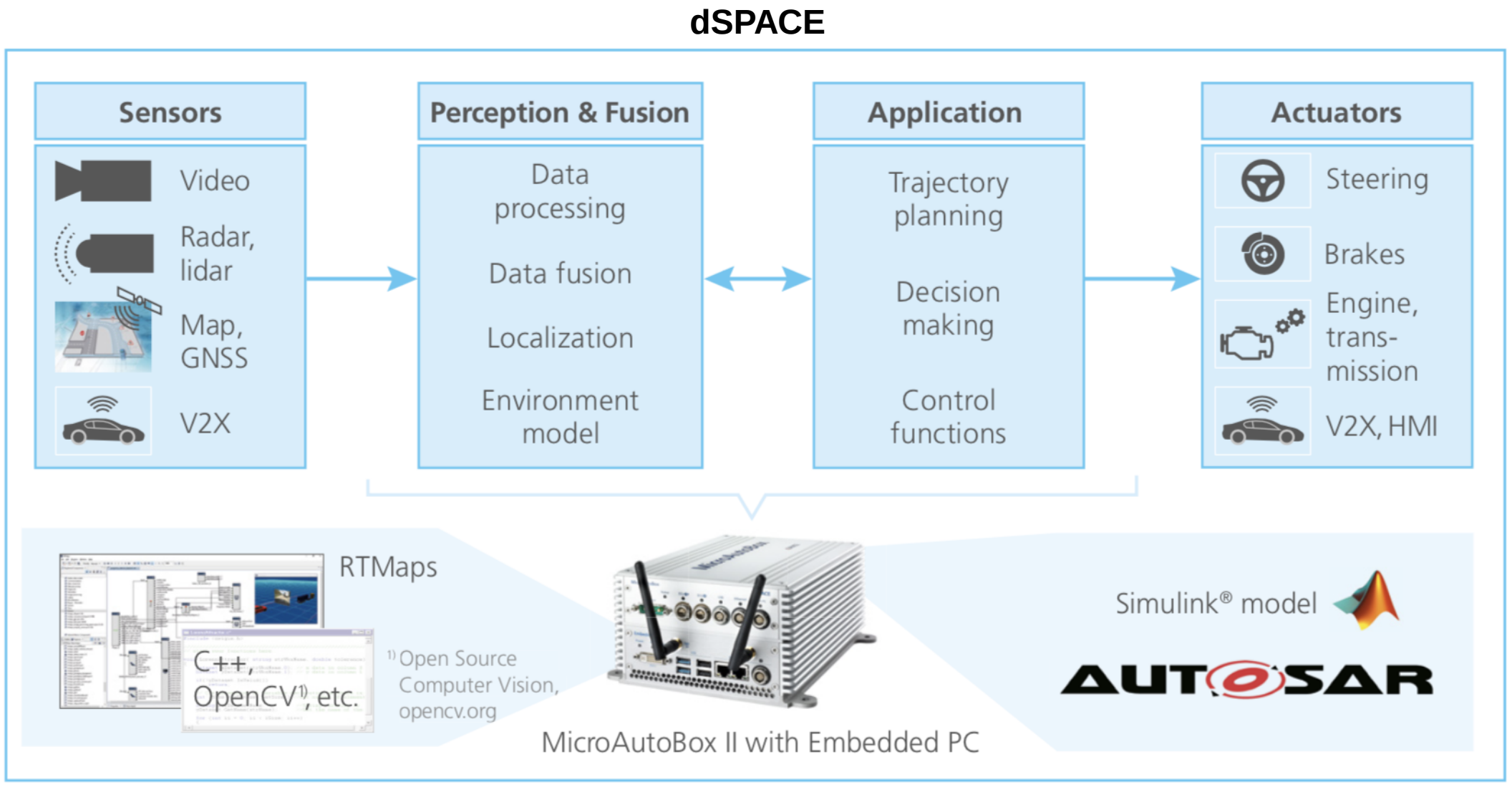


Option	Camera	Radar	Lidar	Ultrasound	GNSS	V2X	Electronic horizon
1	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	n/a	n/a	✓	✓
3	✓	Under development	3-D point cloud	n/a	n/a	n/a	n/a
4	Camera box	ARSG	–	✓	✓	✓	GNSS RF signal

ARSG: Automotive Real-Time Radar Scene Generator; n/a: not applicable; RF: radio frequency

Sensors in the Simulation Environment

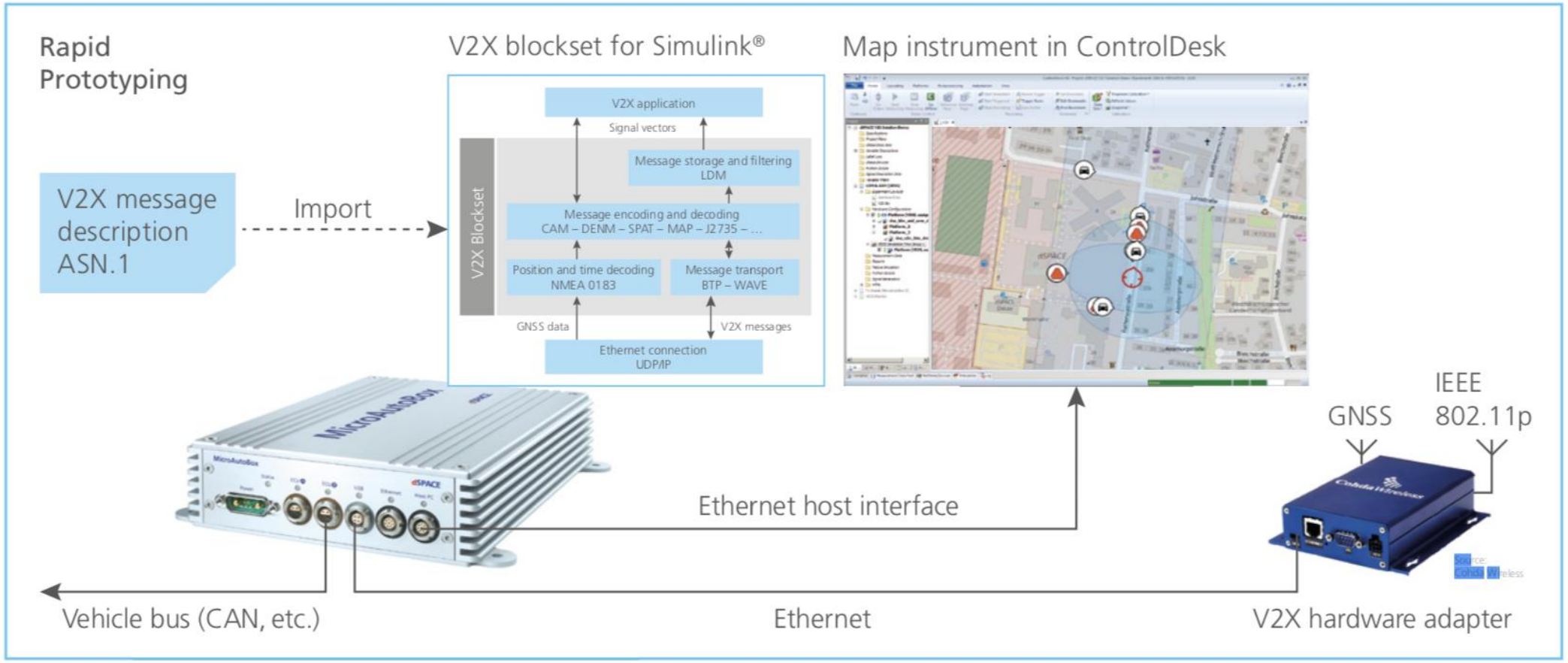
ADAS and Functions for Autonomous Driving



Typical Setup of Prototyping Functions for ADAS and Automated Driving

ADAS and Functions for Autonomous Driving

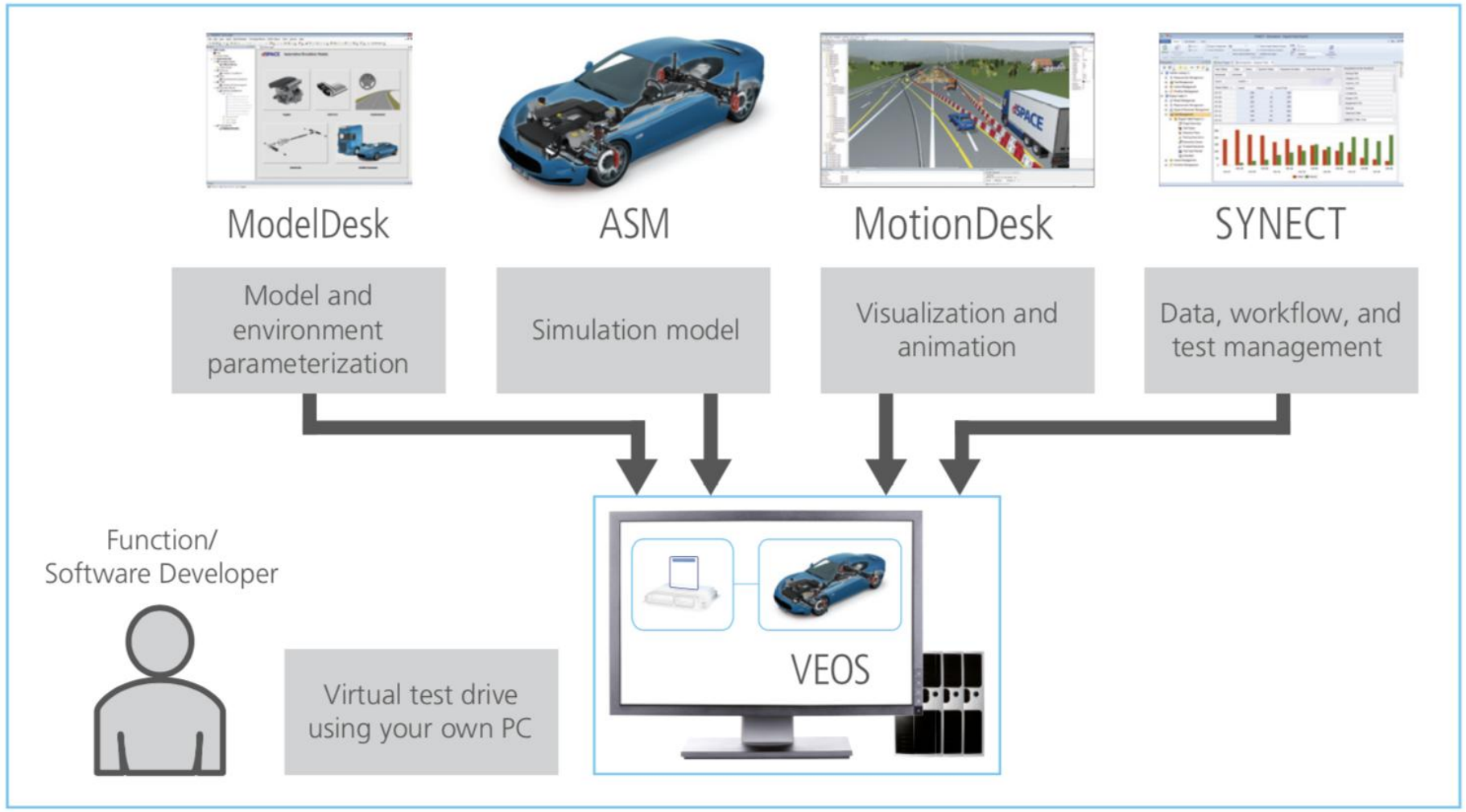
dSPACE



Developing V2X Applications Based on Wireless ad hoc Communication

ADAS and Functions for Autonomous Driving

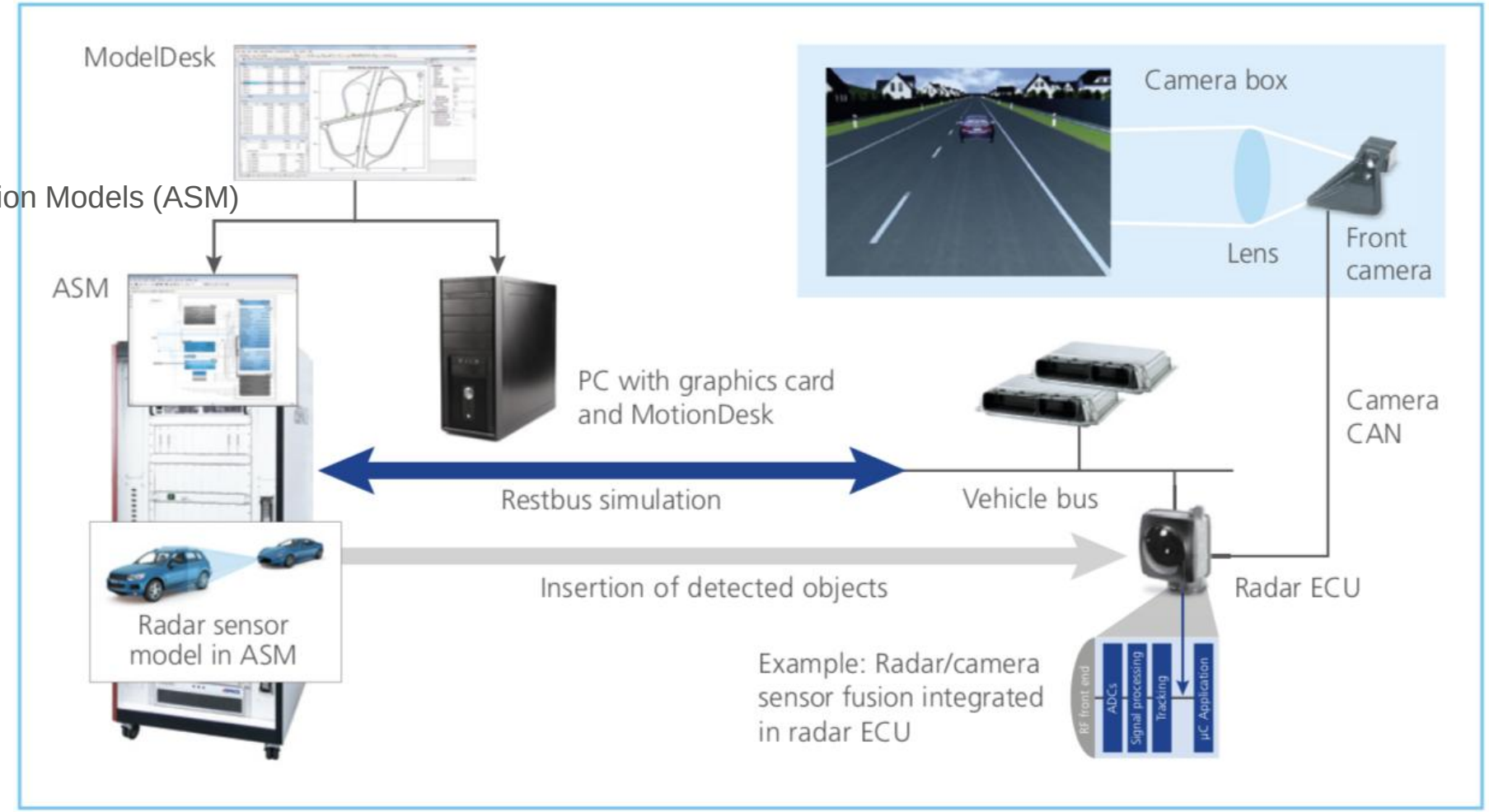
dSPACE



Testing Automated Driving in Urban Areas and on Highways

ADAS and Functions for Autonomous Driving

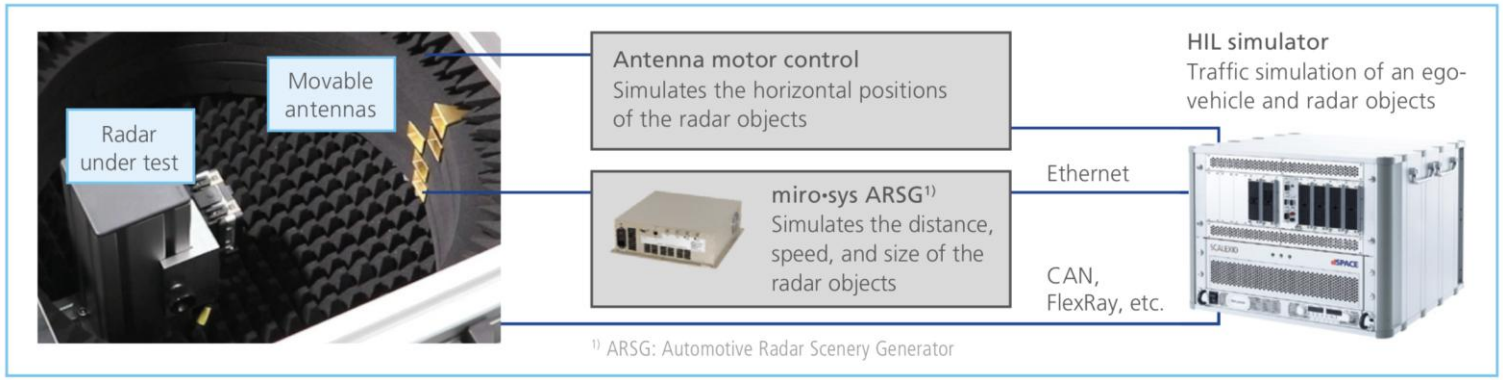
dSPACE



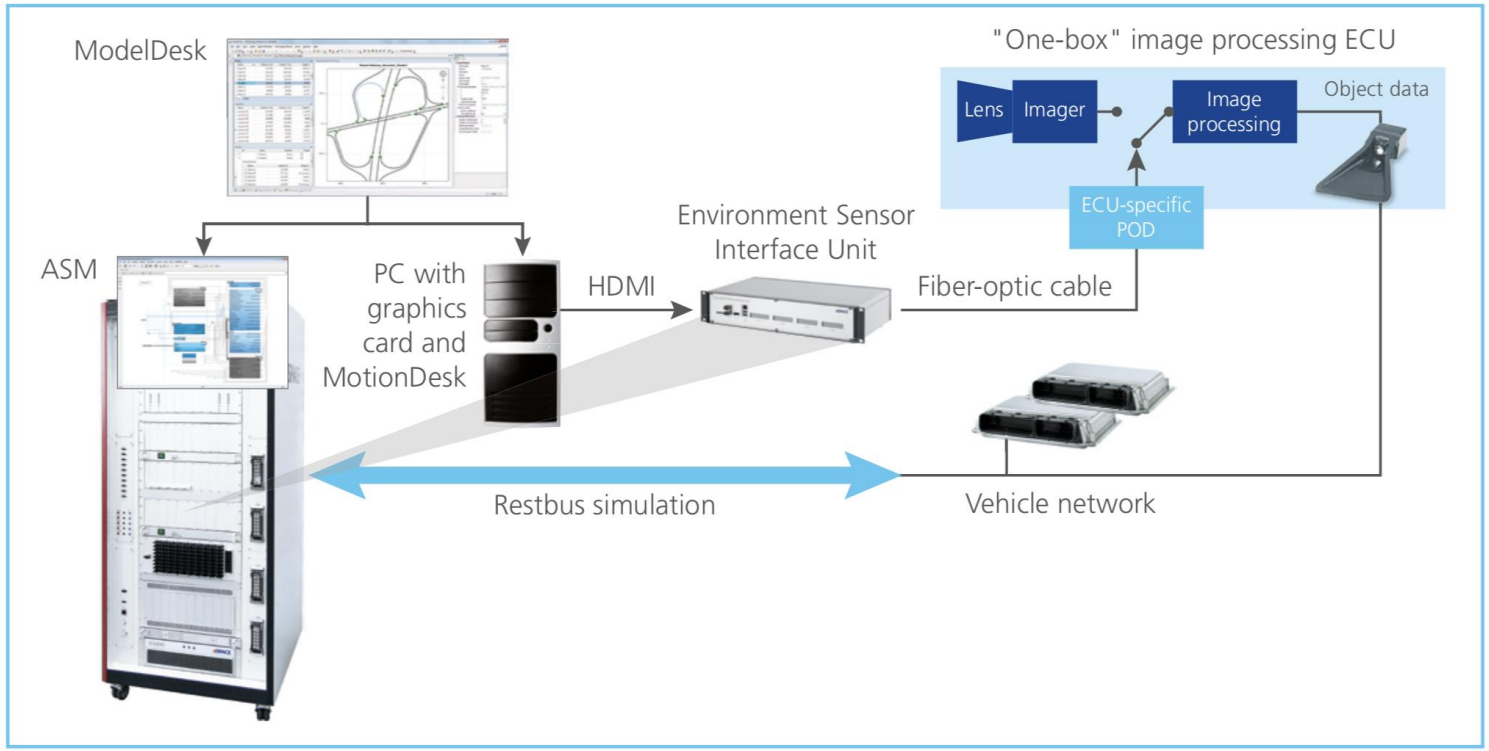
Release Tests for Camera- and Radar-Based Applications Using Closed-Loop HIL System

ADAS and Functions for Autonomous Driving

dSPACE



HIL Test Bench for Over-the-Air Radar-in-the-Loop Simulation



Closed-loop HIL Testing of Multi-sensor Systems

Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing

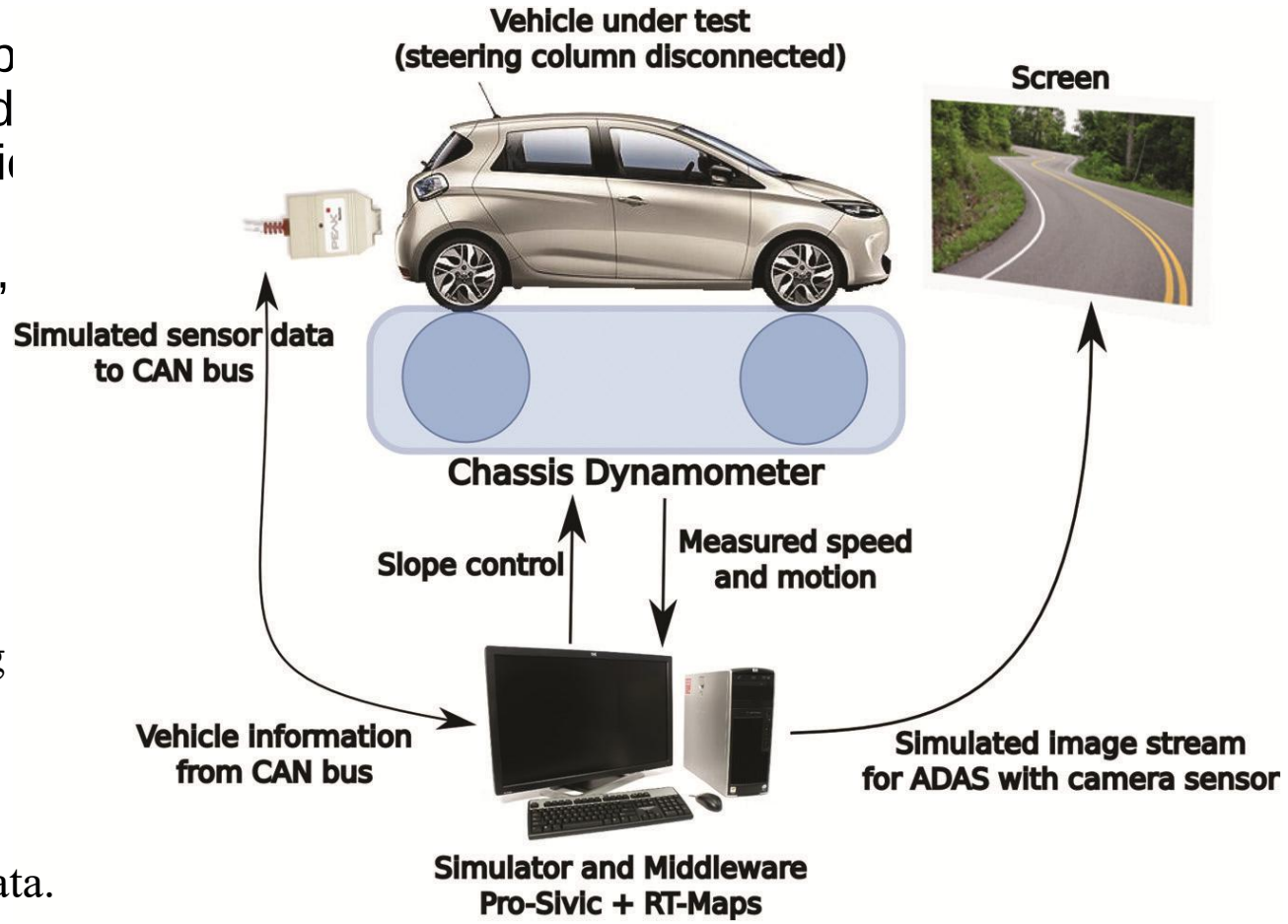
- A virtual testing system built on a chassis dynamometer which enables a complex test scenario to be applied early in the ADAS development process.
- The vehicle test methods can be grouped in two categories: test-bench tests and in-vehicle tests.
- For test-bench tests, three approaches are usually used during the development cycle: Model-In-the-Loop (MIL), Software-In-the-Loop (SIL) and Hardware-In-the-Loop (HIL).
- The in-vehicle tests require a prototype to integrate the developed system, and three approaches are used: test-drives on test-tracks, open-roads and **Vehicle-Hardware-In-the-Loop** (**VeHIL**).
- The **test-track** allows control of some environment parameters (traffic, some weather conditions, road signs, road type and so on) but requires big infrastructures.
- The **open-road** tests require less dedicated infrastructures but are of limited use because of the difficulty to reproduce the needed conditions, and the underlying safety problems.
- Both are costly and time-consuming and can’t be used early in the development cycle because they require heavy engineering efforts to have a fully functional prototype to drive.
- VeHIL is combination of the HIL and test-drives approaches.

Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing

- The classical approach tries to tackle this problem using a multi-stage validation and testing process.
- 1. Model-In-the-Loop (MIL) approach allows quick algo. development without involving dedicated HW.
  - Usually, it involves high-level abstraction SW frameworks running on general-purpose computers.
- 2. SW-In- the-Loop (SIL) validation, where the developed model is evaluated on general-purpose HW.
  - This step requires a complete SW implementation very close to the final one.
- 3. Hardware-In-the-Loop (HIL) involves final HW, running final SW with I/O connected to a simulator.
- This proven process is very widely used in the transportation industry and has enabled the development of very high quality components, integrated into bigger systems or vehicles.
- Modern vehicles however integrate so many such components that the integration phase has become more complex and also requires a multi-step validation process.
- The final integration tests are performed on tracks or roads.
- These real-condition tests are limited because of multiple factors and have a very high cost.

Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing

- VeHIL relies on mobile platforms (Mob Bases) to move targets (fake cars and pedestrians) in front of the tested vehi to trigger the various embedded functions (pedestrian detection, ACC, AEB etc.)



Overview of the SERBER VeHIL system

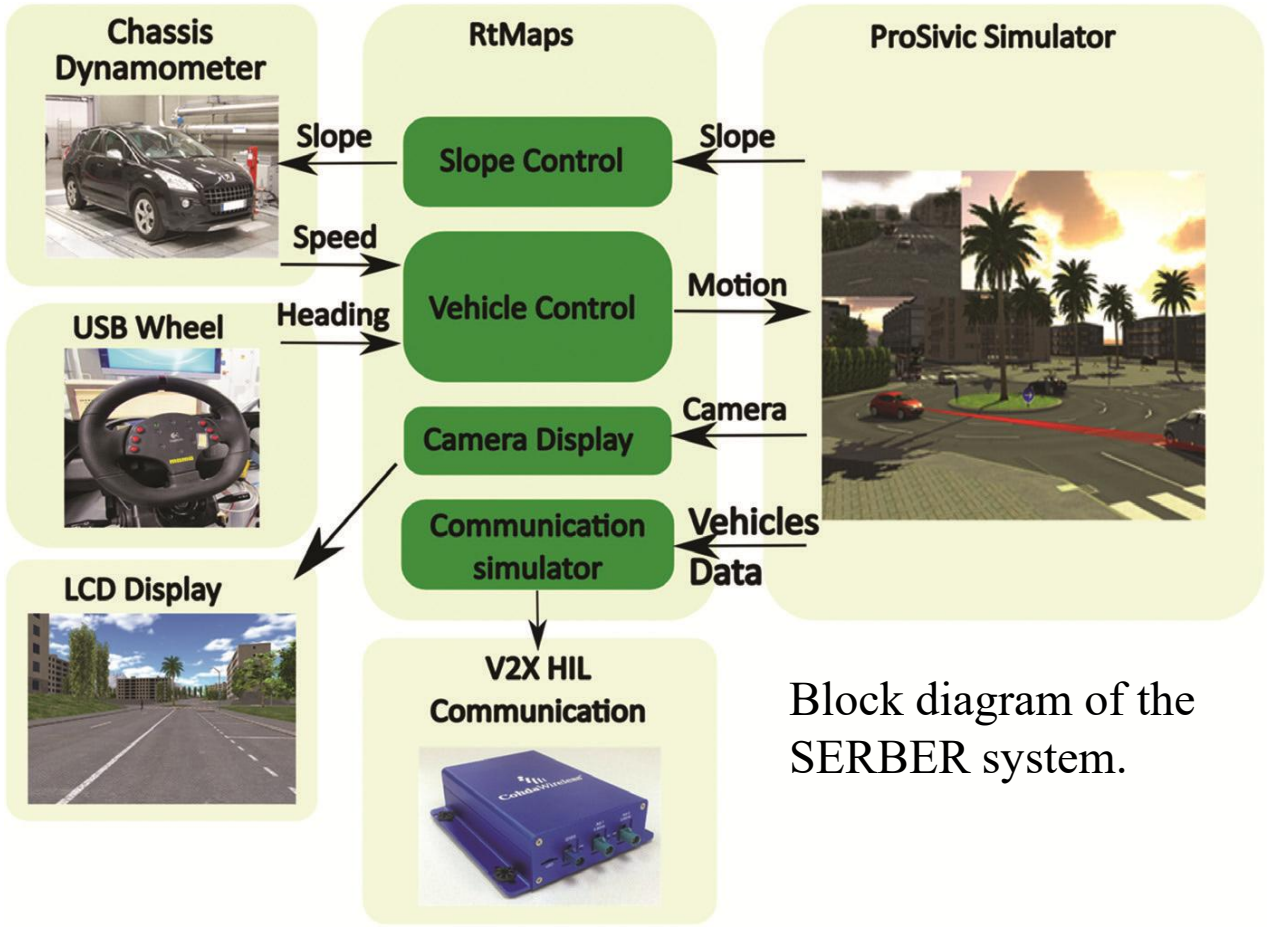
1. the chassis-dynamometer, multi-sensor simulation software (Pro-Sivic) running in a computer;
2. devices to feed the vehicle sensors like LCD screen;
3. the CAN bus interface with synthetic data.

Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing

- How to fool (i.e. feed synthetic data into) the vehicle's sensors with the data produced by the simulation software? Three ways:
  - 1. full simulation is to disconnect completely the ADAS and replace it with a controlled electronic probe, which simulates the ADAS behavior completely.
  - 2. sensor simulation is to disconnect sensor part of ADAS and replace it with an electronic probe.
  - 3. **stimulation** is to keep full ADAS in the loop and send physical stimuli to sensor through HW.
  - The last is preferred, as it keeps ADAS in the testing loop and limits modifications to the vehicle.
- With such a HIL system, multiple scenarios can be implemented and tested in the safety and convenience of an indoor workshop.
- The proposed system is implemented at IRSEEM facilities with a chassis-dynamometer.
- The control system allows interfacing through analog I/O from National Instruments to link the simulation computer with the chassis-dynamometer control system.
- The synthetic data generation for vision, RADAR and LIDAR-based sensors is handled by Pro-Sivic.

Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing

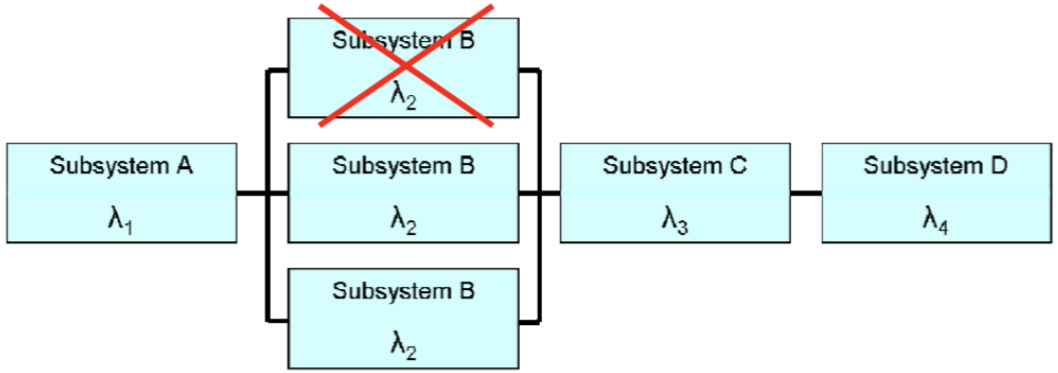
- The SW runs on a PC: a multi-sensor simulation software (Pro-Sivic) from Civitec and a real-time middleware (RTMAPS) from Intempora.
- Pro-Sivic provides kinetic data and sensor data from the simulated vehicle and can be used as a driving simulator.
- **RTMAPS**: a middleware interconnecting all other parts of the system, and it also is a component-based graphical programming framework to easily build multi-tasks or distributed applications.



Block diagram of the SERBER system.

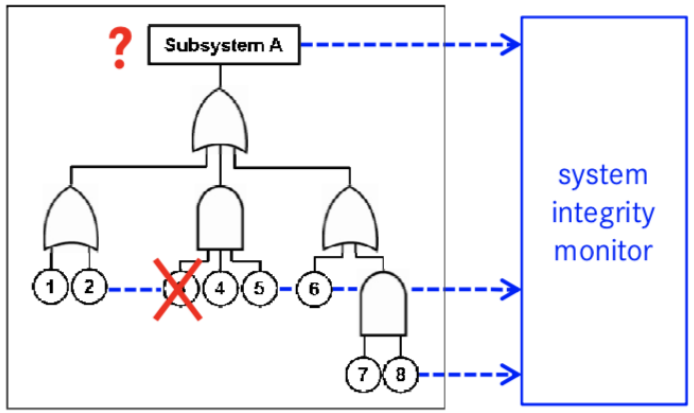
Simulation In Development And Testing Of Autonomous Vehicles At Daimler

- Functional decomposition of complex systems
  - Design for high reliability:
    - Redundant, self monitoring components
      - fault tolerant system design
    - Diverse components
      - Avoid common mode (correlated) faults
    - Fault tree analysis
      - Avoid systematic errors
    - Derive system failure rate by math. model
  - Testing for high reliability:
    - Evaluate system integrity indicators
    - Verify component failure rates
    - Verify un-correlation of component failures
    - Verify rejection of fault propagation



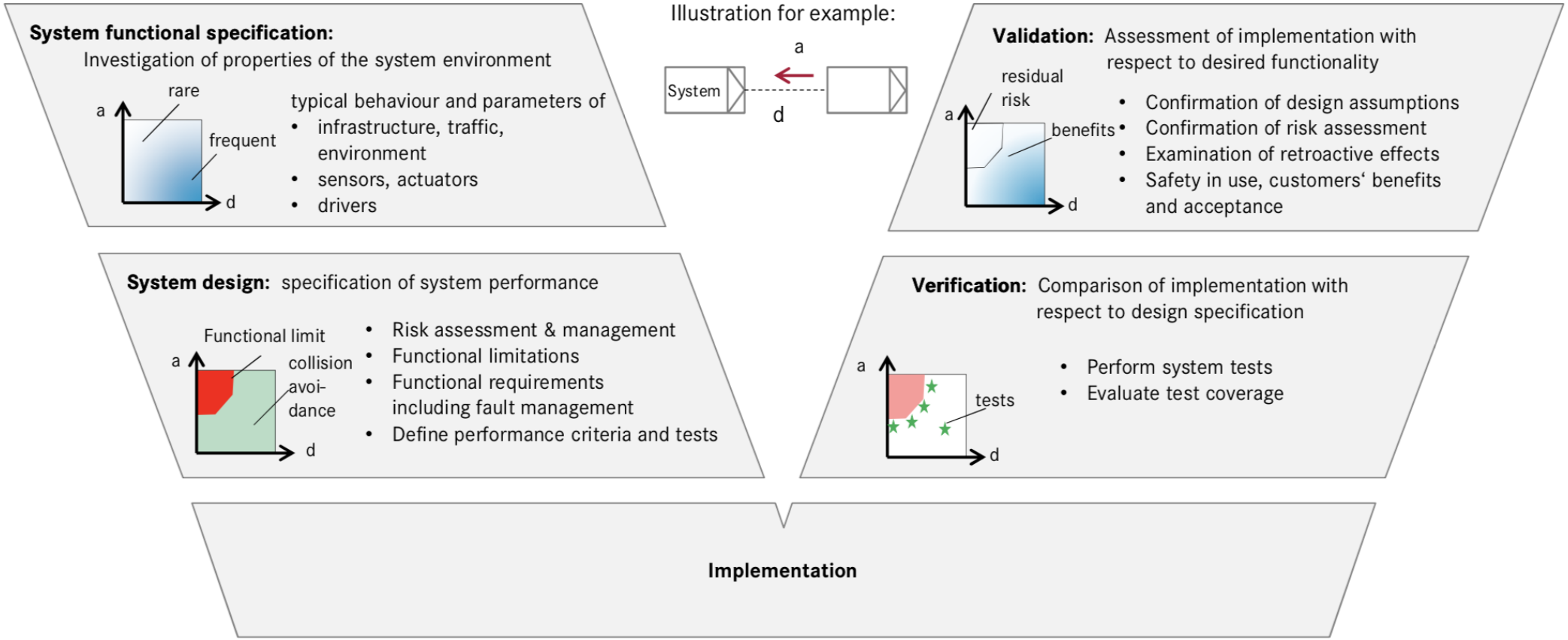
$$\lambda_{total} = \lambda_1 + (\lambda_2)^3 + \lambda_3 + \lambda_4$$

(only if all subsystems are uncorrelated !)



Simulation In Development And Testing Of Autonomous Vehicles At Daimler

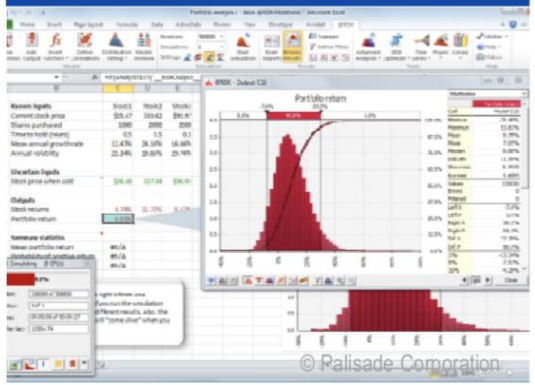
- Verification and validation at the end of the process requires careful specification in the early system design phase



Simulation In Development And Testing Of Autonomous Vehicles At Daimler

- Testing of autonomous driving functions

Virtual tests



☑ Analysis of a huge number of scenarios, environments, system configurations and driver characteristics

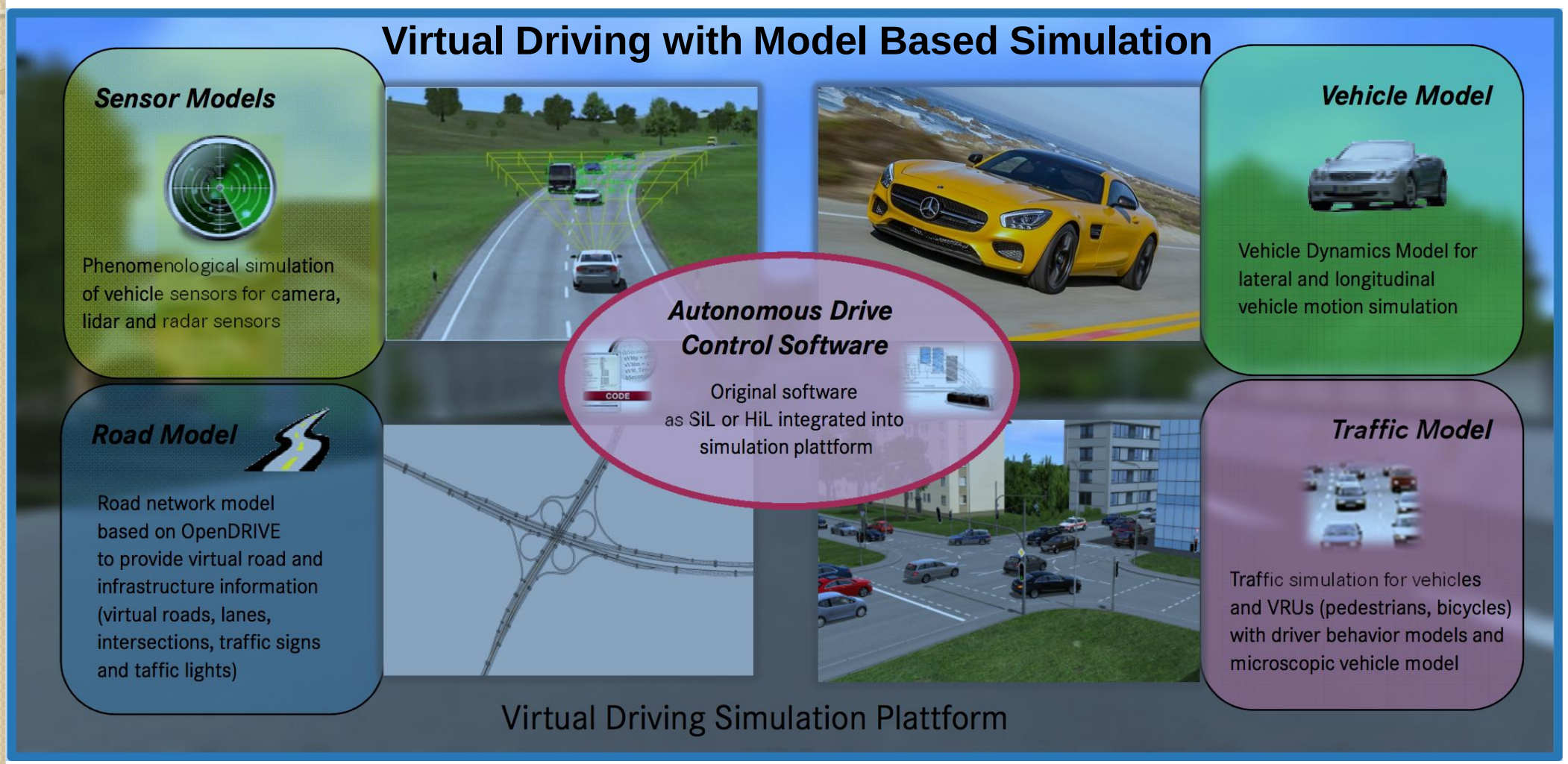
Proving ground tests

☑ Reproducibility by use of driving robots, self driving cars and targets; critical manoeuvres are possible

Field tests

☑ Investigation of real driving situations and comparison with system specifications

Simulation In Development And Testing Of Autonomous Vehicles At Daimler

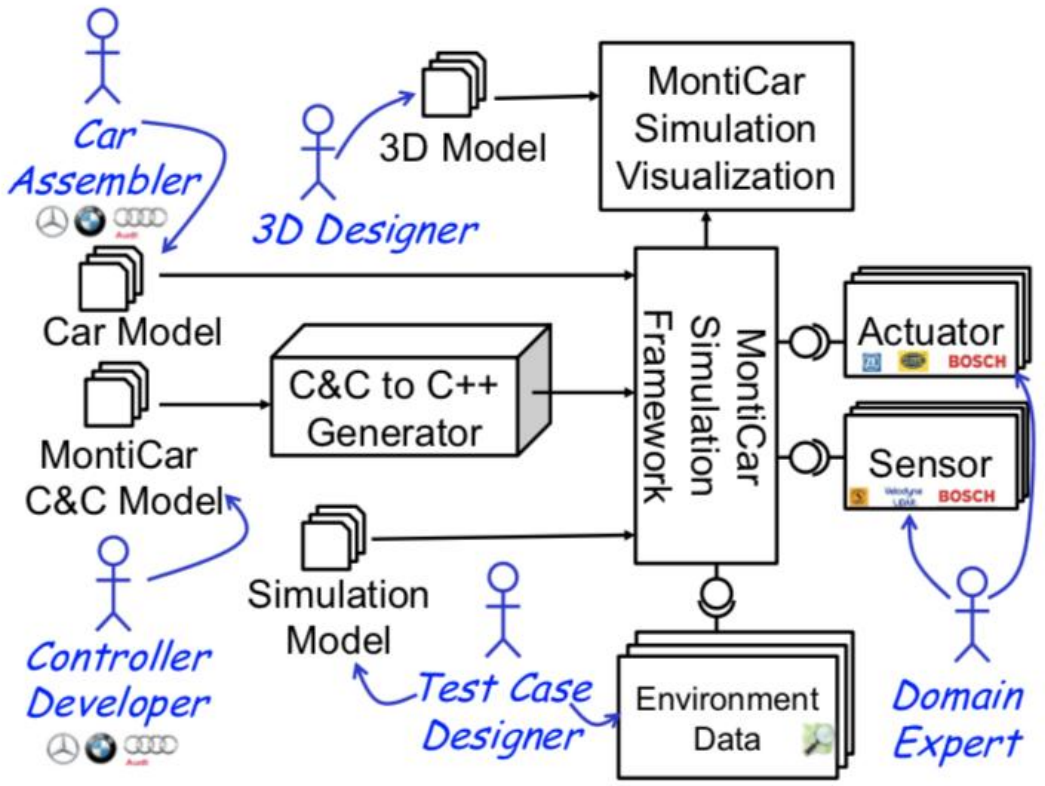


Simulation Framework for Executing Component and Connector Models of Self-Driving Vehicles

- Existing simulators can be separated into high-level and low-level ones, both designed for very specific scenarios and are not suitable for addressing all driving situations.
  - High-level simulators are suitable for mastering large testing environments such as cities, but they lack fine-grained simulation capabilities, e.g., turning of wheels.
  - Low- level simulators provide a high level of detail with realistic motion profiles, usually in small testing environments.
- Combine benefits of both high-level and low-level simulators, to execute component and connector models.
- Simulation developers can choose the most suitable level of detail and integrate real- world environment data from **OpenStreetMap**.
- It allows for adaptations and extensions of the physical vehicle configuration including new sensors, actuators and control systems.
- It has automated testing support and the ability to visualize 3D simulations in a browser.
- The **MontiCAR** Simulator framework supports self-driving car developers with a distributed and collaborative development and testing approach as well as with the ability to partially evaluate very specific parts of the simulated car such as sensors or actuators.

Simulation Framework for Executing Component and Connector Models of Self-Driving Vehicles

- The main part contains multiple car models describing a particular car and one or multiple MontiCAR C&C models to model control systems.
  - These models are independently developed by the *vehicle engineer* and the *controller developer*.
- A *test case designer* creates one or multiple simulation models describing the simulation.
  - Additional environment data can be inserted.
- All sensors and actuators are modeled and integrated by a *domain expert*.
- The MontiCar C&C Model is translated to multi-threaded C++ code, which is executed to control a vehicle during the simulation.
- The visualization part of the simulation framework consists of 3D models created by a *3D Designer* and used in the visualization.



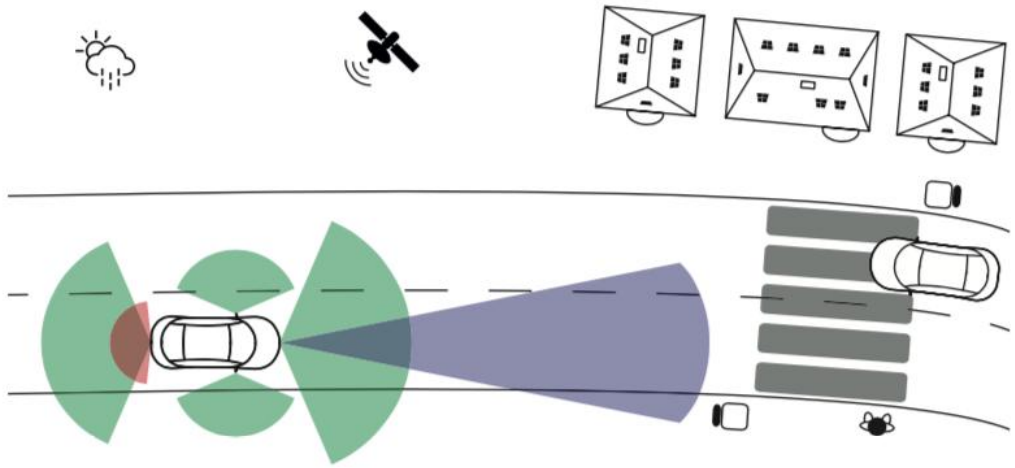
MontiCAR Simulation Framework and its Actors

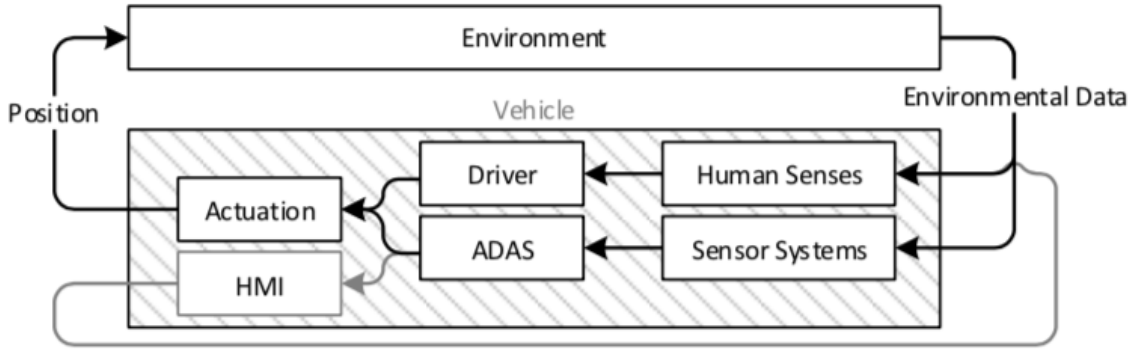
Simulation Framework for Executing Component and Connector Models of Self-Driving Vehicles

- C&C **Unit** Testing without Environment
  - The aim of C&C unit testing, analogue to JUnit testing, is to validate the correctness of single C&C components.
  - If a unit test is successful, the result is a list of components used in this test case.
  - Based on this info., a test coverage is computed.
  - If a unit test fails, then the list of failing components is presented.
  - Note that the inner-defined C&C components are regarded, as well.
- C&C **Integration** Testing with Environment
  - Unit testing controllers, sensors, or actuators can also be done in an environment to simulate real world scenarios.
  - To test different controllers, deviation from the optimal path is computed for each controller.
  - Based on this deviation, the most optimal controller can be identified.

A multi-domain simulation approach to validate advanced driver assistance systems

- **Hardware-in-the-Loop** (**HiL**): Integrating a physical device, often an ECU, into a closed loop simulation to test its functionality in an early stage of the development process.
- In the style of HiL, **Model-in-the-Loop** (**MiL**) or **Software-in-the-Loop** (**SiL**) are used where either a model of a controller or the embedded software is the **Unit under Test** (**UuT**).
- The functionality of controller software, the ECU software, a physical ADAS ECU or even the behavior of a real car within a virtual world, a driver, a powertrain, tires and sensors.
- Additionally the scenario might include pedestrians, a street, houses and traffic lights.





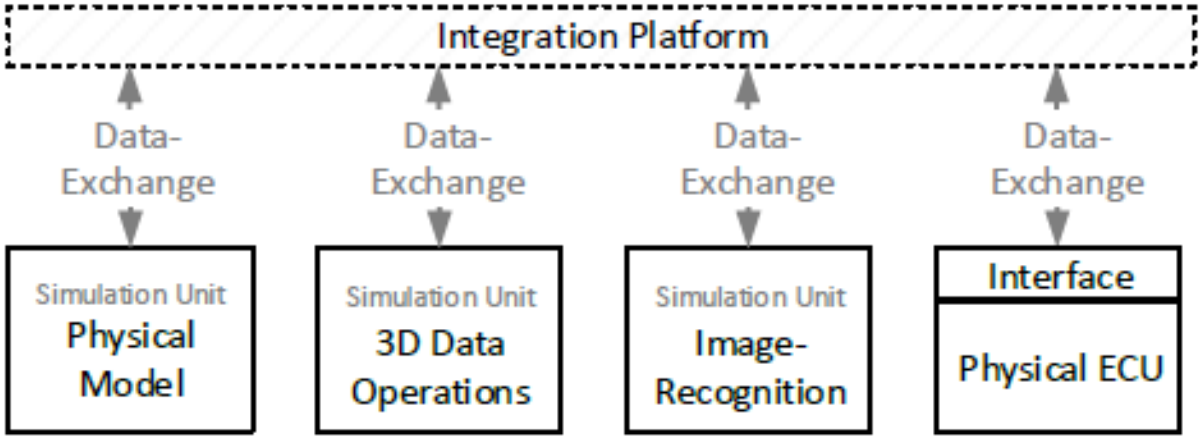
A multi-domain simulation approach to validate advanced driver assistance systems

- **Functional Mock-Up Interface** (**FMI**) combines multiple simulation-units with Inputs and Outputs to realize data exchange and an algo. to schedule execution.
- Each **FMU** (**Functional Mock-Up Unit**) can have its own solver (as FMI for co-simulation) which is optimized for a specific domain.
- Such a platform can be realized using the FMI standard, where a master algo. schedules executions and data exchange among multiple simulators, as FMUs.
- Interface of FMI for co-simulation is a standard to solve time dependent coupled systems consisting of subsystems that are time-continuous or time-discrete.
- By supporting extra/inter-polation of subsystem inputs and communication step-size control, the FMI v2.0 standard addresses numerical challenges when having multiple simulation-units performing time integration independently.

A multi-domain simulation approach to validate advanced driver assistance systems

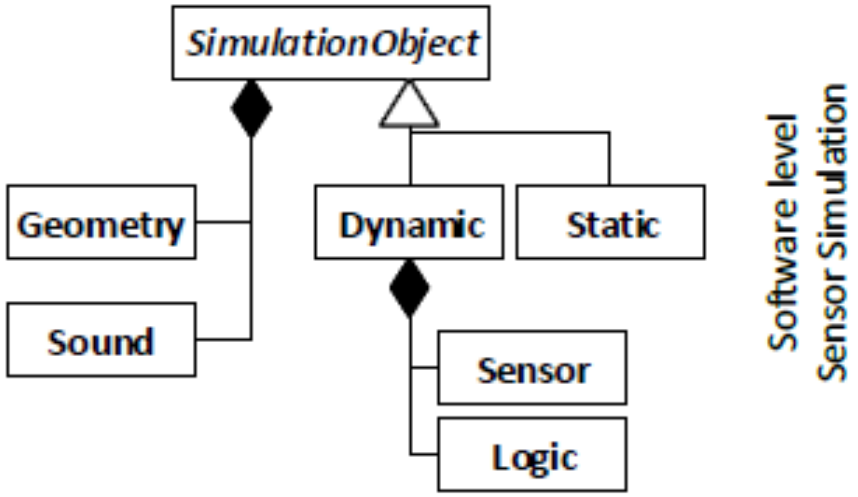
- Separated disciplines can make use of their specific and optimized tools, while having the possibility to include information from other domains.
  - Vehicle dynamics might be represented by a physical model, created with ASCET or MATLAB /Simulink, such as solving ordinary differential equations (ODEs).
  - Sensor-simulation operates on complex three-dimensional meshes to represent objects in a virtual environment, which be realized using a 3D engine.
  - Image recognition algorithms in automotive industry are often implemented and tested by using the **Automotive Data and Time-Triggered Framework** (**ADTF**).
  - Additional domains as aerodynamics or electrical simulations might be included.

System-Level view of an exemplary ADAS HiL set-up

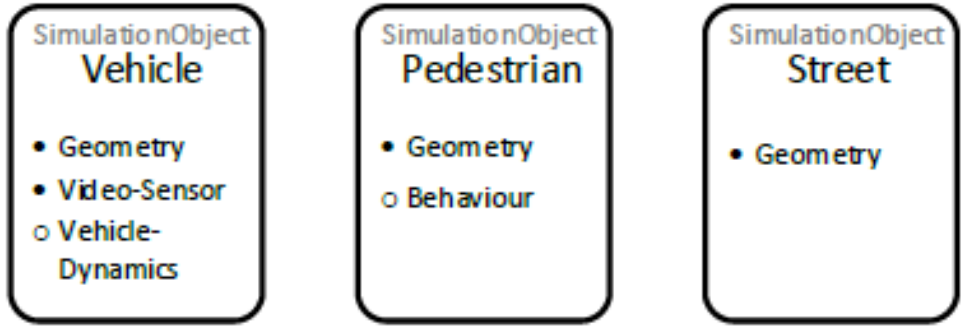


A multi-domain simulation approach to validate advanced driver assistance systems

- The execution chain of the 3D simulator consists of four steps:
  - 1. the inputs of the sensor simulation-unit are read;
  - 2. the objects representing the environment are updated;
  - 3. the sensor effects are simulated;
  - 4. the results are set as the unit's output.
- The composition of components in Game development is referred to as GameObjects;
- SimulationObjects (SOs): Multiple components are composed to it.



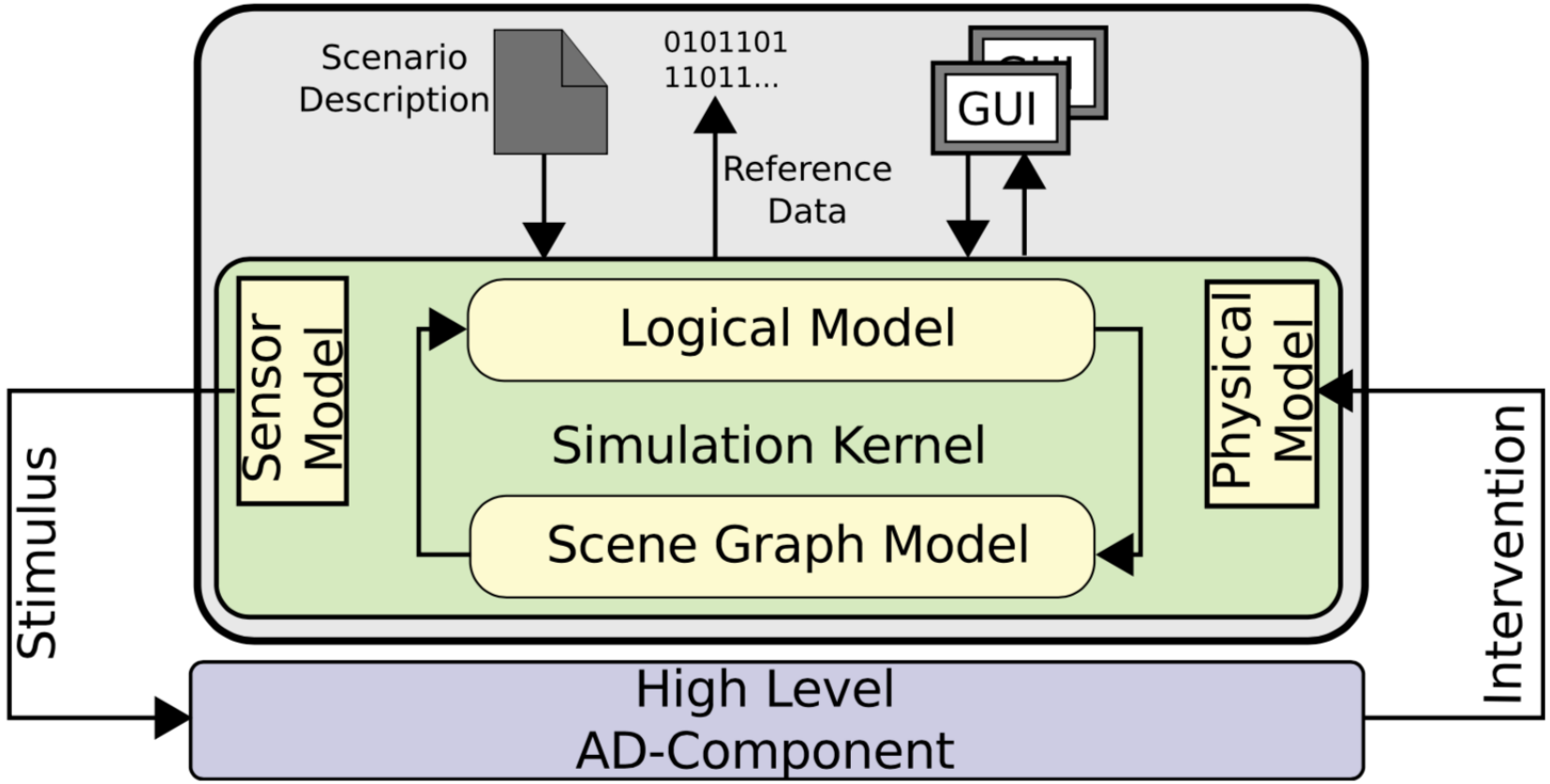
Software level Sensor Simulation



Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios

- By combining foci of different simulations, to provide an increased amount and diversity of traffic scenarios, relevant to development and verification of automated driving functions.
- The vehicle mechanics simulation serves as an integration interface and an object-oriented environment model, called **scene graph model**.
  - This model provides ground truth and noisy info. access through generalized sensor models;
  - The traffic simulation determines the dynamic behavior such as those of traffic participants.
  - A structured approach for the variation of surrounding traffic given real world road networks for the virtual assessment of highly automated driving functions is achieved.
- A concept for realistic simulation scenarios, capable of running in different integration levels, from software- to vehicle-in-the-loop, consisting of four models as
  - The **Logical Model** handles the updating process of simulated traffic participants' states.
  - The **Scene Graph Model** assigns geometric models and additional object properties to the traffic participants and static environment.
  - The **Mechanics Model** defines kinematic and dynamic elements of a vehicle .
  - The **Sensor Model** provides localization used to derive restricted views on the virtual scene.

Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios

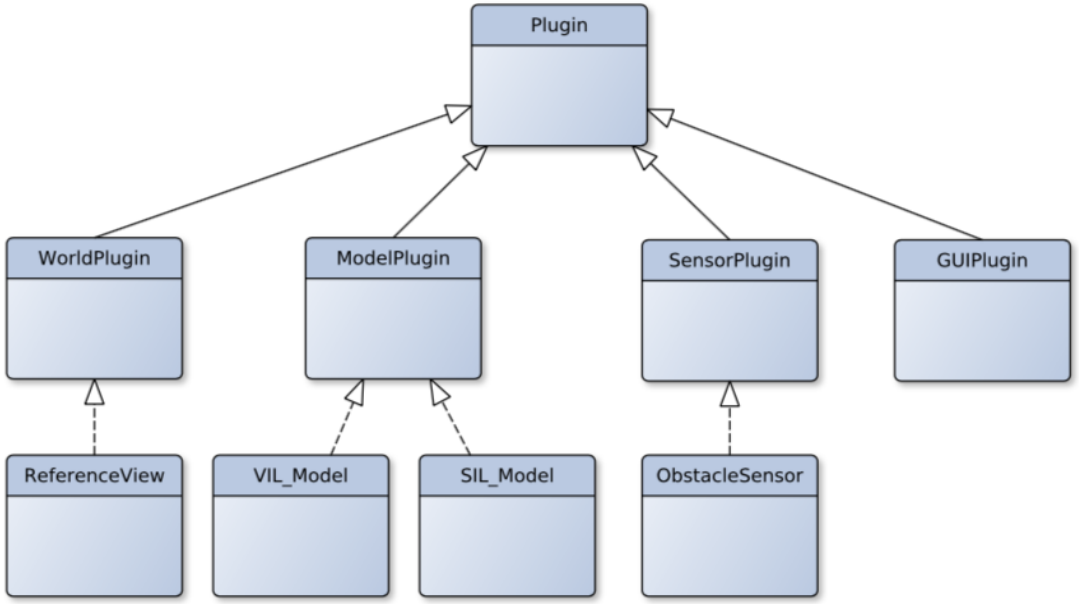


The high level component-under-test is stimulated by the simulation, and an appropriate vehicle interface is called or - in the case of vehicle-in-the-loop - an avatar without any own mechanic element is moved according to the observed reaction of the experimental vehicle.

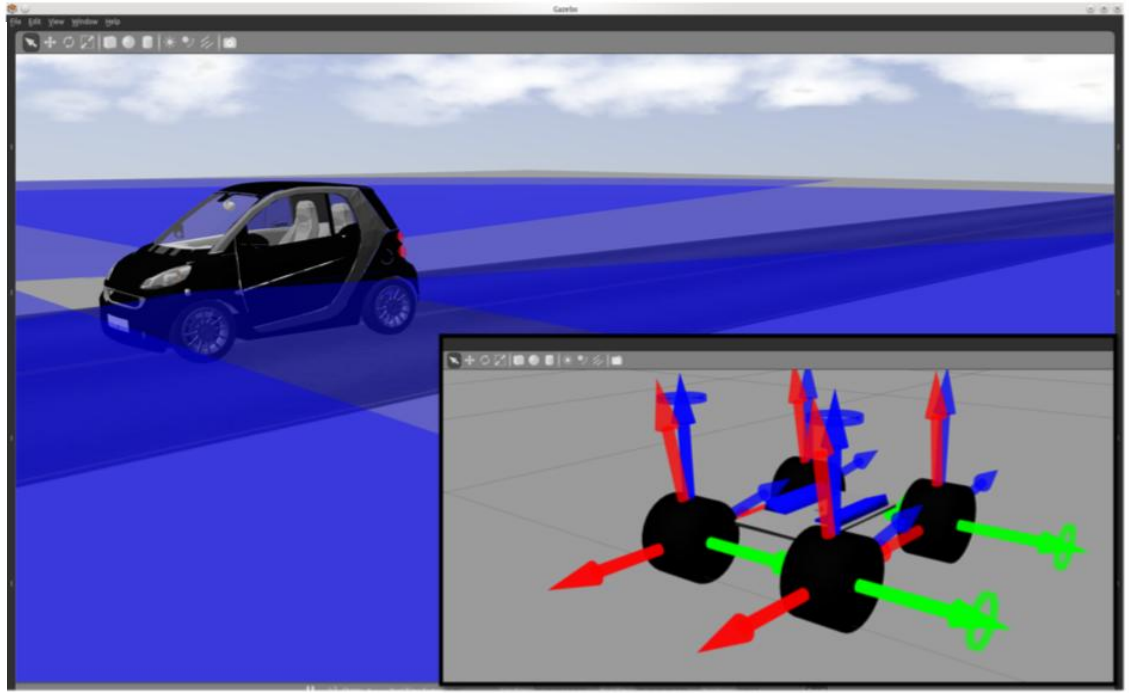
Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios

- Free frameworks, such as robotics simulation *GAZEBO*, robotics middleware *ROS* and traffic flow simulation *SUMO*, an exemplary instance of the logical model;
- The toolchain to create realistic traffic scenarios consists of the following steps:
  - Road networks are extracted from the OpenStreetMap format, converted into a graph based road network description, which is readable by SUMO.
  - Based on additional info. in OpenStreetMaps, such as static environment, geometrical models can be exported to the *Scene Graph Model*.
  - Based on the road network, either random routes or fixed routes are created.
  - Also the scenario description for the *Scene Graph Model* is built up using *Simulation Description Format* (SDF), providing several elements for the definition of the static environment, the road network and the configuration of the high level component to the *Physical* and the *Sensor Model*.
  - To instantiate the *Sensor Model*, the *Scene Graph Model* and the *Mechanical Model, use* Gazebo.
- Gazebo provides several kinds of plugins to map different skills to the framework.
  - It provides a high modularity, which makes Gazebo flexible and scalable and thereby attractive to employ models at different scales and with different interfaces.

Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios

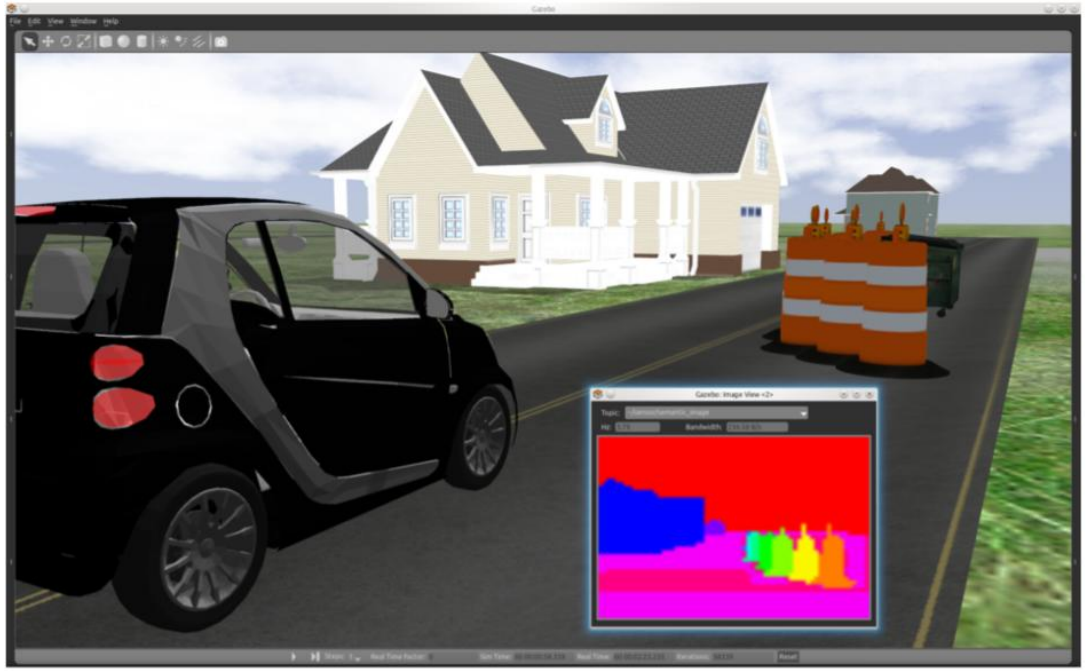


The plugin interface is used for defining multiple models, sensors and view aspects in our simulation frame- work.



Visualization of a detailed vehicle model: This model concerns aspects such as Ackermann steering and suspension units, including actuators, controllers and sensors in order to develop autonomous driving functions. The blue sectors indicate the sensor ranges.

Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios



Visualization of a virtual scene and the scene labels: A ray-casting algo. is used to simulate a camera-based object classification. This abstract sensor then provides obstacle lists, containing info. about perceived obstacles.

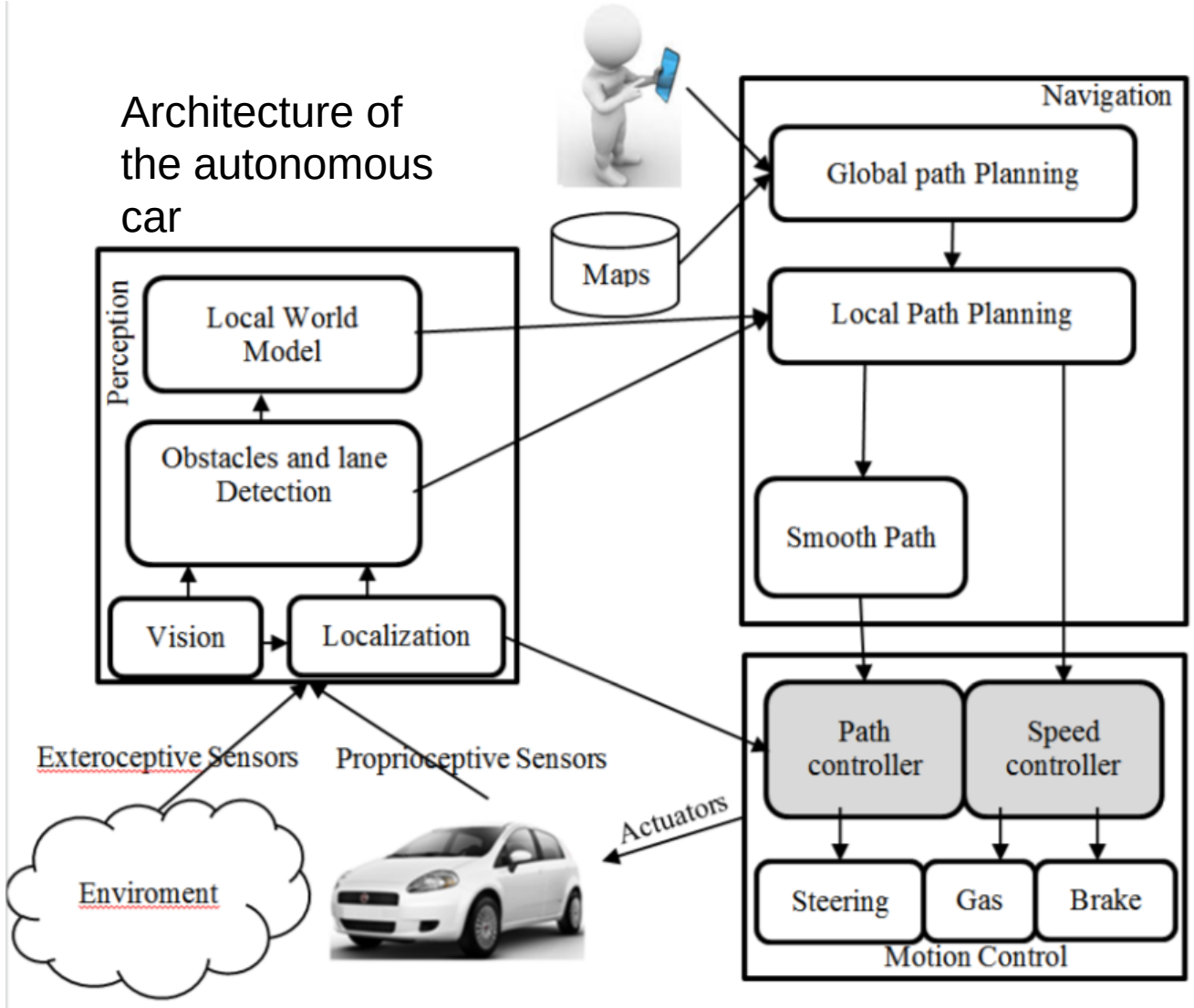
The modeled testing ground has been equipped with virtual traffic participants from the traffic flow simulation SUMO: The **black** avatar is moved according to the measured movement of the real experimental vehicle, whereas the yellow vehicles' behavior is determined by SUMO.

SOFTWARE ARCHITECTURE FOR AN AUTONOMOUS CAR SIMULATION USING ROS, MORSE & A QT BASED SOFTWARE FOR CONTROL AND MONITORING

- Architecture and software used in the autonomous car simulation.
- The programs used are MORSE, ROS and a custom program based on the QT framework.
- Autonomous functions like control of orientation and vehicle velocity are implemented inside the architecture.

Note: **MORSE** (Modular OpenRobots Simulation Engine) is an academic robotic simulator, based on the Blender Game Engine and the Bullet Physics engine. It is a BSD-licensed project.

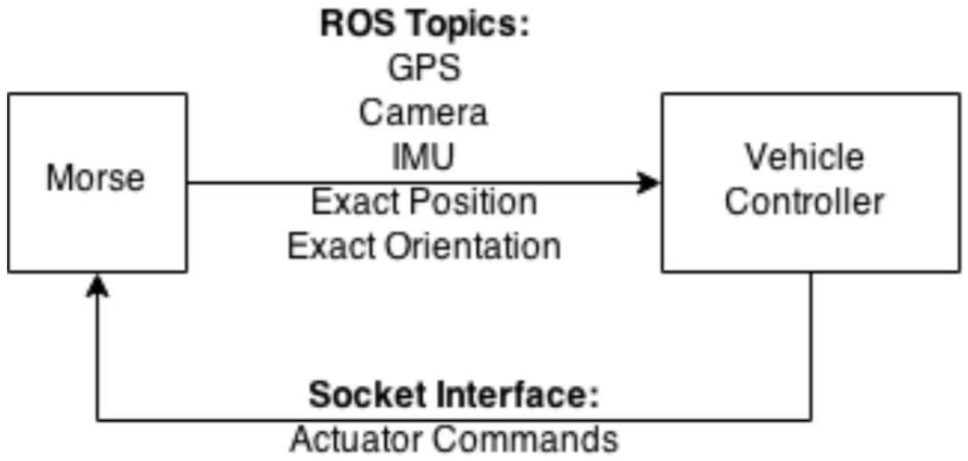
Architecture of the autonomous car



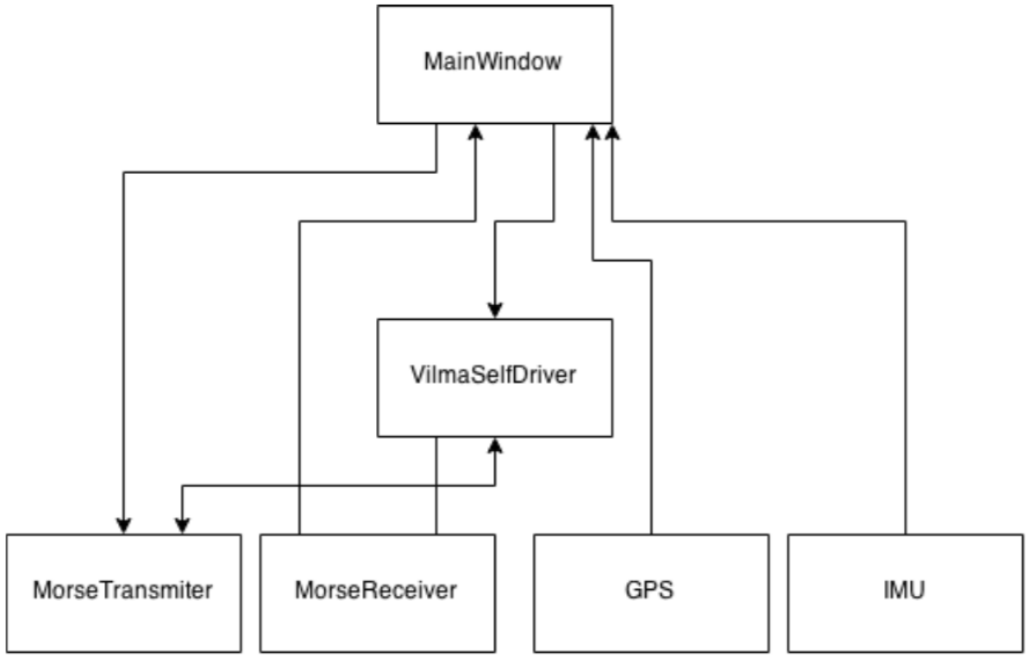
SOFTWARE ARCHITECTURE FOR AN AUTONOMOUS CAR SIMULATION USING ROS, MORSE & A QT BASED SOFTWARE FOR CONTROL AND MONITORING

- The tools available in the front-end are displaying of data from the sensors in real time, displaying of the current value being sent to the car actuators and plotting data in real time from values received by the sensors in a separate window;
- The value update frequency displayed on the screen such as position, orientation and velocity, are controlled internally by a timer which when triggered queries each sensor module, gets an updated value and updates the displayed image.
- The architecture of back-end is highly modular using ROS or any other middleware;
- For each sensor there is a module which receives data from ROS, but since each module has it's own class other middlewares can be used without any modification of the program architecture.
- The modules communicate with each other using functions of the type get and set.
- The internals of each class are not important to the other modules as long as the API is kept the same:
  - GPS, IMU, MainWindow, MorseTransmiter, MorseReceiver, VilmaSelfDriver

SOFTWARE ARCHITECTURE FOR AN AUTONOMOUS CAR SIMULATION USING ROS, MORSE & A QT BASED SOFTWARE FOR CONTROL AND MONITORING



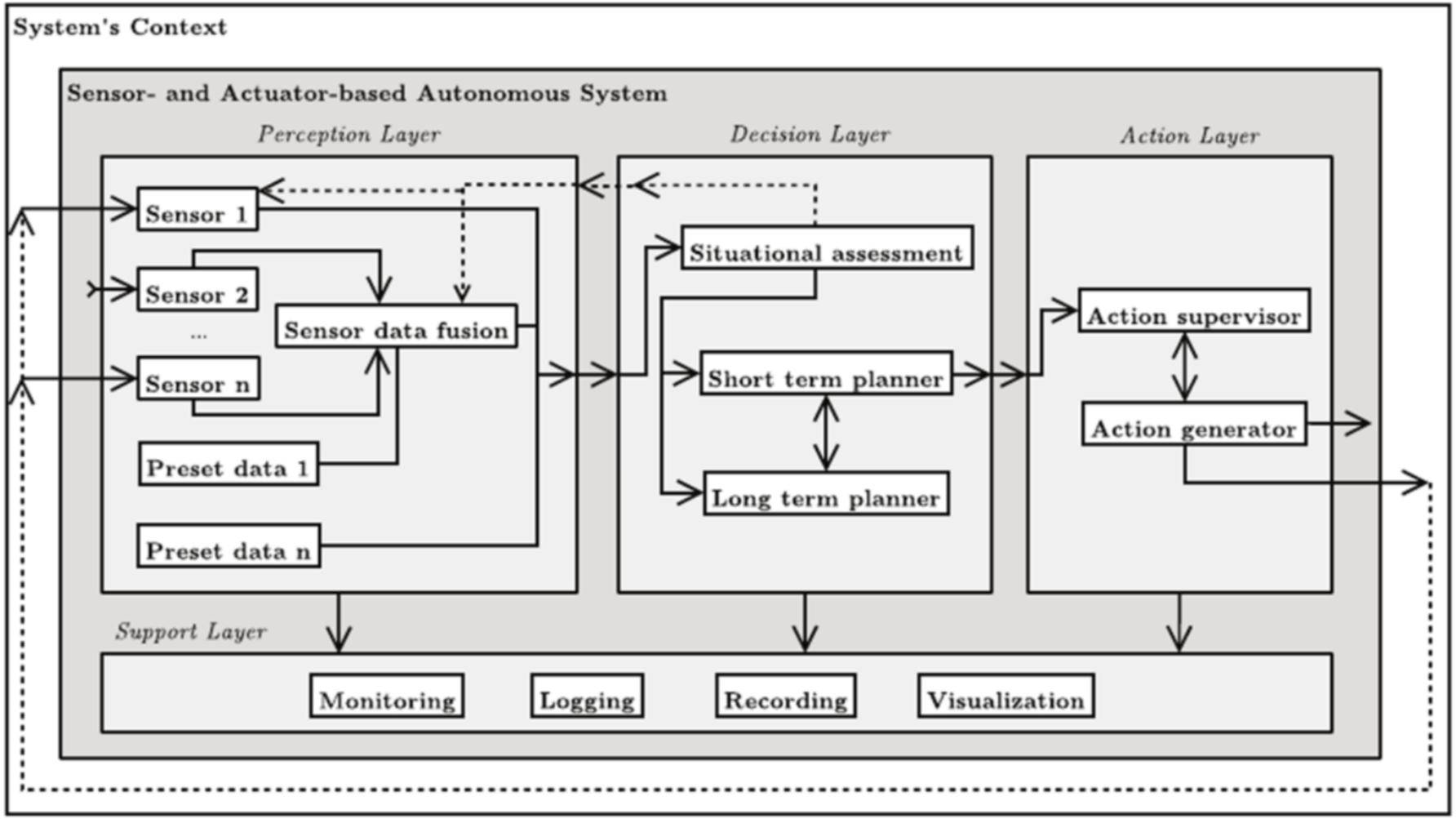
Simplified simulation architecture



Modules communication scheme.

Note: **VilmaSelfDriver** controls the vehicle according to the instructions given in the user interface. It supports controlling the car speed and it also allows the vehicle to follow pre-determined trajectories by a series of waypoints. These waypoints can be entered manually in a table in the user interface or loaded from a pre-recorded text file.

An Integrated Architecture for Autonomous Vehicles Simulation



A generic sensor- and actuator-based autonomous system architecture

An Integrated Architecture for Autonomous Vehicles Simulation

- A taxonomy for comparing autonomous vehicles simulators is proposed:
  - **3D rendering**: The visual robustness of the simulation;
  - **License**: Whether the simulator is open-source, free of charge, or commercially licensed;
  - **External Agent Support**: In order to control a vehicle using an agent-based methodology the simulator should feature a distributed architecture at the control level;
  - **Parallelism/Distribution**: In order to distribute processing power over processor cores or networks;
  - **Level of Maturity**: If the simulator is already widely used and validated;
  - **Fault-tolerance**: When a hardware module fails, higher level modules should rapidly make decisions whether to stop or adapt the control system. When developing a final product such behavior should be strictly tested;
  - **Realistic Scenario Simulation**: The level of realism to simulate difficult context scenarios e.g. snow, day and night, and wind, not only interactively but also physically, i.e. affecting the sensorial input.

An Integrated Architecture for Autonomous Vehicles Simulation

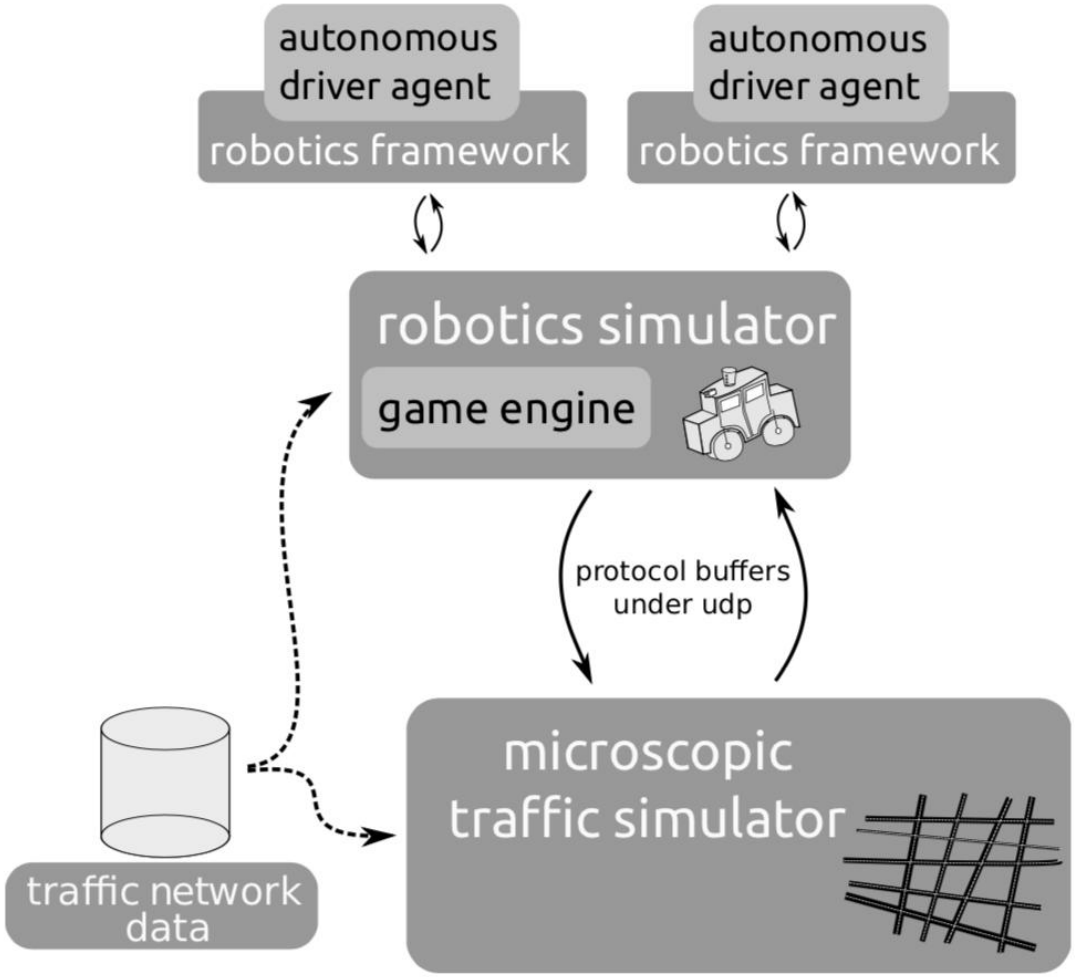
- Evaluating Traffic Simulators for Autonomous Vehicles Simulation
  - **Extensibility:** When using a closed-source SW, its extensibility should be analyzed to study if it suits the integration of other tools, i.e. the level of accessibility of the simulation core;
  - **SW License:** Open-source simulators are generally inferior in features as compared to commercial ones. Nonetheless, when well documented they tend to be more flexible and rapidly extended due to community support;
  - **External Agent Support:** The ability to use the agent technology, not only in driver behavior modeling, but in simulation initiation, control or deployment;
  - **Parallelism/Distribution:** To support a large traffic scenario, simulators must feature distributed processing over several cores or a computer network;
  - **Inter-vehicular communications (IVC):** Virtual communication infrastructure support for V2V or V2I and physical restrictions simulation must be present as well;
  - **Interactivity:** What features are controllable from simulation in run-time, and general graphical aspect;
  - **Level of Maturity:** Whether the simulator is widely used and validated by the scientific community.

An Integrated Architecture for Autonomous Vehicles Simulation

The architecture for autonomous vehicle simulation in a traffic environment.

The two simulators are chosen according to the requirements discussed already, followed by the high level architecture of the system.

- ✓ SUMO is a highly portable, microscopic road traffic simulation package designed to handle large road networks and has a commitment with the academia and research community.
- ✓ USARSim is a HF robotics simulator based on the Unreal Tournament game engine.
  - ➢ USARSim is written in Unreal Script, the official scripting language which offers interface with the engine core.
  - ➢ Drawback is its dependence on Windows Platform, as Unreal Engine 3 supports it.



An Integrated Architecture for Autonomous Vehicles Simulation

- **Microscopic Traffic Simulator** Simulates almost all vehicles with a resemblance to the macroscopic simulation of real traffic streams. It also maintains all infrastructure systems, such as induction loops or traffic light plans. Given the extensibility of the simulator, a higher level statistical framework may be coupled in order to study traffic behavior patterns of individual and cooperative strategies for autonomous vehicles;
- **Robotics Simulator** Performs the simulation of all autonomous vehicles in the environment, along with all its sensors and actuators, and mirrors every surrounding object. It also features a game engine for immersive 3D animation through both powerful physics and visualization modules;
- **Coherent Network Data** Represents the traffic network topology model, as well as its realistic 3D environment data;
- **Autonomous vehicle interface and control** Manages the brain of the vehicle. Typically an external software driver can be deployed to perform the autonomous vehicle high-level tasks using an agent-based methodology. A **Hardware Abstraction Layer** (**HAL**) is underneath for transparent real/virtual world development, and sensor/actuator permutation.

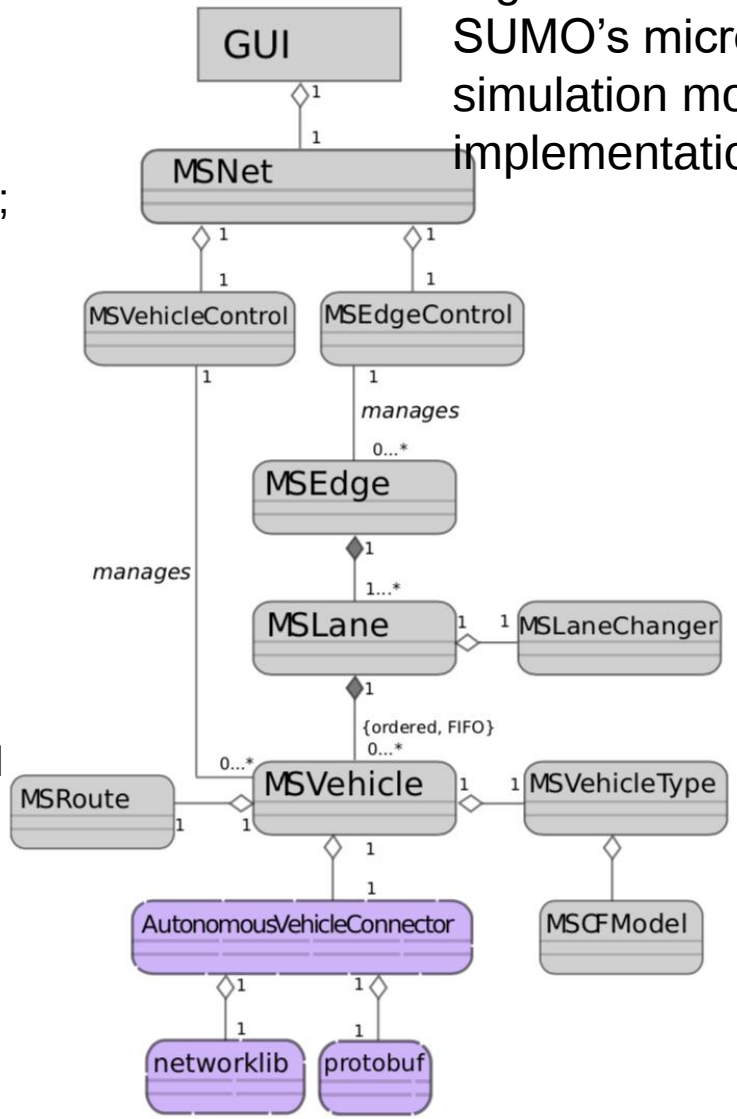
An Integrated Architecture for Autonomous Vehicles Simulation

- Simulation of Urban MObility (SUMO) is a 2D microscopic traffic simulator project.
- Implemented in standard C++;
- Includes all applications to prepare and perform a traffic simulation (network and routes import, dynamic user assignment (DUA), simulation);
- Network Import:
  - Imports VISUM, Vissim, Shapefiles, OpenStreetMaps, RoboCup, and XML-Descriptions;
  - Missing values are determined via heuristics;
- Routing:
  - Microscopic routes - each vehicle has its own;
  - Different DUA algorithms;
- High portability
  - Only standard C++ and portable libraries are used;
  - Available packages for Windows, Linux distributions;
- High interoperability through usage of XML-data.

- **Simulation**:
  - Space-continuous and time-discrete vehicle movement;
  - Manages different vehicle types;
  - Multi-lane streets with lane changing;
  - Different right-of-way rules, traffic lights;
  - Fast openGL graphical user interface;
  - Manages networks with several 10k edges (streets);
  - Fast execution speed (e.g. up to 100k vehicle updates/s on a 1GHz machine);
  - Interoperability with other application at run-time (using TraCI interface 1);
  - Network-wide, edge-based, vehicle-based, and detector-based outputs;
  - GUI and command-line based simulation.

An Integrated Architecture for Autonomous Vehicles Simulation

- **GUI**: Graphical User Interface model, which controls the microsimulation parameters and deployment;
- **MSNet:** It also contains all microsimulation related objects;
- **MSEdgeControl**: Stores and manages edges and lanes, and performs movement of vehicles;
- **MSVehicleControl**: Build and delete vehicles from simulation;
- **MSEdge** A road/street connecting two junctions, consisting in one or more lanes;
- **MSLane** Representation of a lane in micro simulation;
- **MSLaneChanger** Performs lane changing of vehicles;
- **MSVehicle** Representation of a vehicle in the micro simulation, vehicle type, speed, angle, lane, and so on;
- **MSVehicleType** vehicle parameters associated with a real vehicle, containing parameters such as shape, emission class, car-following model, max speed etc;
- **MSCFModel** An abstract class to be implemented by a car-following model;
- **MSRoute:** A vehicle route description.



A general overview of SUMO's microscopic simulation module implementation.

Thanks