openPASS – Content of pull-request for release 0.6

04.07.2019 – Reinhard Biegel
• **Support of scenario based simulation**
  - Static instantiation of agents based on systemConfiguration
  - Dynamic instantiation of agents based on AppConfig (systemConfiguration template), ADAS and sensors are sampled based on probabilistic profiles
  - New agent modules for dynamically instantiated agents
  - Support of conditional interference during the simulation run via EventDetectors and Manipulators

• **New SpawnPoint for scenario based spawning in the World_OSI with randomized spawning of traffic**

• **Update of World_OSI**

• **Improvement of OpenPassSlave**
  - New scheduler
  - Various new importers for scenario based simulation
• Places agents and static traffic objects in World_OSI
• Initial placement of ego and scenario agents according to Scenario.xml
• Additional agents are spawned to reach a defined TrafficVolume (Common cars)
• During runtime additional agents are spawned at start of road
• Implementation of ObservationInterface
• Responsible for adding RunStatistic information to simulation output
• Modules can access generic „Log“ method to add their own data to RunStatistics (interim solution until Publish-Subscriber is implemented)
• Logged values are divided into different groups. ExperimentConfig (in CombinationConfig.xml) defines which groups are written to the output
• Output is saved as XML-file
EventDetectors check for conditions defined in Scenario.xosc

- If all specified conditions are met, an event is inserted in the EventNetwork
- Based on the event, a corresponding Manipulator is triggered
- Manipulators have different types and can act on different scopes

**OSI use-case example:**

- EventDetector activates, if SimulationTime \( > 1 \) (i.e., in first timestep)
- This triggers the ComponentStateChangeManipulator, which sets the ComponentState of the DynamicsTrajectoryFollower in the agent with name „TF“ to Acting.
Overview of new agent modules
Sensor_Driver
- Reads all *dynamic* mesoscopic information and information about own vehicle (e.g. velocity) via the AgentInterface and forwards aggregated data to the other driver modules as SensorDriverSignal

ParametersVehicle
- Reads all *static* information about the vehicle (i.e. VehicleModelParameters) via the AgentInterface and forwards aggregated data to the other driver modules as ParametersVehicleSignal

AgentFollowingDriver
- Implementation of a simple driver. Acts based on data from the signals above. Keeps a constant velocity, or adjusts velocity to the car in front

Algorithm_Longitudinal
- Translates the acceleration wish of the driver or a vehicle component into gear and pedal positions

Algorithm_Lateral
- Translates intended lateral deviation of the driver or a vehicle component into a steeringwheel angle
• This module is an example for an ADAS in openPASS
• It implements a simple autonomous emergency braking (AEB) logic
• If the predicted time to collision (TTC) to another object is below a specified threshold, braking with constant deceleration is triggered
**Sensor_OSI**

- The Sensor_OSI module represents different types of sensors based on the OSI-groundtruth
- The output of Sensor_OSI is OSI SensorData
- Geometric2D-sensor is given as an example implementation of an OSI based sensor
- The Geometric2D-sensor acts as a 2D radar, detecting moving and stationary objects inside a circular sector
- Detection considering visual obstruction is supported

**SensorFusion_OSI**

- The SensorFusion consolidates SensorData of multiple OSI-sensors into a single SensorData structure
- The combined SensorData is forwarded to all ADAS

---

**Sensor_OSI and SensorFusion_OSI**
• The module is responsible for calculating the vehicle dynamics
• Pedal positions, steeringwheel angle and the current gear are translated into new values (velocity, acceleration and position) considering the vehicle's physical parameters
• Calculated data is forwarded as DynamicsSignal
In case a collision occurred, Dynamics_Collision takes over the calculation from Dynamics_REGULARDRIVING.

Collisions are modeled as simple inelastic collisions.

After the collision, Dynamics_Collision decelerates the agent with a constant value of 10 m/s².

Analogous to the Dynamics_REGULARDRIVING module, calculated values are forwarded as DynamicsSignal.
• This module can be used to force an agent to follow a predefined trajectory
• The agent's position is set, velocity and acceleration are calculated based on the given trajectory provided by a CSV-file
• Trajectories can either be given in world coordinates or in absolute or relative road coordinates
• Dynamics_TrajectoryFollower can either be dominant (enforcing the trajectory) or submissive (trajectory can be overruled by an ADAS)
• Since a signal of a specific type can be created by different modules simultaneously, it’s necessary to prioritize one of those signals before allowing consumption by a follow-up module

• Priority lists are used to decide which signal gets forwarded

• Examples:
  • The DynamicSignal can originate from the Dynamics_REGULARDriving, the Dynamics_COLLISION or the Dynamics_TRAJECTORYFOLLOWER. The Dynamics_COLLISION has the highest priority and the Dynamics_REGULARDriving has the lowest priority
  • Different ADAS can output an AccelerationSignal and a SignalPrioritizer is used to decide which ADAS gets prioritized
The AgentUpdater calls setter-functions of the AgentInterface for all values of the DynamicsSignal.
The Sensor_RecordState logs all basic agent based informations (e.g. agentID, velocity, position) via the ObservationLog
• **The Vehicle Control Unit (VCU) is used to configure and handle dependencies between vehicle components**

• **The responsibilities of the VCU are:**
  - Handling of all dependencies in case a component wants to activate
  - Make information about driver, Dynamics_TrajectoryFollower and other vehicle components available to each other
  - Determine the highest allowed activation state of a component and notify the affected component about this state

• To achieve this tasks, each component is assigned a maximum allowed state in each timestep. This state is of type `ComponentState`, which defines Disabled, Armed or Active as allowed states.

• Drivers can be in a state of either Active or Passive.