Scenario Modeling and Execution for Simulation Testing of Automated-Driving Systems

by

Rodrigo Barbosa de Queiroz

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

© Rodrigo Barbosa de Queiroz 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Richard Paige Professor, Dept. of Computing and Software, McMaster University.
Supervisor(s):	Krzysztof Czarnecki Professor, Dept. of Electrical and Computer Engineering, University of Waterloo.
Internal Member:	Derek Rayside Professor, Dept. of Electrical and Computer Engineering, University of Waterloo.
Internal Member:	Stephen Smith Professor, Dept. of Electrical and Computer Engineering, University of Waterloo.
Internal-External Member:	Steven Waslander Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo Professor, Institute for Aerospace Studies, University of Toronto.

Author's Declaration

This thesis consists of material all of which I authored or co-authored (see Statement of Contributions included in the thesis).

This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Rodrigo Queiroz is the sole author of this thesis, which was written under the supervision of Prof. Krzysztof Czarnecki. The development of this research resulted in the following publications:

- (i) R Salay, R Queiroz, K Czarnecki. An analysis of ISO 26262: Machine learning and safety in automotive software. SAE Technical Papers, 2018
- (ii) R Queiroz, T Berger, K Czarnecki. GeoScenario: An Open DSL for Autonomous Driving Scenario Representation. IEEE Intelligent Vehicles Symposium, 2019.
- (iii) R Queiroz, D Sharma, R Caldas, K Czarnecki, S García, T Berger, P Pelliccione. A Driver-Vehicle Model for ADS Scenario-based Testing. ArXiv preprint, 2022 arXiv:2205.02911
- (iv) S Larter, R Queiroz, S Sedwards, A Sarkar, K Czarnecki. A Hierarchical Pedestrian Behavior Model to Generate Realistic Human Behavior in Traffic Simulation. IEEE Intelligent Vehicles Symposium, 2022.

In particular, publications (ii) and (iii) contain material that is also included in this thesis, in Chapters 3, 4, and 6. These publications benefited from collaboration with Prof. Thorsten Berger, Prof. Patrizio Pelliccione, Ricardo Caldas, and Sergio Garcia from Chalmers University of Technology. Profs. Berger and Pelliccione provided extensive feedback on the ideas that went into (ii,iii) and (iii), respectively. Ricardo Caldas and Sergio Garcia mainly contributed the Behavior Tree Parser and Behavior Tree grammar and participated in the literature review and the design of the scenarios used in the evaluation. Finally, Divit Sharma and Dr. Michał Antkiewicz from the University of Waterloo helped implement parts of GeoScenario Server and its integration with the WISE ADS, as presented in Chapter 5.

Abstract

Automated Driving Systems (ADS) have the potential to significantly impact the future of ground mobility. However, safety assurance is still a major obstacle. Field testing alone is impractical and simulation is required to scale and accelerate testing. Further, it covers difficult and rare cases that are too risky to be performed on the closed-course. Evaluating a wide range of operating scenarios in simulation is essential to ensure ADS safety, reliability, and conformity to traffic regulations as the level of automation increases. In order to achieve this goal, Scenario-based testing for ADS must be able to model and simulate traffic scenarios that rely on interactions with other vehicles. Although many languages for high-level scenario modelling have been proposed, they lack the features to precisely and reliably control the required micro-simulation, while also supporting behavior reuse and test reproducibility for a wide range of interactive scenarios.

To fill this gap between scenario design and execution, this thesis proposes a Domain-Specific Language (DSL) for scenario representation, and a model for vehicle behavior in scenario design and simulation. The main research goal is to improve scenario modeling and execution for ADS testing in simulation, contributing to safety assurance in ADS development. First, we present the language GeoScenario to help researchers and engineers to develop tool-independent test scenarios, migrate scenarios between tools, and to evaluate their systems under alternative testing environments. The language is built on top of the well-known Open Street Map standard, and designed to be lightweight and extensible. Second, we propose the Simulated Driver-Vehicle Model (SDV) to represent and simulate vehicles as dynamic entities with their behavior being constrained by scenario design and goals set by testers. This model combines driver and vehicle as a single entity. It is based on human-like driving and the mechanical limitations of real vehicles for realistic simulation. The layered architecture of the model leverages behavior trees to express high-level behaviors in terms of lower-level maneuvers, affording multiple driving styles and reuse. Further, optimization-based maneuver planner guides the simulated vehicles towards the desired behavior.

Finally, our extensive evaluation shows the language and model's design effectiveness using NHTSA pre-crash scenarios, its motion realism in comparison to naturalistic urban traffic, and its scalability with traffic density. We show the applicability of SDV model to test a real ADS and to identify crash scenarios, which are impractical to represent using predefined vehicle trajectories. The SDV model instances can be injected into existing simulation environments via co-simulation.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADAS Advanced driver-assistance system

API Application Programming Interface

BT Behavior Tree

DSL Domain Specific Language

HV Human-operated Vehicle

ODD Operational Design Domain

OSM Open Street Maps

ROS Robot Operating System

SDV Simulated Driver-Vehicle

TTC Time to Collision

UDP User Datagram Protocol

UTM Universal Transverse Mercator

WGS84 World Geodesic System 1984

XML Extensible Markup Language

Chapter 1

INTRODUCTION

Automated Driving Systems (ADS) have the potential to significantly impact the future of ground mobility.[1] Practical benefits include fewer road accidents, reduced traffic congestion, higher occupant productivity, fuel savings, increased mobility for the elderly, disabled and blind, and many more [71]. ADS adoption can also have enormous economic implications. Morgan Stanley [71] estimates ADS can contribute to $1.3 trillion USD in annual savings to the US economy, and over 5.6 trillion USD globally. This estimation takes into account fuel savings, productivity gains, and savings due to crash avoidance.

However, safety assurance is still a major obstacle. In 2021 alone, the Department of Motor Vehicles of California/US reported 127 collisions involving autonomous vehicles from Waymo, Cruise, Argo AI, and other ADS developers [11]. The National Highway Traffic Safety Administration (NHTSA) in the US published the initial round of data collected after the Standing General Order issued in 2021 that requires manufacturers and operators to report certain crashes involving vehicles equipped with ADS (SAE Levels 3 to 5 [63]). The preliminary report shows 130 ADS crashes over 10 months ending in May 2022, with Waymo, Transdev, and Cruise reporting the most ADS crashes [64]. Public perception is also not favorable, indicating low consumer confidence in the capabilities of the technology. A 2020 survey commissioned by the advocacy group Partners for Automated Vehicle Education (PAVE) shows that nearly half Americans would not get in a self-driving taxi [2].

Field testing alone is impractical. RAND Corporation estimates that billions of miles driven are necessary to provide clear statistical evidence they are safer than a human [42]. In order to ensure ADS safety, reliability, and conformity to traffic regulations, we need

[1] ADS-equipped vehicles are often called Self-Driving Vehicles, Autonomous Vehicles, or simply AVs.

to evaluate a wide range of operating scenarios, and simulation is required to scale and accelerate testing. Further, simulation can cover difficult and rare cases that are too risky to be performed on the closed-course. Moreover, as the level of automation increases, more driving tasks are transferred from the human driver to the ADS, which has to deal with disturbances of the real traffic and interactions with human-operated vehicles (HVs) and pedestrians. It is imperative that scenarios for ADS testing in simulation reflect how these dynamic interactions between humans and the subject system can unfold in real traffic.

Scenario-based testing plays an important role in this process. It was originated as a black-box testing paradigm from Software Engineering where scenarios are used as a central representation that guides the testing phases [43]. Further, scenarios can also be used to support development throughout the entire life-cycle and across teams. The ISO 26262 standard that guides development of safety-critical electrical/electronic vehicle systems mandates the use of scenarios to support many phases of the development process [38].

As an example, Figure 1.1 shows a typical scenario for ADS testing based on the NHTSA crash data [54, 55]. Although simple, this scenario is ranked as one of the most frequent crash scenarios (7.2% relative frequency) of all 5.9 million police-reported crashes compiled and aggregated by the report. The ADS can encounter a similar situation as the trailing vehicle and must be able to stop in time to avoid or mitigate the imminent front-to-rear impact.

There are multiple ways in which scenarios such as the one shown are created. Engineers can manually create scenarios based on requirements and the Operational Design Domain (ODD) of the system, use expert knowledge on common traffic situations the ADS must be able to cope with, use search-based techniques to automatically generate scenarios [18], or reproduce and augment scenarios collected from naturalistic traffic [75, 58] and crash reports [54] (such as the example in Figure 1.1).

Regardless of the creation approach, two fundamental components are required for scenario-based testing: (i) a formal representation to express a scenario, and (ii) a simulation environment (the toolset) with models capable of executing such scenarios as intended by design. Multiple representations have emerged in recent years, providing a formal definition of the scenario concept and all its components: road network, weather, events, success and fail criteria, and traffic participants, such as vehicles and pedestrians and their behavior in the scenario. Examples include OpenScenario [8], M-SDL [5], and SDL [81]. Similarly, many simulators have also emerged (or were repurposed) for ADS testing. Some tools focus on models for the Ego vehicle and sensor simulation (Camera, Lidar, GPS, IMU) such as VREP [60] and Microsoft's AirSim [69]. Others support scenario simulation in some capacity such as CARLA [30], LG's SVL [12], SUMO [10], or Virtual Test Drive

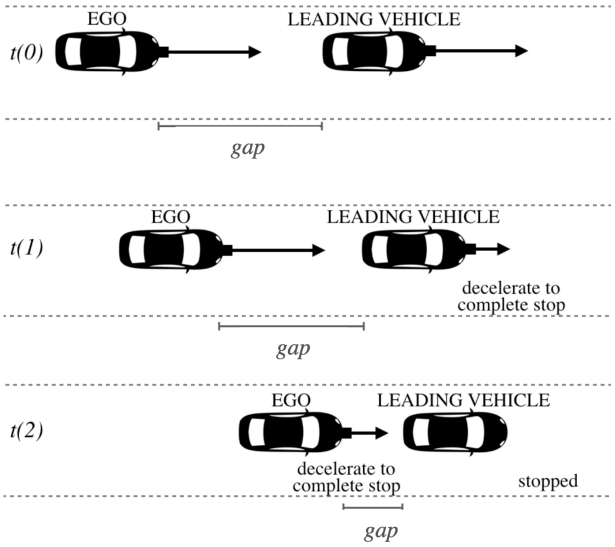


Figure 1.1: Test scenario based on typical rear end pre-crash scenario from NHTSA studies [54, 55]. At $t(0)$, both vehicles are following the same lane, while Ego (the ADS-equipped vehicle) is keeping a safe TTC (Time to Collision). At $t(1)$, the leading vehicle changes its behavior, starting to decelerate until a complete stop. At $t(2)$, Ego (the following vehicle) needs to stop in order to avoid a collision.

(VTD) [13]. They allow testers to create and execute scenarios with both Ego and other traffic participants in simulation.

However, modeling and simulation in scenario-based testing for ADS development still has major shortcomings:

- *Limited interoperability:* Many heterogeneous representations exist across stakeholders, methods, and simulation tools. For example, scenarios from early requirements are typically represented by natural language, while scenarios for simulation require executable instructions. Simulation tools either run scenarios based on a representation exclusive to their own environment (a tool-specific language), or require scenarios to be programmed from scratch by testers using traditional computer code (e.g., Python, C++) and a tool-specific API. Migrating scenarios between tools requires code translation, and it is particularly difficult when constructs are highly coupled with exclusive functions from the tool. Further, it requires extra effort to manage and synchronize multiple versions of the same scenario to satisfy each tool.

- *Limited consistency*: Existing solutions yield inconsistent results across tools and executions. When a scenario representation is adapted (i.e.,translated), the differences in the fundamental structures and assumptions of the simulation capabilities can introduce unintended differences and change how the scenario unfolds, affecting consistency. Scenarios can also be represented with varying levels of abstraction, and not always encode the necessary detail for execution in simulation. This is particularly difficult for the behavior of dynamic agents such as HVs. The scenario execution relies on the underlying simulation model to handle the behavior, but this model is often not formally defined outside of the code itself. The potential mismatch between what is specified in the scenario and the actual execution leads to inconsistency and compromises test reproducibility.

- *Limited traffic realism*: At the level of scenario representation, tool-independent languages are typically limited to relatively simple models for vehicle behavior in traffic. For instance, specifying "what a vehicle must do" and "where it must be" in a particular scene, but without detailing "how" it moves. They do not encode the necessary detail to guide the simulation of the vehicle in a complex and dynamic environment as the traffic. Consequently, the simulation tool is required to implement the actual vehicle behavior, either by translating the high-level definition to the vehicle simulation model, or by forcing the change in vehicle state while disregarding the limitations of a real vehicle in traffic. This mismatch leads to unrealistic vehicle motion, and compromises the validity of test results if Ego-to-HV interactions do not represent the real traffic. While there are efforts in improving realism in traffic simulation, they are built for simulated agents to drive independently without collisions (intelligent agents forming a Mass AI). They have limited controllability, in contrast with the goals of scenario-based testing.

- *Limited coverage*: Limitations of the scenario representation or the underlying simulation models constrain the range of scenarios that can be executed, or the level of accuracy in which a scenario can be simulated. For instance, if either the representation or the simulation lack the ability to encode and execute the nuances of the human driving, scenarios will not be able to realistically reproduce situations from the real traffic. Further, when the limitations or differences between representations arise, they lead testers to use simple structures as the "lowest common denominator". For instance, using pre-defined trajectories (PDTs) to represent the vehicle behavior in traffic. Such limitations constrain interactions between Ego and the traffic, and thus the range of scenarios that can be expressed and executed.

The main research goal of this thesis is to improve scenario modeling and execution for ADS testing in simulation, contributing to safety assurance in ADS development. In particular, this thesis focus on two main aspects: the scenario representation, and a model for vehicle behavior in scenario design and simulation. This research has two parts:

- *A DSL for Autonomous Driving Scenario Representation*: In this first part, we propose GeoScenario as a Domain-Specific Language (DSL) for scenario representation. Since typical scenarios for testing self-driving cars aim at reproducing real traffic situations, they are similar by definition and must be able to offer the same set of core features. The language is designed to cover these features and help researchers and engineers to develop tool-independent test scenarios, migrate scenarios between tools, and evaluate their systems under alternative testing environments, including both computer simulation and closed course. The language is built on top of the well-known Open Street Map standard, and is designed to be simple and extensible. We apply GeoScenario in the simulation infrastructure of an ADS project, and demonstrate the language usage in practice to test a self-driving car software stack in simulation. The language is the foundation for the Simulated Driver-Vehicle Model.
- *A Model for Simulated Driver-Vehicle Behavior in Scenario-Based Testing*: In the second part, we propose a model to express and simulate realistic HV behavior in ADS scenario testing, while affording high expressiveness, execution accuracy, scalability, and reuse. We refer to our model as GeoScenario Simulated Driver-Vehicle Model (or simply SDV). This model extends the base scenario-definition language GeoScenario from the first part with HVs as dynamic agents in both scenario representation and simulation execution. The model is based on human-like driving and the mechanical limitations of real vehicles for realistic simulation. We use a layered architecture and leverage behavior trees to express high-level behaviors in terms of lower-level maneuvers, affording multiple driving styles and reuse, and covering a diverse range of scenarios.

Methods for scenario generation (knowledge-driven or data-driven), scenario coverage, and metrics for evaluating scenarios are out of the scope of this thesis. However, these methods can benefit from our approach, since a better model for scenario representation allows faster scenario development with behavior reuse and consistency across tools. The proposed models also improve accuracy and realism when running scenarios in simulation, allowing testers to cover highly interactive scenarios and expand

the range of scenarios that can be generated. Particularly, scenarios that rely on Ego-to-HV and multi-vehicle interactions and that are too difficult to simulate with current methods can benefit the most from our approach.

After implementing the language parser and execution model in a simulation toolset, we evaluate GeoScenario and the SDV model in terms of scenario development effectiveness, realistic vehicle motion, practical applicability for scenario-based ADS testing, and finally scalability with traffic density. The following research questions guide our evaluation:

- **RQ1**: Can realistic and interactive scenarios for ADS testing be effectively modeled and executed via GeoScenario and SDV models?
- **RQ2**: Can SDV models generate realistic vehicle motion?
- **RQ3**: Can using GeoScenario and SDV models improve the effectiveness of scenario-based testing of a real ADS?
- **RQ4**: How does the model performance scale with traffic density?

1.1 Research Contributions

In summary, the main contributions of this work towards advancing scenario modeling and execution for ADS testing in simulation are:

- (i) GeoScenario as an open and tool-independent DSL for ADS scenario-based testing.
- (ii) A Simulated Driver-Vehicle model to support design and execution of driver behavior and vehicle interactions in scenario simulation.
- (iii) The Evaluation with a series of experiments that demonstrate how scenarios from the NHTSA Crash Typology can be designed and simulated with GeoScenario, the applicability of our method in practice with a real ADS, and how our models can replicate the human-driving behavior from naturalistic traffic.
- (iv) An open catalog of executable test scenarios, designed with GeoScenario and SDV model, and tested with a real ADS platform.
- (v) An open-source toolset to support scenario design and execution that includes the reference implementation and can be easily integrated with existing simulation tools and any ADS software stack for testing.

1.2 Outline

The thesis is organised as follows. Chapter 2 covers the necessary background related to this research, alternative DSLs for scenario representation and how vehicle behavior and Ego-to-vehicle interactions are handled in existing solutions. Chapter 3 presents GeoScenario, including the fundamental concepts of the language and the design decisions governing its creation. In Chapter 4 we introduce the Simulated Driver-Vehicle Model (SDV), detailing its layered architecture and how the vehicle motion is generated. In Chapter 5 we detail the reference implementation, its integration with alternative simulation environments, and performance considerations. In Chapter 6, we perform an extensive evaluation to test the effectiveness of our approach. Finally, Chapter 7 presents the conclusions and future work.

Chapter 2

Background and Related Work

This chapter provides the necessary background for this thesis and an overview of the related work. This research is primarily related to two major topics: Scenario Representation and Driver Behavior, discussed in Section 2.4; and Models for Traffic Simulation, discussed in Section 2.5. Finally, Section 2.7 introduces Behavior Trees and their classical formulation from the literature. These concepts will be revisited in Chapter 4 when the Behavior Tree formulation is adapted as part of the SDV Model.

2.1 Basic Terminology

SAE provides a common terminology for automated driving in the "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems - J3016" [63] (originally released in 2014, and last updated in 2021). The standard introduced the famous SAE Automation Levels spanning from Level 1 (no automation) to 5 (full automation), describing the roles of the driver and the system for each level based on system capabilities. The main distinction arise between 2 and 3, where the latter requires driving automation system that monitors the driving environment and performs the entire Dynamic Driving Task while a human capable of driving the vehicle is responsible for the fallback task. From this level and higher (3 to 5) the driving automation system can be defined as Automated Driving System (ADS). We summarize the key terminology for this thesis, and refer to the standard for a full description:

- Automated Driving System (ADS): The hardware and software driving the vehicle (i.e., performing the dynamic driving task), regardless of whether it is limited to a

specific operational design domain. This term is only applicable to systems in levels 3, 4, or 5.

- Dynamic driving task (DDT): The task of driving the vehicle. Includes the operational tasks (steering, braking, accelerating, monitoring the vehicle and roadway), the tactical tasks (responding to events, determining when to change lanes, turn, use signals, etc.) and monitoring the environment (including other road users) to recognize the need for a response.
- Operational Design Domain (ODD): The operation environment and conditions for which the ADS is designed to handle. For example, geographical restrictions (geofencing), time-of-day, the presence or absence of certain traffic.
- Fallback: The response by the user or by the ADS to operate the vehicle after a failure or when vehicle exists the ODD to achieve a minimal risk condition (MRC).

Other key terms used throughout the thesis are listed in Table 2.1.

2.2 Scenario-Based Testing

The term *scenario* is used inconsistently in the literature [40, 34, 74, 36]. Although its usage varies depending on the discipline, the main components are similar: actors, background information on actors and assumptions about the environment, goals, actions and events [36]. In the remainder, we rely on Ulbrich et al. [74] who analyze the concept of scenario (and other related terms) across multiple disciplines and propose a consistent definition based on requirements for testing automated vehicles:

" *A scenario describes the temporal development between several scenes in a sequence of scenes. Every scenario starts with an initial scene. Actions and events as well as goals and values may be specified to characterize this temporal development in a scenario. Other than a scene, a scenario spans a certain amount of time.* " (Ulbrich et al., 2015)

" *A scene describes a snapshot of the environment including the scenery and dynamic elements, as well as all actors' and observers' self-representations, and the relationships among those entities* " (Ulbrich et al., 2015)

This temporal development from the Initial Scene is illustrated in Figure 2.1. From a single initial scene, a scenario can evolve through alternative paths leading to different

Table 2.1: Summary of key Terminology

Term	Description
Ego	The entity representing the ADS-operated vehicle. In the context of simulation, the term is used to refer to the simulated vehicle operated by the ADS under test.
Scenario	The temporal development between several scenes in a sequence of scenes (more in Section 2.2)
Test Scenario	A scenario used for testing. The term is used interchangeably with just "scenario" in this thesis, since we focus on ADS testing.
Scene	A snapshot of the environment, including the scenery and dynamic elements, actors, observers, and the relationships among those entities (more in Section 2.2)
Actor	An element of a scene acting on its own behalf. They include dynamic agents and traffic signals.
Dynamic Agent	An actor in a scenario that have the ability to move. For example, vehicles or pedestrians.
Scenery	All geo-spatially stationary elements in a scene. For example, the road network, traffic signs, and static objects.

scenes. Each path is by definition a single individual scenario. A scene can be interpreted as a snapshot of the environment, and is composed by the scenery (stationary elements), dynamic elements (elements that have the ability to move, or whose state changes within the scene), actors, and observer self-representation (attributes and states).

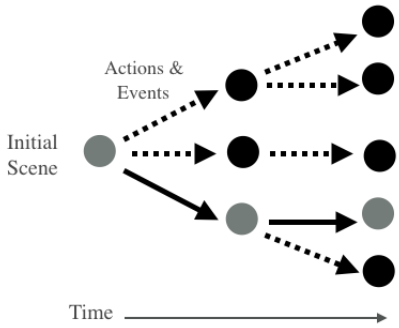


Figure 2.1: A Scenario (solid line) as a temporal sequence of actions&events (edges), and scenes (nodes). Adapted from [74]

The *scenario-based design paradigm* considers scenarios as a central concept to support the development of complex systems throughout the entire lifecycle, from helping to derive initial requirements to validating the system during the testing [36]. Kaner et al. [43] define *scenario-based testing* as the dominant paradigm of black-box testing, where scenarios are used to check how the system copes with both nominal and off-nominal situations. In the automotive context, ISO 26262 [38] and ISO/PAS 21448 [39] guide the development of safety-critical electrical/electronic vehicle systems and mandate the use of scenarios as part of validation activities.

Scenarios can be described at several levels of detail and expressed using formal, informal or semi-formal notations [36]. Menzel et al. [51] propose three such levels within the ISO 26262 systems engineering process (see Figure 2.2):

- (i) functional scenarios, being high-level natural language descriptions in the concept phase.
- (ii) logical scenarios, being semi-formal models with state space parameters and their ranges in the development phase.
- (iii) concrete scenarios, represented in an executable format with concrete values in the test phase.

In this work we focus on levels (ii) and particularly (iii), since they are closer to the level of detail required for scenario execution in simulation.

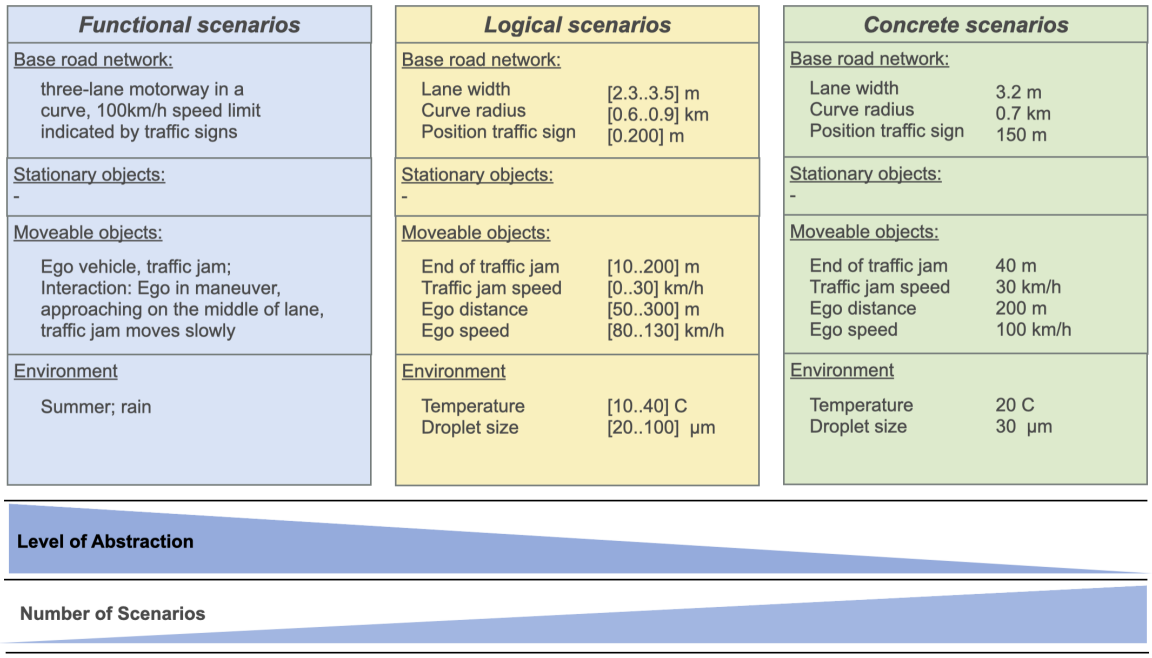


Figure 2.2: Scenario abstraction levels as defined by Menzel et al. [51]

2.3 Scenario Design and Generation

Researchers and engineers design scenarios based on expert knowledge and the common traffic situations the ADS must be able to cope with, or by reproducing and augmenting scenarios collected from traffic databases. For example, CommonRoad [19], a benchmark for motion planners, extracts scenarios from NGSIM data [58].

A scenario can also be systematically generated to achieve specific test goals, e.g., lead the system to trigger a certain behavior such as an emergency maneuver, or find a critical situation leading to a crash. For example, Abdessalem et al. [17, 18] use evolutionary optimization methods combined with surrogate model learning to find crash scenarios. Given a parameterized scenario space, the evolutionary search produces subsequent generations of parameter values with increasing criticality based on how the system performs under simulation. Similar methods are also used to test autonomous parking systems [24].

2.4 Scenario Representation and Driver Behavior

Multiple tool-independent DSLs have emerged in recent years, providing a formal definition of scenario structure, behavior, test conditions, and pass/fail criteria to support scenario-based design and testing in simulation. The goal is to offer a uniform representation and semantics across methods and tools. The scope and structure of each language vary, but fundamentally they all define how vehicles behave in traffic and orchestrate interactions with Ego that must be executed by a simulation tool during the test. We focus our discussion on how some of the prominent languages specify this behavior.

OpenScenario 1.0 [8] is a standard managed by the Association for Standardization of Automation and Measuring Systems (ASAM). The format describes dynamic content in driving simulation applications in combination with OpenDRIVE [7], which specifies the road structure. It covers traffic and driver behavior, weather, environmental events and other features. It includes the description of a driver, but there is no model for driver behavior in any form other than "road following." The standard also does not contain maneuver models or a vehicle model. Maneuvers are described in terms of *actions* (e.g., change the vehicle's position or speed), or trajectories (defined as a polyline, clothoid, or spline).

The Measurable Scenario Description Language (M-SDL) [5] expands the concepts of OpenScenario 1.0. The language uses *modifiers* to change the behavior of the agents similarly to *actions* from OpenScenario 1.0. It introduces parameter variability (a range instead of a single value) along with constraints to narrow down values and connect multiple parameters (e.g., velocity of vehicle A is between 10 and 20 m/s and less than vehicle B). The language represents the vehicle behavior at the logical abstraction level and supports generating concrete scenarios by picking random values while obeying the constraints (see Figure 2.4 for an example). The format is planned to be merged with OpenScenario 2.0.

Scenic [33] is a probabilistic programming language to define and sample scenes of a scenario for testing ADS. The generated scenarios can be used to test the system under several situations, produce a training dataset, or support debugging. It allows the definition of vehicles and actors, locations as coordinates, missions and goals, triggers to alter the ego-vehicle's behavior under certain conditions, and constraints. An example of a constraint is that the ego-car must be within the 30° view of a second car while driving. The language is used to output scenes consisting of the assignment to all the properties of each object defined in the scenario, plus any global parameters based on the inputs explained above.

Scenario Description Language (SDL) [81] proposes a two-level abstraction approach to scenario representation. In SDV level 1, a textual description of the scenario at a

```
scenario traffic.overtake:
    v1: car # The first car
    v2: car # The second car
    p: path

    do parallel:
        v2.drive(p)
        serial:
            A: v1.drive(p) with:
                position([5..]m, behind: v2, at: start)
                lane(same_as: v2, at: start)
                lane(left_of: v2, at: end)
            B: v1.drive(p)
            C: v1.drive(p) with:
                position([30..]m, ahead_of: v2, at: start)
                lane(same_as: v2, at: end)
```

Figure 2.3: Measurable Scenario Description Language (M-SDL) from Foretelix. Example available in the official documentation [5]

higher level of abstraction can be used by regulators or system engineers. Whereas in SDV level 2, a formal machine-readable language can be used by simulators. Scenarios can be transformed from level 1 to 2 with the inclusion of additional details, and the reverse transformation from level 2 to 1 is achieved by abstracting details.

SceML [66] is a graphical framework for Scenario-Based Testing that can be used to create or visualize scenarios. It supports different abstraction levels, modularity and the ability to reuse substructures. This representation is parsed into OpenScenario format to generate a concrete representation before the scenario can be simulated.

A common trait amongst all representations is that they are primarily declarative languages. They define "what" must happen in a scenario during key events without specifying "how." This approach relies on external models running in simulation to handle the execution, requiring mapping the scenario representations to these models.

2.4.1 Road-Network Representation

In the context of scenario representation, a map format can be used to express the features of the Road Network. Open Street Map (OSM) [6] is a well-known collaborative project to create and publish free maps using an open XML format. However, OSM and other general map standards do not contain detailed information about the road topology at the lane level. Therefore, they are not suitable to be used in a scenario representation as is.

Lanelets [22] is an open extension of the OSM format specifically to support Road Net-

work representation for automated vehicles. By definition, *lanelets* are *"atomic, interconnected, drivable road segments geometrically represented by their left and right bounds"* [22]. The bounds are encoded by an array of OSM nodes forming a polyline. Together, they compose the *Lanelet Map*. With lanes represented by road-segments with precise boundaries, lanelets can be used to compose the Road Network of a scenario.

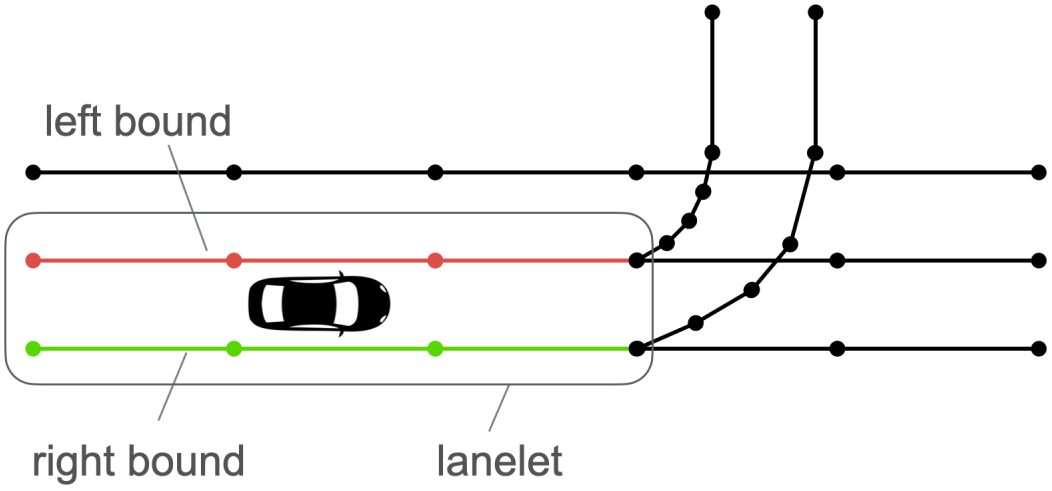


Figure 2.4: Graphical representation of a lanelet. Concept image adapted from documentation [22]

2.5 Models for Traffic Simulation

Traffic simulation has a wide range of applications and can be used to generate the motion of vehicles at various levels of detail. Macroscopic traffic models describe vehicle motion and interaction in terms of flow and density, and are mainly used for large scale simulation over a road network [68]. Since they are not suitable for street-level motion and interactions between vehicles, they cannot be used for ADS testing.

In contrast, microscopic traffic models can generate vehicle motion and interactions at the individual vehicle level at the cost of limited scalability [25]. They are able to encode simple rules that allow a vehicle to follow way-points or the structure of the road, avoid frontal collisions by alternating between driving and stopping, and perform maneuvers

triggered by conditions [35, 45, 30]. However, while capturing this reactive behavior, they usually lack enough detail to simulate complex interactions between the vehicle under test and other road users in realistic conditions. For example, they often use simplistic motion limited to a constant velocity throughout a maneuver and disregard the physical limitation of a real vehicle. They also cannot represent complex interactions, such as vehicles responding to merge attempts, using the available road space to navigate around obstacles, or skillfully navigating an intersection with multiple influencing factors (e.g., vehicles, pedestrians, and traffic regulation). The supported behavior is rigid and it is hard or impossible to encode the fine-grained details that replicate human driving.

Some micro-models target a particular maneuver, for example, a lane-change model encoding the accelerating/decelerating behavior based on surrounding vehicles [53], or the driver's decision and conditions that trigger the maneuver [82]. Whereas these models better capture details at the maneuver level and allow testers to cover a range of parameters, they are suitable for testing specific functions and subsystems (for instance, testing the ADS emergency break) in a very constrained environment. They do not cover the complexity of the full driving task required for scenarios in system-level testing. Attempting to combine multiple maneuver-specific models into a simulated agent would be challenging, since every model has its own set of assumptions and constraints.

A different approach is to learn models directly from data. Krajewski et al. [49] build a lane-change model by using unsupervised learning to extract primitive attributes from lane changes observed in the highD dataset [48]. The resulting model can then be used to generate synthetic lane change maneuver trajectories in new scenarios. The main limitation in a purely data-driven approach is the inherent bias in the data used to built the model. While most available data-sets cover common situations, driver mistakes and safety critical scenarios are rarely captured in such data sources. Also, they can capture the diversity of driving styles in one road environment, but are difficult to generalize to other environments. TrafficSim [72] uses a hybrid approach to build a model by learning from naturalistic data and also encoding common-sense rules to guide the driving task. This hybrid approach shows promising results in imitating the human-driving and its diversity of driving-styles, while still reacting to traffic. However, agents are not fully controllable and cannot be adapted to new scenarios by freely assigning new goals or styles based on a new scenario design.

Overall, traffic simulation models are built for simulated agents to drive independently without collisions. As a result, they tend to limit the controllability by the tester. This is a contrast with scenario representation models that aim at providing the expressiveness to model a multitude of scenarios (and often disregarding how exactly each step will be performed). For scenario-based testing, the simulation model must serve the scenario goals.

If the evolution between scenes is not controllable, and the agents are not guaranteed to reach the target situation (as specified by parameters, such as a time gap for a maneuver), even the most realistic traffic simulation will not be suitable. Thus, scenario-based testing requires expressiveness, controllability, and realistic behavior.

2.6 Simulation Tools

CARLA [30] is an open-source simulator created to support development, training, and validation of automated driving vehicles. Built with Unreal Engine, it provides a library of digital assets (urban layouts, roads, buildings, traffic props, vehicles) and a suite of sensor simulation (GPS, camera, lidar). It supports environmental conditions (such as time-of-day, weather) and scenario simulation with dynamic actors. Scenarios can be defined using a Python interface, or with OpenScenario (although coverage is limited). Launched in 2017, the simulator gained substantial popularity with the research community.

Several other simulators are briefly mentioned next. Simulation of Urban MObility (SUMO) [10] is an open source microscopic and continuous traffic simulation package designed to simulate large road networks in low fidelity. The project is mainly developed by the Institute of Transportation Systems at the German Aerospace Center. SVL [12] is an open-source high fidelity simulator built with Unity Engine. It was created and maintained by LG Electronics America R&D Lab until January 2022 when development was suspended. Microsoft's AirSim [69] is a simulator based on Unreal Engine, initially built for Drone simulation and later adapted to support wheeled vehicles. Notable proprietary and closed-source simulators: Virtual Test Drive (VTD) [13], Metamoto, and Cognata.

2.7 Behavior Trees

Behavior Trees (BTs) is a control architecture that emerged from the gaming industry to structure the task switching in autonomous virtual agents. Initially, they were developed as a way to increase modularity and reuse in the control structures of non-player characters (NPCs), but quickly expanded to robotics and other fields of research. For example, they have been used to structure a robotic arm task for object manipulation [21].

They are an efficient way of structuring complex systems that are both reactive and modular: reactive as the ability to react quickly and efficiently to changes, and modular as the extent to which the components of a system can be broken down into individual building

blocks and recombined. When complexity of NPC behavior increases, it is essential to be able to work with individual components rather than the whole behavior at once.

The tree-like structure of BTs conveys a hierarchical understanding of how composition operators coordinate elemental behaviors to perform the desired overall behavior. This explicit organization facilitates the process of creating a behavior and understanding the conditions leading to a task. Bagnell et al [21] highlight these properties in his work using behavior trees in robotics:

"Real-time introspection facilitates the development of behavior trees by providing a way to observe what the system is executing at any time. By seeing which behaviors are running, which have succeeded and which have failed, one can quickly determine how the overall task is performing." (Bagnell et al, 2012, pg.3) [21]

Finite State Machines (FSMs) have been the standard choice to design task-switching structures, but the Behavior Tree design aims to address FSM shortcomings with improved modularity, reusability, scalability, and readability [27]. From a theoretical point of view any Behavior Tree can be described by a FSM.

The expressive, modular, and explicit representation of BTs makes them a suitable candidate for representing driver behavior in test scenarios and we use them in our model. To the best of our knowledge, the use of behaviour trees to represent driver behavior in scenario representation for ADS testing in simulation is still unexplored. There are several Behavior Tree implementations. In this Section we describe the classical formulation, and in Chapter 4 we describe how we adapt and integrate BTs to our model.

Classical Structure

In its classical formulation, a BT is a directed rooted tree with parent and child relations. The internal nodes of the tree are the *control nodes* and leaf nodes are the *behavior nodes*. Control nodes must have at least one child. In some implementations, a binary tree can be used and the nodes are limited to two children.

The node without a parent is the root node, where execution starts with a signal that propagates to the remaining nodes by traversing the tree from parent to children (depth-first search) and enables the ordered execution of a node at a certain frequency (tick). Graphically, they are typically represented with the children below the parent node (see Figure 2.5). When a node receives a tick, it can be executed and returns a status to the parent: (i) *success* if it achieved its goal, (ii) *failure* if it did not succeed, or (iii) *running* if the task is in progress. Nodes that did not execute remain in a *neutral* status.

Control nodes: (or operators) are responsible for coordinating the execution of their children nodes. In the classical architecture there are three operators: the *fallback*, *sequence*, and *parallel*. We describe each operator and their respective algorithms from Colledanchise and Ogren [27] as follows.

The *fallback* operator resembles the logic operator *or*, but with a short-circuit semantics. This node commands a sequential execution of its children, left-to-right, and returns success immediately when a child succeeds; otherwise it executes the next child. It returns failure when none of the children succeed (Figure 2.5).

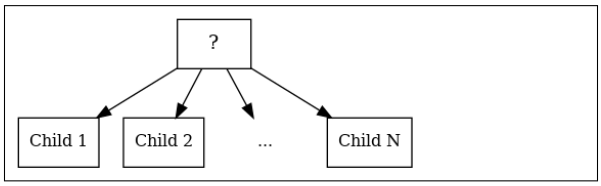


Figure 2.5: Behavior Tree with Fallback node

Algorithm 1 Fallback node pseudocode (assuming N children)

```
1: for i ← 1 to N do
2:     childStatus ← Tick(child(i))
3:     if childStatus = Running then
4:         return Running
5:     else if childStatus = Success then
6:         return Success
7:     end if
8: end for
9: return Failure
```

The *sequence* operator resembles the short-circuit logic operator *and*. This node also commands a sequential execution of its children, left-to-right, but returns failure immediately when a child fails. Otherwise, it executes the next child. It returns success when all of the children succeed (Figure 2.6).

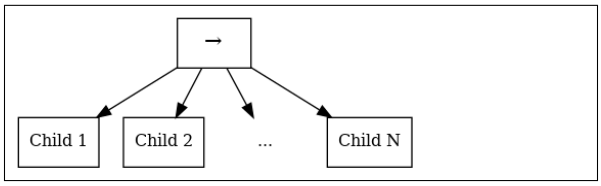


Figure 2.6: Behavior Tree with Sequential node

Algorithm 2 Sequence node pseudocode (assuming N children)

```
1: for i ← 1 to N do
2:     childStatus ← Tick(child(i))
3:     if childStatus = Running then
4:         return Running
5:     else if childStatus = Failure then
6:         return Failure
7:     end if
8: end for
9: return Success
```

Last, the *parallel* operator commands the execution of all children at the same time (or in sequence with no interruptions when parallelism is not possible). The rule for success or failure can be defined by alternative policies. They can return success when all children succeed, or when a given threshold of M children succeed (Figure 2.7).

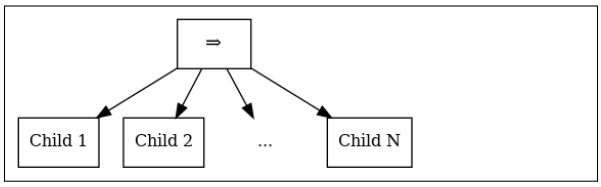


Figure 2.7: Behavior Tree with Parallel node

Algorithm 3 Parallel node pseudocode (assuming N children and success threshold M)

1: **for** $i \leftarrow 1$ to N **do**
2: $\quad childStatus \leftarrow Tick(child(i))$
3: **end for**
4: **if** $\sum_{i:childStatus(i)=Success} 1 \geq M$ **then**
5: $\quad$ **return** Success
6: **else if** $\sum_{i:childStatus(i)=Failure} 1 > N - M$ **then**
7: $\quad$ **return** $Failure$
8: **end if**
9: **return** $Running$

Behavior nodes: They are responsible for encoding domain-specific tasks, which BTs then compose into the overall desired behavior. These nodes are the interface to the concrete low-level behaviors. A behavior node returns success when its task succeeds, or 'running' while the task is under execution, or failure when the task fails. The expressive, modular, and interpretable representation of BTs makes them suitable for representing driver behavior in test scenarios. Table 2.2 summarizes all node types and status.

Table 2.2: Behavior Tree Node Types

Node Type	Symbol	Succeeds	Fails	Running
Fallback	?	if one child succeeds	if all children fail	if one child returns running
Sequence	$\rightarrow$	if all children succeed	if one child fails	if one child returns running
Parallel	$\Rightarrow$	if $>= M$ children succeed	if $> N - M$ children fail	else
Action	text	upon completion	if impossible	during completion
Condition	text	if true	if false	Never

Chapter 3

GeoScenario: An Open DSL for Autonomous Driving Scenario Representation

This chapter introduces GeoScenario as a Domain Specific Language for scenario representation in the context of ADS testing in simulation. We introduce the concept of a common scenario definition language, discuss the design decisions governing the language creation, detail the language architecture and how key components are used to compose a full scenario, and demonstrate its applicability in practice by integrating with an ADS software stack testing infrastructure. The DSL presented in this Chapter is the foundation for the Simulated Driver-Vehicle Model introduced in Chapter 4.

3.1 Introduction

Scenarios can support the ADS development throughout the entire life-cycle and across different teams, from helping to derive initial requirements to validating the system in computer simulation and coordinating tests in closed-course [20]. However, many heterogeneous representations exist across stakeholders, methods, and simulation tools using formal, informal or semi-formal notations. For example, scenarios from early requirements are typically represented by natural language, while scenarios for simulation require executable instructions. While multiple simulators for ADS testing can be used (see Section Simulation Tools 2.6), they typically require scenarios to be programmed from scratch by

testers using traditional computer code (e.g., Python, C++), or tool-specific APIs (in some instances, proprietary and closed-source). Some simulators support scenario representation with higher level of abstraction, but they are typically exclusive to their own environment (a tool-specific language). In order to design and run scenarios, engineers need to learn tool-specific languages or program simulated traffic from scratch. The learning curve increases, and migrating scenarios between simulation environments requires extra effort. Scenario translation between tools is particularly difficult when constructs are highly coupled with exclusive functions from the tool's API. Further, it requires extra effort to manage and synchronize multiple versions of the same scenario to satisfy each tool.

Since typical scenarios for ADS testing aim at reproducing realistic traffic situations, they are similar by definition and must be able to offer the same set of core features. Thus, the scenario representation can be harmonized. A well-designed, tool-independent domain-specific language (DSL) that is expressive enough to cover these features has the potential to help researchers and engineers to develop tool-independent scenarios, migrate them between different tools, and evaluate their systems under alternative testing environments, including both computer simulation and closed course. There are multiple potential benefits from a common definition:

- Provides a formal definition of structure, behavior, test conditions and pass/fail criteria.
- Helps engineering test scenarios using a formal structure rather than natural language.
- Provides consistency in scenario representation across all development phases, from requirements to validation, supporting the scenario-based design approach.
- Improves requirements understanding with scenarios formally designed since early stages of development.
- Ensures same understanding of scenarios between different stakeholders, for example, regulators and manufactures.
- Allows engineers to share and reuse scenarios between teams, companies, or research groups.
- Facilitates the migration of scenarios between simulation and testing tools.
- Promotes open databases sharing catalogs of scenarios with a unified format.

- Alleviates the learning curve to users of simulation tools, avoiding verbose scenario programming or learning many tool-specific languages.
- Provides a better understanding of the testing environment capabilities based on the level of compatibility with the language and a unified set of interpretation rules.
- Fosters the development of common libraries to parse scenarios, promoting similar interpretation and feature-set across tools.
- Helps people involved in tests to understand and coordinate a scenario before its execution, for example, a safety driver and a test engineer.

We propose GeoScenario as a DSL for scenario representation. In contrast to a general-purpose language, a Domain Specific Language [31] is a language specialized to a particular application domain. The target application domain is scenario-based testing in the context of ADS validation. GeoScenario is a modelling language, used to model scenarios. The model expresses a scenario in a structure that is defined by a consistent set of rules in the specification that must be used to interpret the meaning of the structure and execute scenarios in simulation.

GeoScenario is a textual language, serialized in XML file format with *'.osm'* file extension. Visual tools are encouraged to facilitate the scenario design, since the domain is based on geographic reasoning over a structured road. As a textual language it relies on standard keywords following a structure. GeoScenario represents both structure and behavior. The structural part is static, and entails the components that are part of the scenario (vehicles, pedestrians, the road network). The behavior is dynamic, and determines what must happen in the scenario during execution, and how the different actors must interact.

In the language design process, we identify relevant elements of the problem domain that compose typical test scenarios and need to be formally defined and executed in simulation testing. The language is built on top of the well-known Open Street Map standard, and designed to be simple and extensible. Additionally, we provide a tool-set to easily design and validate scenarios using our DSL. We apply the DSL to the simulation infrastructure of the WISE ADS Project [15], demonstrating its applicability in practice, and we provide, in addition to the reference implementation, an open catalog of test scenarios for the research community.

3.2 Designing a scenario language

In this section we discuss essential requirements for a well-designed scenario language for testing automated vehicles, and how we address them in GeoScenario. We identify key elements that compose a scenario, discuss the main scenario design approaches they must support, and the basic principles we follow to make the language practical.

3.2.1 Supporting Test Design

The ISO 26262 standard for functional safety [38] provides a framework based on the V-model as a reference to guide all development phases of safety-critical electric/electronic vehicle systems. According to the standard, scenarios are used to support the development process, from requirements to the test phase by supporting test cases. Further, ISO/PAS 21448 [39], an extension of ISO 26262, mandates the use of scenarios as part of validation activities.

Scenarios can be created using different approaches: designed by experts based on functional requirements and designs and hazard analysis, reproducing or augmenting situations collected from traffic data, or a combination of the two mentioned methods (for example, designing manual scenarios and extracting primitives from naturalistic databases, or systematically generated to find critical scenarios (see Background, Section 2.3). A scenario language should be able to support all approaches. It must be simple and human readable, yet be able to represent precise trajectories collected from traffic data, support input space exploration from methods generating scenarios, and also support unknown stochastic behaviour for sampling methods. GeoScenario is designed to support all these approaches.

3.2.2 Scenario Orchestration

When dynamic elements in a scenario follow pre-defined paths, we assume a deterministic evolution from the initial scene. However, when the ADS is responsible for the Ego's driving mission, a scenario can evolve to alternative scenes, and its execution becomes nondeterministic. Scenarios described in CommonRoad [19] are clear examples of this challenge. Dynamic elements are defined as time-discrete states of a trajectory containing position and orientation over time. After slicing NGSIM data in different scenarios, one vehicle is selected to represent the Ego, while the remaining vehicles are selected as dynamic elements with their original trajectories. The challenge arises the moment Ego starts to perform differently from the original vehicle (by different route, velocity or maneuver). The

scenario then evolves to a different situation. This is a natural limitation of any scenario directly reproduced from traffic data. Consequently, a model for scenarios must be able to orchestrate the evolution between scenes with a flexible language for Actions & Events.

A different approach is to specify intelligent dynamic agents making decisions, behaving like human drivers and pedestrians, and reacting to every other traffic agent (including the Ego). However, this brings the challenge of modeling complete and realistic behavior of traffic agents.

We design GeoScenario to provide ways of reproducing trajectory data, but also create mechanisms to orchestrate its evolution under different conditions (time and space), allowing engineers to carefully craft scenarios that explore controlled situations. Dynamic models of agent behavior and maneuvers are introduced in Chapter 4.

3.2.3 Basic principles

We design GeoScenario using the following basic principles. (i) *Reuse*: Leverage existing open formats to build a new language on top of well-known and used structures. With this approach, existing tools can be reused to support our new language with only minor adjustments. (ii) *Simplicity*: The language is simple enough to be human readable when simple scenarios are modeled. Tools are encouraged to support complex scenarios (e.g., with multiple agents) and facilitate geographic reasoning. (iii) *Coverage*: It is able to express the main components of a scenario. (iv) *Extensibility*: It can be easily extended with new features and specializations of its standard components. (v) *System independence*: It supports test cases for different ADS designs, operating on different levels of automation. (vi) *Tool independence*: It can be interpreted and executed by alternative simulation and test environments. (vii) *Executability*: It can express concrete scenarios that can run in simulation without an additional language.

3.3 GeoScenario Architecture

GeoScenario is developed to express a scenario in a formal language, following the requirements discussed on Section 3.2. The format is XML-based and built on top of the OSM standard. The main components include: Ego start position and goals, a road network, agents (vehicles and pedestrians), paths, and triggers & actions. Additional elements are omitted for simplicity, but they are available in our full specification. Figure 3.1 shows a meta-model (a.k.a., syntax model) of our components (the Route will be introduced in

Chapter 4). Figure 3.2 illustrates a sample scenario with the main components in place. In the next section we describe how all those elements work and interact.

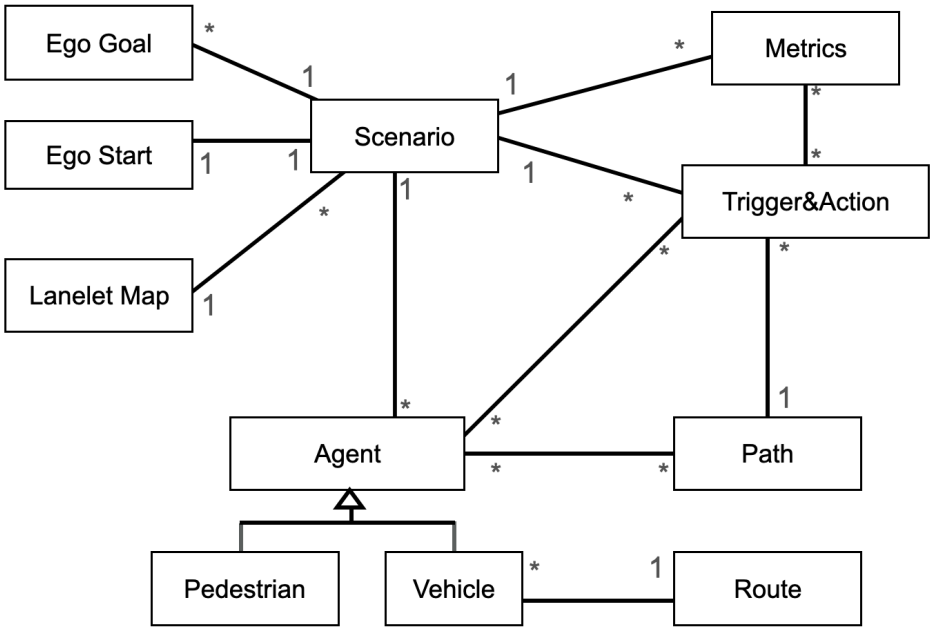


Figure 3.1: GeoScenario meta-model (class diagram notation from UML).

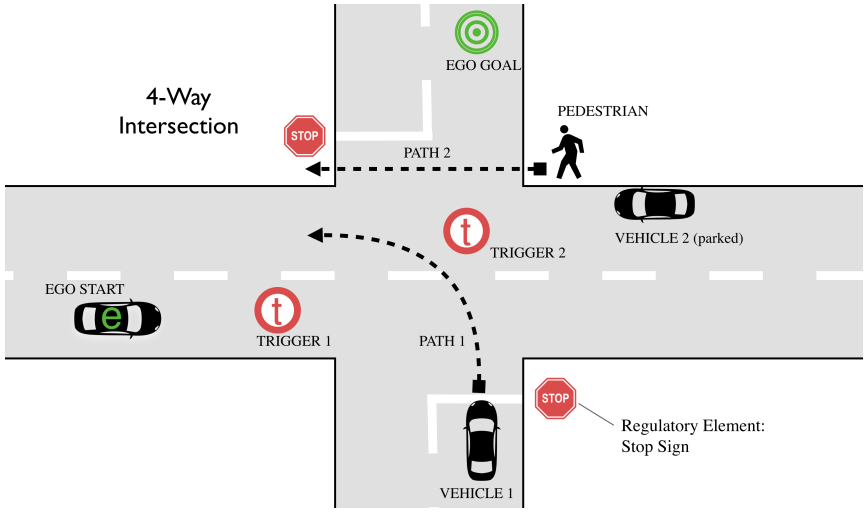


Figure 3.2: Overview of the main GeoScenario components in a 4-way intersection scenario.

3.3.1 GeoScenario Basics

All GeoScenario elements are based on two OSM primitive types: *nodes* and *ways*.

- **Node** is the core element of GeoScenario, representing a specific point on Earth's surface. Each node comprises an ID number and a pair of coordinates (latitude and longitude). Nodes are used to define standalone point features (e.g., a *vehicle* or a *pedestrian*), but also to compose the shape of other elements (e.g., a path).
- **Way** is an ordered list of nodes defining a polyline. Ways are used to define linear features such as *paths* and boundaries of areas (solid polygons that represent an obstacle on the road, or a named area for dynamic element placement). To define areas, the way's first and last node must be the same and are called *closed way*.

All elements (nodes and ways) can have tags describing attributes of an element with the pair of text fields k (key) and v (value). We use a tag gs to define an element's role in the scenario, that is, the element's function in the GeoScenario model (e.g., $gs = vehicle$). Elements without a gs tag do not have a specific role in the scenario, but can be used to compose other elements. For example, nodes composing a path do not have a gs tag. An element cannot have two tags with the same key.

All elements with a role must also contain the tag '*name*' (with a few exceptions). The name is a unique string that identifies one element in a scenario. This tag is used to derive

relations between elements. Nodes have coordinates in the WGS84 coordinate frame (as part of the OSM standard).

There is a fixed dictionary of tags documented in our GeoScenario specification, but we highlight the main properties per element in this thesis. Listing 3.1 gives an example showcasing a vehicle node and its basic components.

Listing 3.1: GeoScenario element example

```
<node id='1' lat='43.5094' lon='-80.5367'>
    <tag k='gs' v='vehicle'/>
    <tag k='name' v='leading_vehicle'/>
</node>
```

3.3.2 Ego and the Driving Mission

In a scenario, Ego is the entity representing the ADS-operated vehicle. In our language we decide not to define actions or maneuvers for the Ego. Instead, GeoScenario only specifies initial conditions and goals. During a test case execution time, the ADS is a black box system responsible for deciding the best route and maneuvers based on the traffic conditions (road network, static objects, dynamic agents on the path, etc.). We decide for this approach to allow the language to be system independent and to reflect a real world driving scenario. In practice, a driving mission is given to the driver or ADS as a global location to be reached as a long-term task. The initial condition is defined as a node representing Ego's starting position and orientation. We assume Ego always starts a scenario in a parked position.

Listing 3.2: Ego Goal Nodes

```
<node id='2' lat='43.5094' lon='-80.5367'>
    <tag k='gs' v='egogoal' />
    <tag k='order' v='1' />
</node>
<node id='3' lat='43.5095' lon='-80.5378'>
    <tag k='gs' v='egogoal' />
    <tag k='order' v='2' />
</node>
```

The goal is defined as an *egogoal* node. A scenario can have multiple ordered goal locations. They represent the intermediate and final driving missions the ADS should

achieve. The final goal for the driving mission task is the one with highest order number and must finish the scenario with a success state. The nodes can be used to compose a global path for the system, or create a goal point on the system's internal map. However, this is particular to the ADS configuration and is out of the scope of our model.

3.3.3 Scenery and Road Network

We use Lanelets [22] to represent the scenario road network (more in the Background Chapter, Section 2.4). We decide to use Lanelets because of their compact and lightweight structure; the GeoScenario format follows a similar spirit itself. The road network is stored in a separate XML file to make replacements easy. However, a scenario can only be interpreted within the context of the road network. Consequently, a GeoScenario must always be distributed with its associated road network file.

To represent stationary obstacles that are not part of the road network, but block or limit the drivable surface, we introduce *static object*. Static objects can be defined as a single node, a way, or a closed-way. A closed-way can assume arbitrary shapes, but in order to be valid, it must have the first and last node reference pointing to the same node ID. A reference to a model can be used to give the object a more defined form, and additional attributes such as width, length, height can also be used. We chose to keep the GeoScenario simple and flexible, and the model can be defined elsewhere.

A *traffic light* physical object is part of the scenery and is defined in the road network layer (lanelet). However, the state is dynamic and must be defined in the GeoScenario file. For example, the standard "green, yellow,red" states. Note that different countries can have different states and standard sequences. The time per state is given in (s) as a list with the *duration* attribute, or optionally as intervals in simulation time. Traffic light states can also be changed by trigger (introduced later in Section 3.3.5). Additional light types (e.g., turn, pedestrian walk, etc) can be added either as an independent traffic light element, or as part of the same interconnected traffic light state as sub-lights.

Listing 3.3: Traffic Light

```
<node id='1' lat='43.5094' lon='-80.5367'>
    <tag k='gs' v='trafficlight' />
    <tag k='name' v='intersection1_northlight' />
    <tag k='states' v='green,yellow,red' />
    <tag k='duration' v='15.0,3.0,20.0' />
</node>
```

3.3.4 Dynamic Elements

We define as *dynamic elements* all GeoScenario elements that are able to move (having kinetic energy) or are able to change their state. This is different from Geyer's [34] definition of dynamic elements, which are based on the temporal extent of the scene. In GeoScenario, a parked vehicle is also defined as a dynamic element. Dynamic elements that are able to move are called *agents*, and are separated in two types: vehicles and pedestrians. Both are represented as nodes and share similar attributes. Vehicle is defined with the tag $gs = vehicle$, and pedestrian with tag $gs = pedestrian$. The *orientation* tag is used do define an agent's initial orientation (for example, a vehicle yaw). In our model the orientation is given in degrees, with origin on East and clockwise direction. Different types of vehicles (e.g., car, truck, bus) are represented with the same type, with an optional attribute *model* specifying a vehicle model. We do not specify details of the vehicle model dynamics or 3D meshes. Therefore, testing results must take into account additional details of the simulation infrastructure running the scenario. A *speed* attribute (in km/h if not specified) is used to define a reference velocity.

In order to move, vehicles and pedestrians need to be assigned to a path. A *path* is defined as a Way element, and can be used for both vehicles and pedestrians. Paths should be interpreted as splines composed by ordered connected nodes. When a dynamic agent is assigned to a path, it will travel along the path with its reference speed.

To support more realistic kinematics with variable velocity and acceleration, or to reproduce scenarios from recorded traffic data, an agent can be assigned to a *speed profile*. When a path has a speed profile, it must contain nodes with the tag *agentspeed* to indicate the target speed for the agent once it reaches that node. The agent must always try to match the speed of the next node in its path with a constant acceleration. Alternatively, nodes can contain the tag *time* with the time in seconds to reach the node since the start of the path. With high density paths (i.e., more nodes) and a speed profile, a GeoScenario model can represent a diverse range of traffic situations, manually designed by experts, extracted from real traffic by sensors, or imported from naturalistic driving databases. Figure 3.3 shows three alternative stopping paths with varying density. More realistic motion requires more nodes, but at the cost of verbosity. As examples, Listing 3.4 shows a typical dynamic agent as a vehicle, and Listing 3.5 shows its path defined as a way. Note how the ID references to the nodes composing a path are given by the tag *nd* and must be interpreted as an ordered list.

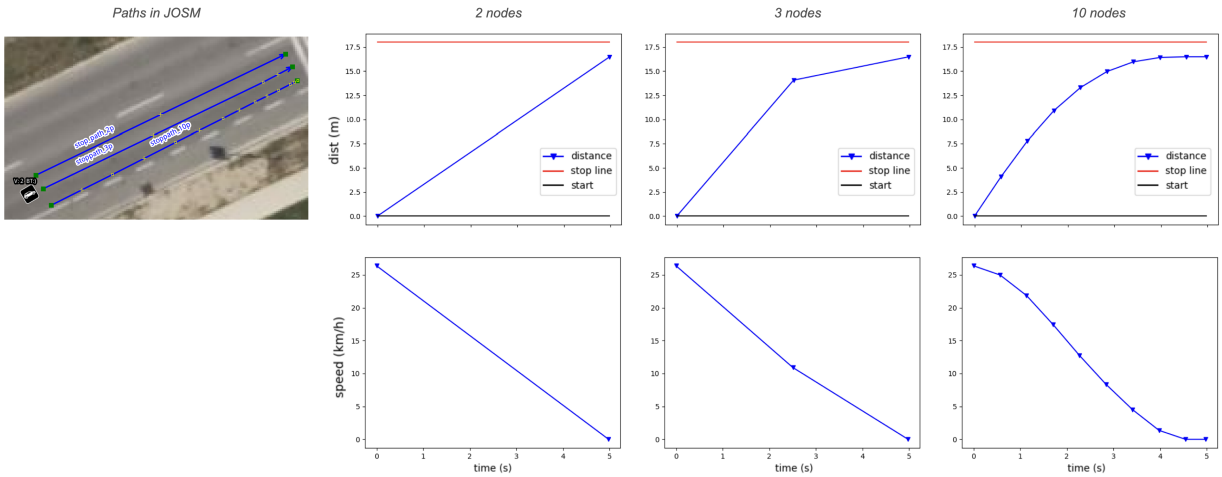


Figure 3.3: GeoScenario paths with alternative speed profiles for a stopping maneuver from 28 km/h to a complete stop in 5 seconds.

Listing 3.4: Dynamic agent

```
<node id='1' lat='43.5094' lon='-80.5367'>
    <tag k='gs' v='vehicle' />
    <tag k='name' v='leading_vehicle' />
    <tag k='speed' v='30' />
    <tag k='orientation' v='45' />
    <tag k='path' v='northpath' />
    <tag k='usespeedprofile' v='yes' />
</node>
```

By default, all paths are grounded to fixed node coordinates. We introduce the tag *abstract* to define flexible paths. Abstract paths are designed on fixed coordinates, but during execution must be shifted to a new origin point based on the agent's current location. Abstract paths can be used to design dynamic maneuvers. For example, a lane change that can occur at different locations of the road network.

Listing 3.5: Path

```
<way id='39'>
    <tag k='gs' v='path' />
    <tag k='name' v='vwest_path' />
    <tag k='abstract' v='no' />
    <nd ref='3' />
    (...)
    <nd ref='6' />
</way>
```

Some properties can be described by a fixed value or by value ranges. As an example, a dynamic agent's speed can be defined by a fixed value (e.g., 30 km/h) or by a range (e.g., from 20 to 40 km/h) using the notation [20:40], and [20,25,30,40] for a list of arbitrary discrete values. The variable attribute notation is used for scenarios that rely on sampling and test input mutation and allows our model to represent both logical and concrete levels of scenario. Mutation of test input values is commonly used is software testing, including driving automation systems [17]. Assigned values represent boundaries, and a concrete value is selected before a scenario is executed. Stochastic behavior with probability distributions is also supported. However, since many different probability distributions can be used, (e.g., Gaussian), they must be defined elsewhere. If no distribution is explicit, we assume a random value from Uniform Distribution. Listing 3.6 shows pedestrian with variable speed. For example, studies have found pedestrian walking speeds at crosswalks ranging from 5.32 km/h to 5.43 km/h for younger individuals [46] and we can use this value range as boundaries for our pedestrian speed.

Listing 3.6: Dynamic agent

```
<node id='-1' lat='43.5094' lon='-80.5367'>
    <tag k='gs' v='pedestrian'/>
    <tag k='name' v='crossingpedestrian'/>
    <tag k='model' v='adult'/>
    <tag k='speed' v='[5.32:5.43]'/>
    <tag k='path' v='crosswalk_west_path'/>
</node>
```

3.3.5 Triggers & Actions

In GeoScenario we introduce *triggers & actions* to orchestrate how a scenario evolves. The basic concept is to add trigger nodes in strategic places of the road network, and activate different actions over dynamic elements. Each triggers has *owners* and *targets*. Owners activate triggers, whereas targets execute the action (Figure 3.4). Owners can be the Ego itself or agents (vehicles, pedestrians). Targets can be any dynamic element whose state can change over the scenario, but can not be Ego. This rule follows our assumption of the ADS as a black box system, limited to the initial conditions and the driving mission. Actions can change an agent's state, or the scenario itself. Listing 3.7 shows a trigger example.

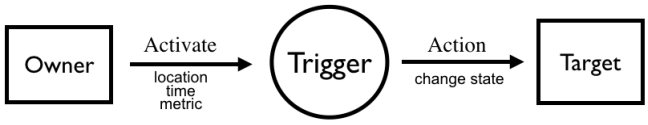


Figure 3.4: GeoScenario Trigger. When the owner activates a trigger, an action is executed on the target. The trigger can be activated when the owner reaches the trigger node location, when the Scenario reaches a certain time t, or when a metric between two agents reaches a certain value x.

A trigger can be activated by three types of conditions, or by a combination of them:

- *(i) Time*: activated when the scenario execution reaches a given time t. A set of timed triggers allow the designer to control the scenario in chronological order with timed events. For example, at a given time $t = 10$, a pedestrian starts crossing an intersection.
- *(ii) Location*: activated by overlap, when the owner reaches the trigger node location. Must be placed over strategic points of the Road Network. They are especially useful when timed events can not guarantee Ego and other agents are at the right place at the right moment. For example, one can place a trigger with $Owner = Ego$, and and action for a pedestrian to start a path over a crosswalk. This trigger guarantees the walking happens at the desired distance between Ego and pedestrian.
- *(iii) Metric condition*: activated when a given condition based on a metric is true. This trigger allows situations where an Action needs to be performed with no specific

location, but at any location after a relative condition. For example, a vehicle moving over a path on the road starts to decelerate to stop only when the distance between Ego and the vehicle is less than 100 meters. To support a condition, a GeoScenario needs to track a given metric between agents.

A metric is also defined as an element in GeoScenario, by explicitly declaring which agents are tracked. We encourage scenarios to include references for how a metric is calculated since different approaches can be used. For example, TTC can be computed using a variety of methods leading to different values [76, 67, 61].

Listing 3.7: Trigger

```
<node id='4' lat='43.50909' lon='-80.53654'>
    <tag k='gs' v='trigger' />
    <tag k='activate' v='location' />
    <tag k='name' v='start_trigger' />
    <tag k='owner' v='Ego' />
    <tag k='target' v='leading_vehicle' />
    <tag k='apath' v='west_path' />
    <tag k='aspeedprofile' v='yes' />
</node>
```

This thesis only provides an overview of the model. All details (with examples) can be found in our project's repository.[1] Additionally, the model can be easily extended with new features and attributes to support tool-specific requirements.

3.3.6 Tool Set

Accessible tools are important to make the model useful for researchers and engineers and adopted by the community. Since our format was developed on top of OSM primitives, we adapt its standard map editing tool: JOSM [3]. This is a free and open-source Java Application commonly used to create and share maps with the OSM Server. By adding a set of custom presets and style sheets, we can now easily design and understand a GeoScenario on top of the Road Network (Lanelet layer) and other map layers (e.g., Bing Maps, ESRI maps) before its execution. Figure 3.5 shows a sample scenario designed in our custom tool. The second tool is the GeoScenario Checker: a set of scripts to evaluate a Scenario's conformity with the language. Both tools are available at the Project's website along with their usage instructions.

[1] https://git.uwaterloo.ca/wise-lab/geoscenario

The tools are integrated with GeoScenario Server, a complete scenario simulation toolset. The server is introduced in Chapter 5 and consolidates the implementation of all models discussed in this thesis.

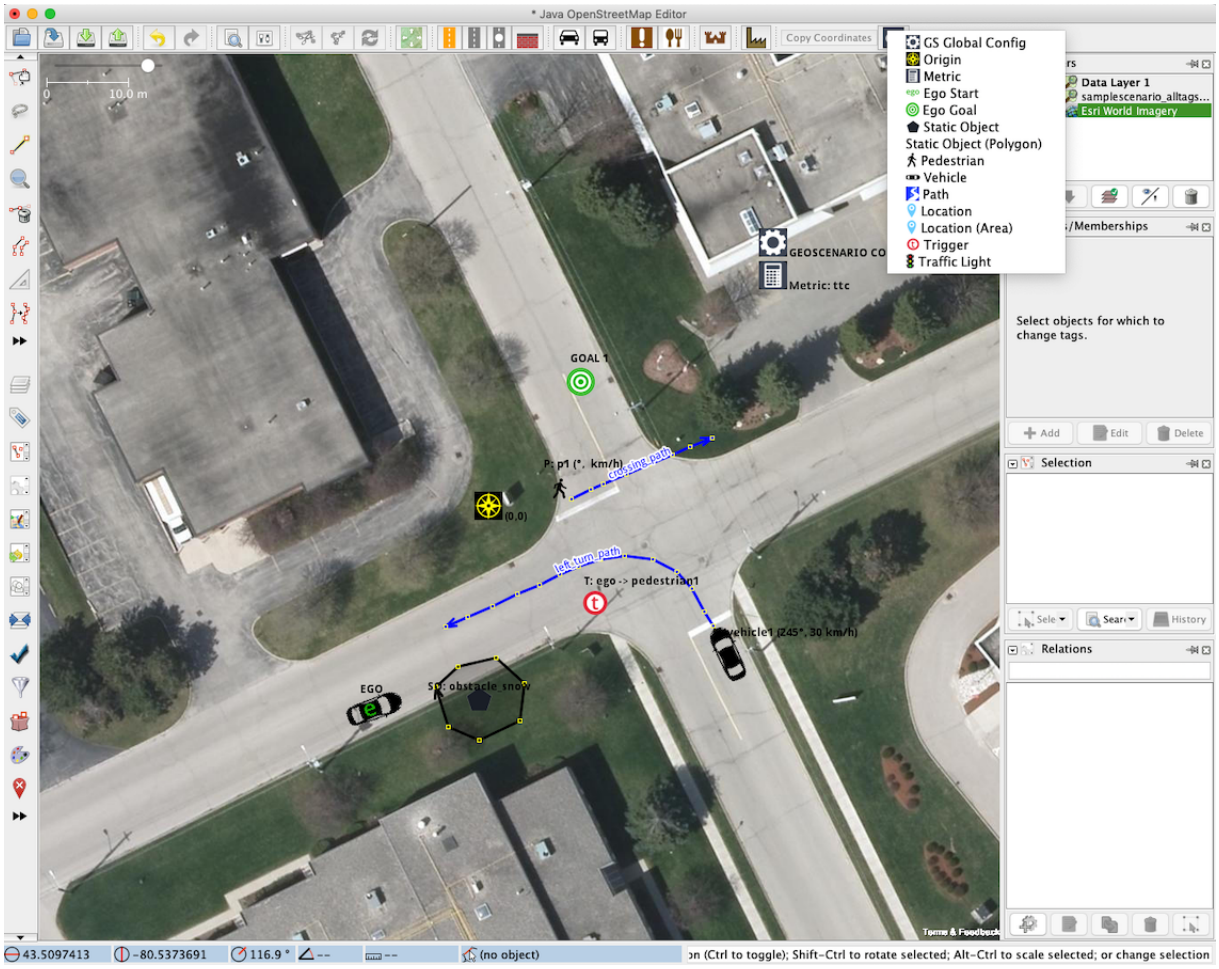


Figure 3.5: JOSM adapted to GeoScenario

3.4 Application

We incorporate GeoScenario to the WISE ADS Project testing infrastructure as the format to design and run scenarios for testing in simulation. In this section we describe our project and present a sample scenario, tested in simulation with the software stack.

3.4.1 The Research Platform

"UW Moose" is the University of Waterloo self-driving research platform. The platform is a Lincoln MKZ Hybrid modified to autonomous drive-by-wire operation and a suite of lidar, cameras, inertial and vision sensors (see Figure 3.6). The car is equipped with computers to run a complete autonomous driving system, integrating mapping, sensor fusion, motion planning, and motion control software in a custom autonomy software stack fully developed at Waterloo as part of the research. The system was the first Canadian-built ADS to be tested on public roads in Canada in August 2018. More info is available on the project website [15].

The autonomy stack is implemented on top of Robot Operating System (ROS) framework [9]. ROS offers an inter-process communication interface based on publish/subscribe anonymous message passing. We explore this interface to isolate the components we want to test and use data from our simulation tools to create a realistic testing environment. We focus our test on motion planning modules to explore Ego's interactions with other traffic agents. Because the publish/subscribe system is anonymous, we can isolate Motion Planning modules by simulating data from sensors (e.g., GPS, IMU), and Perception modules (vehicle and pedestrian detection) through ROS topic messages. We assume all sensors work without failure. This means our simulation environment is able to provide accurate detection of all vehicles within the range of the sensors in a map representing the Road Network.

Figure 3.6: "UW Moose" research platform. Lincoln MKZ Hybrid modified to autonomous drive-by-wire operation and a suite of lidar, cameras, inertial and vision sensors.

3.4.2 The Simulation Infrastructure

The Simulation toolset (WISE Sim) was developed on top of Unreal Engine 4.19 [2]. To support GeoScenario we create the *Scenario Manager* module, which is able to parse and execute GeoScenario and Lanelet formats using Unreal native features (e.g., environment creation, 3D object handling and collision checking), and to simulate detection. Vehicle motion dynamics for Ego simulation is handled by an external module *MKZ Vehicle Model*, adapted from [77]. Figure 5.4 shows an overview of the simulation architecture.

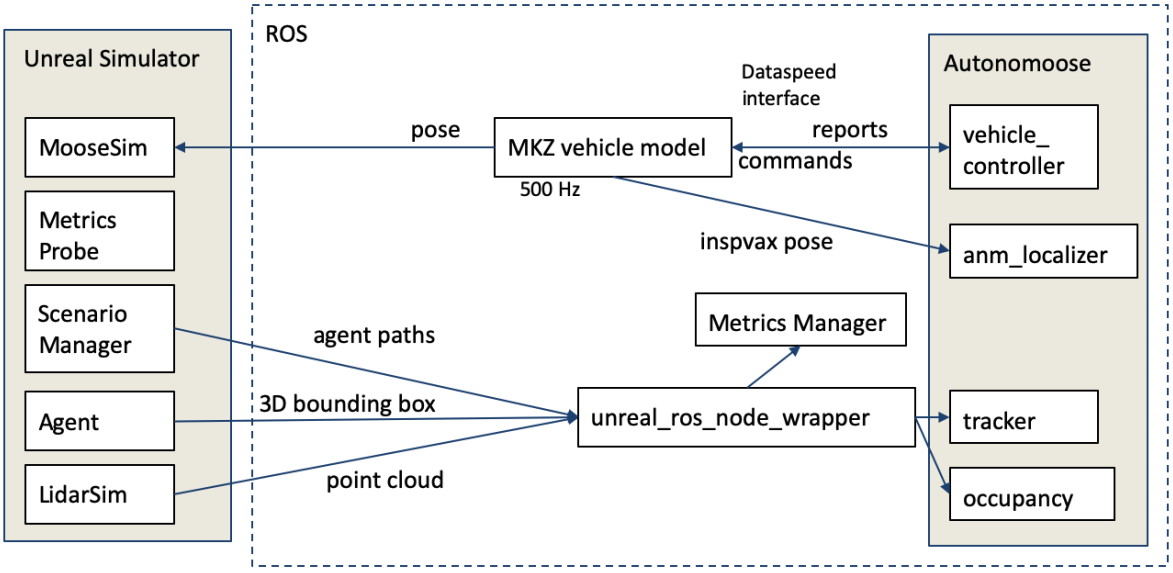


Figure 3.7: WISE Sim architecture. Scenario Manager is responsible for parsing and running GeoScenario and Lanelet files.

3.4.3 Designing a Scenario

We model the most frequent crash scenario according to NHTSA (lead vehicle stopped) as illustrated in Figure 1.1. We model this car-following scenario with a single Ego goal at the end of the road to define a driving mission, a dynamic agent as the leading vehicle shortly after the Ego start position following the road with constant speed over the *east_path*. When the leading vehicle reaches a trigger, it switches to *decelerate_path*. This path

[2] https://www.unrealengine.com

contains a speed profile, decelerating from 50 km/h to a complete stop in a short space. Ego is the trailing vehicle and must react to avoid a collision. If a collision happens, the ADS failed and the scenario must end. Figure 3.8 shows how we model our scenario, and its execution in our simulation environment. Triggers with different conditions can be used to explore this scenario with different ranges, and paths with varying speed profiles can be used to explore different types of deceleration (e.g., an aggressive deceleration profile increasing the level of difficulty).

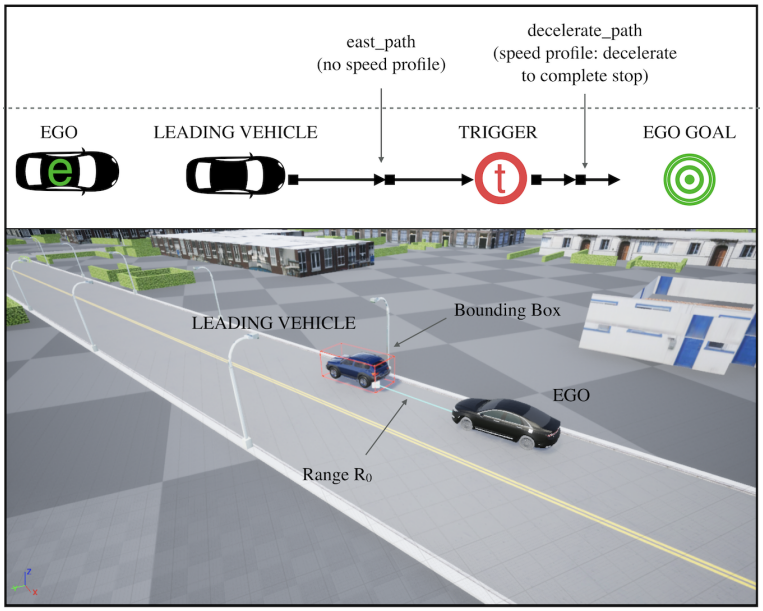


Figure 3.8: Rear end pre-crash scenario modeled in GeoScenario and executed in WISE Sim. Both vehicles are following the same lane. When reaching the trigger, the leading vehicle switches to a decelerating path profile until a complete stop. The blue line between vehicles indicates the *range* (distance between two vehicles). The red box is the bounding box used to simulate detection and bypass perception modules.

3.5 Limitations and Future Work

GeoScenario does not specify ADS actions beyond the initial conditions and goals composing a Diving Mission. This assumption was key in the language design to maintain system-independence. We model the interactions solely based on how other agents behave

during the scenario evolution. We do not include driver or vehicle behavior models and do not include advanced motion dynamics for vehicles. Instead, we model behavior and motion at the basic level using trajectories as paths with speed profiles and triggers to alternate behavior. Catalogs of trajectories can be used to mitigate this limitation, but scenarios that require dynamic interactions between multiple vehicles tend to be difficult to model and simulate (we address this limitation in the next chapter).

GeoScenario does not include elements to define environmental properties, such as precipitation, or fog. We also do not model the effects of weather and road conditions on friction, sensor performance, and vehicle dynamics. We recognize the importance of such elements in a scenario, but they vary greatly based on both the simulation capabilities on rendering the scene with varying levels of fidelity and the model for the subject system running in simulation.

We plan to improve GeoScenario to support testing with a focus on the perception task and include environment conditions that must be simulated to affect detection. New behavior models for dynamic agents can be added to support scenarios exploring agent behaviour that goes beyond path following for both vehicles and pedestrians. For example, simulating a distracted pedestrian talking over the phone to test the system's capabilities to predict and handle unsafe behavior. We address the vehicle behavior limitation in Chapter 4 with a model based on Behavior Trees and realistic motion. A model for realistic Pedestrian Behavior is presented in [50].

Another branch of improvement is to guide scenario generation. So far, parameters with value range and stochastic distributions are supported, but they do not help guiding the generation towards specific goals (for example, finding critical scenarios by performing successive input optimization). Metric composition with optimization goals can be modeled as part of the GeoScenario, whereas the optimization method can be defined outside of the model.

We built a database of testing scenarios with a wide coverage of requirements, exploring a wide range of system capabilities. Scenarios are both manually designed by researchers and extracted from traffic data. All scenarios are openly available, and the database can be extended with new scenarios from the community.

3.6 Chapter Conclusion

In this chapter we propose GeoScenario as a DSL for scenario representation. We identify key elements that compose typical test scenarios and which need to be formally declared

and executed on self-driving vehicle testing. By adopting GeoScenario in the simulation infrastructure of the WISE ADS Project to validate the autonomy stack under simulation, we demonstrate its applicability in practice. The language is built on top of well-known Open Street Maps primitives, and designed to be lightweight and easily extensible.

A toolset to easily design and validate scenarios using our DSL is publicly available. With the adoption of the format in WISE Lab projects, we publish a collection of tool-independent scenarios for self-driving vehicle testing. With an open format and open-source tools, we plan to expand the public scenario set into a shared database of tool-independent scenarios with contributions from the research community. The base language is still limited when scenarios require dynamic interactions between vehicles. We address this issue in the next chapter with the addition of a model for dynamic vehicle behavior and interactions targeting scenario design and a revision of the language as GeoScenario 2.0 integrated with the model. The full format specification is available in the evolving online documentation. [3]

[3] geoscenario.readthedocs.org

Chapter 4

A Driver-Vehicle Model for ADS Scenario-based Testing

In this chapter we propose the Simulated Driver-Vehicle Model (SDV) to represent and simulate vehicles as dynamic entities with their behavior being constrained by scenario design and goals set by testers. This model extends the base scenario-definition language GeoScenario introduced in Chapter 3. The layered architecture of the model leverages behavior trees to express high-level behaviors in terms of lower-level maneuvers, affording multiple driving styles and reuse. Further, optimization-based maneuver planner guides the simulated vehicles towards the desired behavior. We detail the model's architecture and how the components interact.

4.1 Introduction

Testing automated driving systems (ADS) requires simulating a wide range of operating scenarios to ensure their safety and conformity to traffic regulations and industry standards. As the responsibility for the driving task shifts from the human driver to the ADS with increasing levels of automation [62], the system is required to handle interactions with the other road users, in particular with human-operated vehicles (HVs). Scenarios for verification and validation must reflect how these dynamic interactions between humans and the subject system can unfold in real traffic.

Figure 6.5 shows an example based on the National Highway Traffic Safety Administration's (NHTSA) pre-crash scenario catalog [54]. In this scenario, the vehicle operated

by the subject ADS (aka Ego vehicle) moves in traffic when V_2 cuts in front of it, leading to a near-collision. This cut-in maneuver likely triggers a reaction by several other close-by vehicles, and the Ego's reaction strongly influences how the scenario unfolds. Testing the ADS capabilities in collision avoidance in such scenarios requires models that are able to represent and simulate the dynamics of the traffic, including the HVs and their interactions with Ego.

Many Domain-Specific Languages (DSLs) for scenario-based testing have emerged to support the scenario design and representation (see Chapter 2). As such, they include models for HVs and allow testers to define the HVs' behavior and guide their interactions with Ego when executed by simulation tools during testing. However, these languages are typically limited to relatively simple models, such as using an event-based orchestration mechanism that directly manipulates the simulated vehicle's primitive attributes [8, 5, 13]. In such a model, a condition may trigger a direct assignment to the vehicle's position and velocity. The focus of these languages is declaratively specify "what a vehicle must do" and "where it must be" in a particular scenario, but without detailing "how" it moves. Consequently, this approach relies on a simulation tool to implement the actual vehicle behavior, either by translating the high-level definition to the vehicle simulation model internal (and often implicit) to the tool or by forcing the change in vehicle state, while disregarding the limitations of a real vehicle in traffic. The potential mismatch between what is specified by the language and the actual vehicle behavior compromises test reproducibility across simulation environments and validity of the test results.

An alternative approach is to use catalogs of predefined trajectories (PDTs), either extracted from traffic or designed by testers using a variety of mathematical functions or polylines, and orchestrate them via triggers [59, 8, 19, 13]. This approach brings realistic trajectories to HVs (as described in Chapter 3). However, the traffic is a dynamic system with complex interactions amongst participants, as the example in Figure 6.5 illustrates. Further, ADS sub-systems often exhibit some non-determinism [47, 65], which leads to non-determinism in the overall ADS behavior. A predefined behavior at the level of trajectories is unlikely to adequately account for the wide range of possible reactions in a dynamic interaction between the ADS and the traffic and also achieve reproducibility. Further, such PDTs are typically specific and limited to certain road geometries.

To fill this gap between scenario design and execution, this thesis contributes a model to specify and simulate realistic HV behavior in ADS scenario testing, while affording high expressiveness, execution accuracy, scalability, and reuse. We refer to our model as the GeoScenario Simulated Driver-Vehicle model (or simply SDV model). It extends the base scenario-definition language GeoScenario [59] with HVs as dynamic agents in both scenario representation and simulation execution. The SDV model encapsulates driver and vehicle

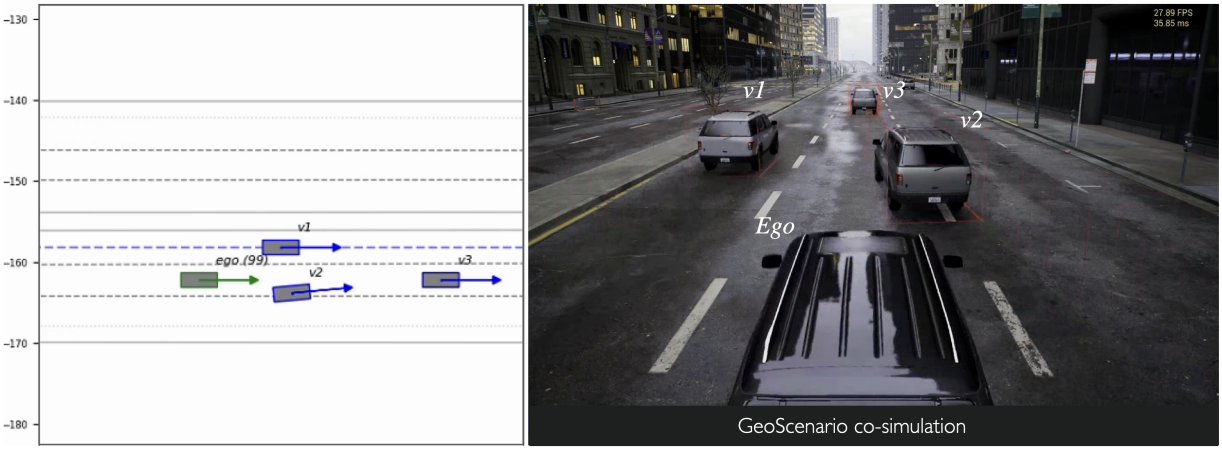


Figure 4.1: A challenging interaction between Ego and HVs based on a pre-crash scenario from NHTSA [54] and using the SDV model in simulation: (left) the SDV Model as V_2 performs the cut-in maneuver targeting Ego, and (right) a high-fidelity co-simulator renders the scene.

as a single entity and is based on two main layers: Behavior and Maneuvers. The Behavior layer is a higher level of abstraction aimed at coordinating the vehicle behavior using an explicit, user-oriented DSL. The Maneuver layer generates the actual vehicle trajectories and is designed to approximate how real vehicles drive on the road and optimize for the scenario test objective. Both layers are highly configurable to allow multiple driving styles, subject to the physical limitations and nonholonomic properties [44] of a typical road vehicle operating on structured roads.

A reference implementation of the model is available to the research community with a full scenario simulation tool that is ready to be integrated in co-simulation with any simulation environment (see Chapter 5). The toolset and additional model documentation is available in the companion website.[1]

[1] https://geoscenario2.readthedocs.io

4.2 Target Qualities

We identify target qualities a model must have to effectively represent and simulate HV behavior:

- *Reactiveness*: It must be able to react and adapt its behavior according to the road network and traffic rules, as expected from any traffic participant. For example, adapting to the road geometry and lane markings, reacting to regulatory elements and obstacles on the road, yielding or driving with right-of-way through intersections. Further, The model must dynamically respond to the unfolding simulation of the surrounding traffic, interacting with Ego and other participants.

- *Realism*: The model must be able to encode the nuance and diversity of the human driving. Further, it must reflect real vehicle motion (non-holonomic constraints). Some models in scenario simulation disregard such limitations, for example, using simplistic motion limited to a constant velocity throughout a maneuver, or abrupt changes from a moving state to a complete stop. The lack of expressiveness in the scenario design can also result in unrealistic motion when the representation (the DSL) do not encode the necessary detail to guide the simulation.

- *Effectiveness in scenario design*: Realistic simulation alone is not enough as the primary goal of the model is to support scenario development. The model must be able to express a diverse set of behaviors, allowing testers to cover a wide range of operating scenarios (*expressiveness*). Testers also need the flexibility to modify and reuse behavior across scenarios, accelerating the scenario design (*behavior reuse*). Finally, the model must be able to accurately simulate scenario and vehicle behavior with respect to scenario objective (*execution accuracy*).

- *Scalability*: The model must be able to scale with traffic density, for example, a scenario with multiple vehicles interacting with Ego in a busy freeway. It requires multiple instances of the model, and efficient use of computational resources in real-time simulations. Efficiency is also important to accelerate faster-than-real-time simulation in batch test and increase the number of scenarios with a fixed time budget.

We also identify a design constraint. Typically, macro-traffic models rely on a multi-agent orchestration entities to control the traffic flow. For example, a "*junction manager*" controlling a non-signalized intersection, or vehicle-to-vehicle (v2v) communication to coordinate interactions. Both solutions are commonly used for mass AI simulation in the

game industry. However, Ego is a black-box system and cannot be controlled by such entities or engage with v2v coordination with HVs. Additionally, other vehicles in traffic could be running inaccessible models (mixed participants). Therefore, the model must not rely on external entities or communication with other vehicles to perform the driving task. Current solutions lack the necessary qualities to support scenario-based testing. This led us to build a new model.

4.3 SDV Model Design and Architecture

We now introduce the structure of the GeoScenario Simulated Driver-Vehicle (SDV) model and its components. For simplicity and scalability, the model combines driver and vehicle as a single entity, abstracting away driver inputs, such as steering angle, braking, and throttle. The resulting behavior (model output) is the vehicle movement: position, velocity, acceleration, and heading (yaw) at each point in time, referred to as *VehicleState*.

We design a layered architecture inspired by the seminal works of Michon [52] and Boer et al. [23], which propose a hierarchical structure of the driving task with strategic (e.g., route selection), tactical (maneuver selection), and control (maneuver execution) levels. According to Boer et al. [23], the driving task can be characterized as a goal-directed behavior, where the goal is typically composed of "*a set of higher-level needs whose interaction affects how drivers orchestrate the set of observable low-level driving tasks.*" Our model architecture targets the tactical and control levels, with a focus on the ease of use for testers to express the overall tactical behavior in the behavior layer, and the remaining two layers providing reusable maneuver planning and execution in (see Figure 4.2):

- The *Behavior Layer* structures the driver tactical behavior. It breaks down the complex driving task into smaller tasks and coordinates maneuvers via a user-oriented DSL that consists of BTs and elemental maneuvers.
- The *Maneuver Layer* is responsible for trajectory planning. It turns a maneuver decision from the BTs into a viable motion profile based on the road, the surrounding actors, and the maneuver parameters.
- The *Execution Layer* is responsible for trajectory execution in simulation. The result is the vehicle state as the output from the SDV model to the simulation.

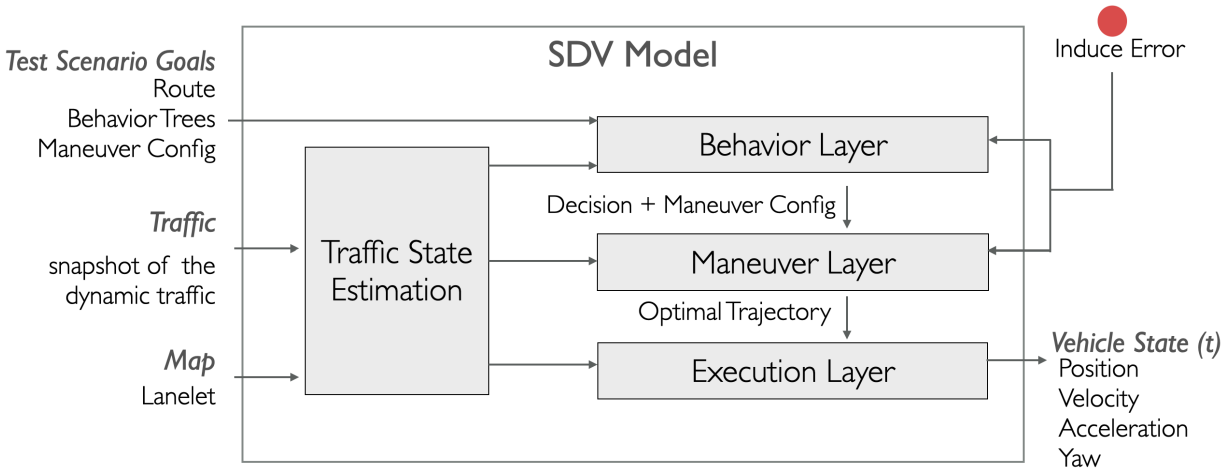


Figure 4.2: GeoScenario SDV model overview. The model combines driver and vehicle into a single entity and outputs the resulting vehicle state in simulation.

We rely on two general assumptions: Our model assumes scenarios are performed on structured urban and rural road driving, as well as right-hand traffic (although the models can be easily adapted to left-handed traffic). Next, we formalize our model representation and break-down the layers.

4.4 World Model and Vehicle Representation

We assume that a simulation holds a ground-truth bird's-eye-view representation of the world in two-dimensional Cartesian coordinates, including all the static elements of the scenario (road geometry and network, regulatory elements, static objects) and a simulation state for all dynamic elements (pedestrians, vehicles, and regulatory element state). Amongst the vehicles, Ego represents the vehicle under test, and its state is determined by an Ego vehicle model controlled by the ADS under test. The remaining vehicles can be SDV model instances, vehicles simulated by an unknown model, or vehicles controlled by a human during test. In the world representation, they are equal traffic participants with

a body and a physical presence. All dynamic actors are perceived by each other through their type and the state over time:

$$VehicleState_{\text{Cartesian}}(t) = [x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}, \theta]_t \tag{4.1}$$

The internal representation, systems, and the nature of their decision-making process responsible for the driving task are not accessible. This assumption is a core concept in our model to approximate our simulation model to real interactions in traffic, and maintain consistent behavior in simulation with mixed participants.

4.5 Vehicle Motion

The vehicle *driving mission* is defined by a *start state* and a *route* assigned in GeoScenario as part of the scenario design. The *scenario route* is a sequence of points to be visited (in order), and its last point is the *goal* (see Section 4.5.1 GeoScenario Route). From the Lanelet Map routing graph [57], we generate a sub-map of connected lanelets visiting each route point on a shortest path, if such a route exists in the road network. With all the lanelets in this route, a *global path* is formed by a sequence of points from the lane centre line. It is used to guide the vehicle motion and its progress along the route to the goal point. If the vehicle deviates from this route (after a scenario event), a new route is generated from the last state to the remaining route points.

The SDV parameterizes and plans its motion in its dynamic Frénet reference frame [79], rather than the global Cartesian coordinates of the simulation environment. This is motivated by the fact that safety requirements on the motion of an on-road vehicle are typically specified relative to its Frénet frame derived from the local lane geometry (e.g., see [70]). For a given global path segment surrounding the vehicle position (which aligns with the local lane centre line), we fit a spline, which we refer to as the *reference path* and use it for the frame transformation.

The reference frame (*Frénet frame*) is given by the tangential $\vec{t}$ and normal $\vec{n}$ vectors at the point along the arc length of this path that is closest to the vehicle. The resulting frame's S axis represents the longitudinal displacement along this path, and the D axis represents the lateral displacement (Figure 4.3). The vehicle motion is represented by a trajectory that combines two independent polynomial functions $S(t)$ and $D(t)$ in the Frénet frame, and T as the total time (4.2). Velocity and acceleration are the first and second derivatives, respectively, yielding the longitudinal and lateral state (4.3):

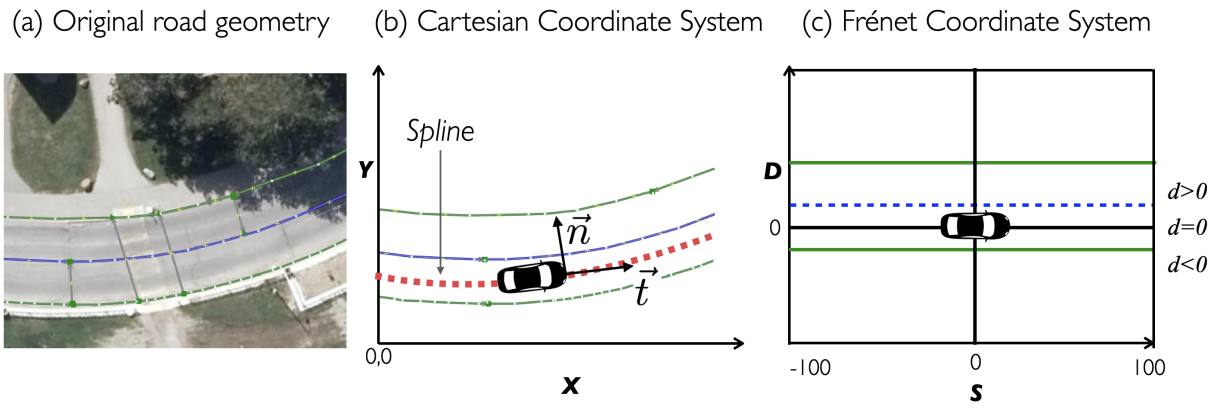


Figure 4.3: Road Geometry and vehicle displacement from original coordinates are transformed into Frenet frame using the tangential and normal vectors $\vec{t}$, $\vec{n}$ from the lane centre line (shown in red).

$$Trajectory = [S, D, T] \tag{4.2}$$

$$\begin{aligned} VehicleState_{\text{Frénet}}(t) = [&S(t-t_0), \dot{S}(t-t_0), \ddot{S}(t-t_0), \\ &D(t-t_0), \dot{D}(t-t_0), \ddot{D}(t-t_0)] \\ &\text{for } 0 \leq t - t_0 \leq T \end{aligned} \tag{4.3}$$

The SDV trajectory is planned by the Maneuver Layer (Section 4.8) in the Frénet frame, and the current SDV state is then translated to the Cartesian frame state (4.1) by the Execution Layer at each simulation cycle. This approach allows us to store the road geometry, surrounding traffic, and vehicle trajectories using a parametric representation that is road-geometry independent and computationally inexpensive to execute in simulation. Our method does not require a controller and driver inputs (e.g., throttle, brake) for a separate Vehicle Model, since we leverage the simulation nature of the model to output the vehicle pose from the Frénet-to-Cartesian transformations.

4.5.1 GeoScenario Route

A Route is defined in GeoScenario as a node, or composed by an ordered list of nodes (a way object, with minimum of 2 nodes) where the last node is the goal location. Each node

is a 2D point (lat, lon) intersecting a valid lanelet (a lanelet in which vehicles can drive). A Driving Mission is given to a vehicle when a route is assigned: "Drive using the shortest path from the vehicle start position to the last node n, while visiting each intermediate node in order along the Lanelet map".

Listing 4.1: GeoScenario route example

```
<!-- GeoScenario Route-->
<node id='20'  lat='49.0072755' lon='8.457139' />
<node id='21' lat='49.0081623' lon='8.4582812' />
<way id='10'>
    <nd ref='20' /> <!-- node 1-->
    <nd ref='21' /> <!-- node n (goal)-->
    <tag k='gs' v='route' />
    <tag k='name' v='my_north_route' />
</way>

<!-- GeoScenario Route with single node (goal)-->
<node id='30' lat='49.0072755' lon='8.4571394'>
    <tag k='gs' v='route' />
    <tag k='name' v='my_east_route' />
</node>

<!-- Vehicle-->
<node id='-1' lat='49.0072655' lon='8.45704'> <!--start position-->
   <tag k='gs' v='vehicle' />
   <tag k='btype' v='SDV' />
   <tag k="route" v="my_north_route" />  <!-- assigned route-->
</node>
```

A route node is considered as visited only if the vehicle reaches the longitudinal distance along the intersecting Lanelet or any neighbours. The neighbour Lanelets are evaluated for situations in which a vehicle is changing lanes along the route as part of the scenario behavior. *Cycle Routes* are defined using multiple points along the same Lanelets, respecting the order. A node is considered visited only if the previous node was visited.

4.6 Traffic State Estimation

The *Traffic State Estimation* is a support task transforming the state of the static and dynamic elements, including Ego, that surround the reference path into the SDV's reference frame (see Figure 4.2). Since both the SDV and the traffic are moving, the task predicts the state of the world for the next point in time when the Maneuver Layer will generate a new trajectory for the SDV. This predicted traffic snapshot in Frénet frame, along with the map, represents a simplified representation of the world, which is then used for decision making and trajectory generation. With the Frénet frame limited to the surroundings of the vehicle, a Cartesian representation is also stored for decision making with vehicles outside of the reference path (for instance, vehicle interactions in an intersection).

4.7 Behavior Layer

Given a *route* and the *estimated traffic state*, this layer performs the decision making, modelled using BTs. In each execution cycle, the SDV executes the main BT, which is a directed rooted tree with internal nodes being operators controlling the flow and leaf nodes being either (i) conditions to be evaluated (based on the traffic state), (ii) decisions that start (or end) maneuvers, or (iii) references to sub-trees.

Figure 4.4 shows a graphical representation of two sample trees, with the left one being the main tree, and the right one being a sub-tree referred to from the main one. The main tree first checks the sequence node, which tests whether the vehicle reached its goal; if this test succeeds, then the tree will issue a decision to stop (stop maneuver). Otherwise the vehicle continues driving by executing the sub-tree on the right, which first checks if there is a lead vehicle, in which case it issues the follow maneuver; otherwise it commands cruising at a set velocity.

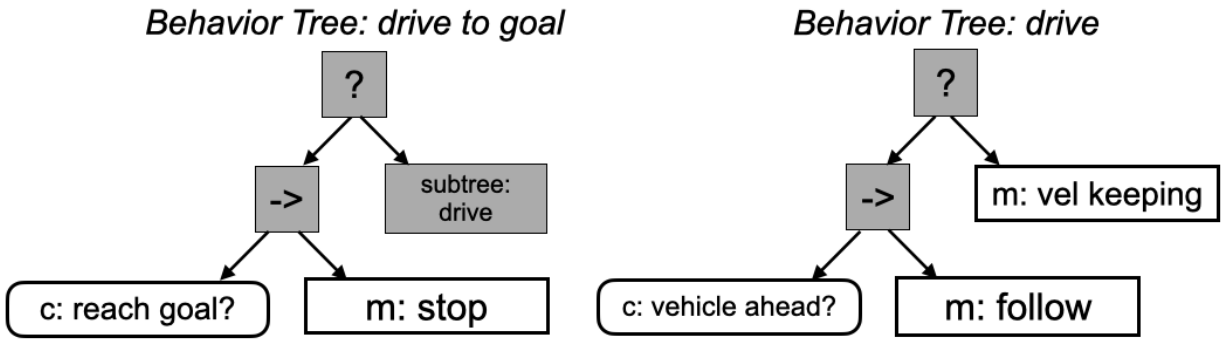


Figure 4.4: Graphical representation of a sample SDV BT structuring the decision-making with conditions (c) and maneuvers (m). '?' is the *fallback* operator (short-circuit or), and $\rightarrow$ is the *sequence* operator (short-circuit and).

The key motivation to use BTs is to provide test engineers with an easy-to-use means to specify scenario-specific SDV behavior. Rather than using a full-fledged behavior planner for an entire ODD (Operational Design Domain) to control an SDV, they can specify scenario-specific "micro-planners" by composing, parameterizing, and, if needed, customizing reusable BTs, expressed in a user-oriented DSL. This is possible since the driving task can be broken into smaller sub-tasks (e.g, road following, handling traffic lights, switching lanes), each encapsulated in a separate tree, and stored in a library. Test engineers can select the BTs representing the behaviors needed for a test scenario from the library and easily compose them through the sub-tree reference mechanism (as in Figure 4.4). They can also modify the driving style of an SDV by modifying BT parameters, and inject misbehaviors, such as dangerous cut-ins, by replacing normal maneuvers with BTs that represent such misbehaviors. Compared to using a full-fledged behavior planner, this approach shields the engineers from the decision logic needed to support other scenarios and thus eliminates unnecessary complexity.

4.7.1 Composing a Behavior

We follow the Behavior Tree classical formulation (see Section 2.7), with *control nodes* (sequence, fallback, parallel) coordinating the execution of their children nodes and *behavior nodes* adapted to domain-specific tasks that compose the vehicle behavior: *Condition* and *Maneuver*. Conditions evaluate the vehicle *self state* and the estimated *traffic state* and must be satisfied to return *SUCCESS*. Table 4.1 shows a sample list of condition nodes

available in the SDV model that can be used to compose a custom behavior (the full list is available in Appendix A). The *Maneuver node* represent the current decision in the form of a maneuver and its configuration. This decision can be replaced if a new maneuver node is reached on the same tick. Finally, we also add the *subtree node* as a composition node with another Behavior Tree starting from the root (one can also override nodes from the subtree to specialize the behavior). For each planning tick, the Behavior Tree is executed from the root node in a depth-first search until the root returns its state (entire tree is traversed). The last selected maneuver is the decision that propagates to the next layer. The syntax in text form uses the symbols `‘‘?"`, `‘‘->"`, and `‘‘||"` to denote the sequence, fallback, and parallel control nodes respectively. Appendix B shows the complete SDV Behavior Tree grammar in ANTLR4 format [1].

A full Behavior Tree after subtree composition is shown in Figure 4.5. Chapter 6 shows a practical BT and how it captures a range of behaviors via parameters (Figure 6.7). Further examples are available in the online documentation.

Table 4.1: Condition nodes

Condition	Description
reached_goal	Success if vehicle has reached or passed the goal point.
at_lane_change_segment	Success if vehicle is inside a road segment where a lane change is required to continue on route.
vehicle_stopped	Success if vehicle is not moving.
vehicle_yielding	Success if vehicle is stopped and at yielding position (stop line or right before a conflicting lanelet). *vel* and *distance* (from the stop line) are thresholds to define the yielding state. A small velocity threshold can be used to account for noise in the estimation.
vehicle_parked	Success if vehicle has stopped and at parking position. *vel* and *distance* (offset from the center of the lane) are thresholds to define the parked state. A small velocity threshold can be used to account for noise in the estimation.
can_lane_change	Success if a lane change to *target_lane* (specified in the node or set by a previous action) can be performed. *gap* and *time_gap* can be used to configure the acceptance conditions.
distance	Success if distance to given vehicle (in Cartesian) is between *min* and *max* (inclusive) in meters.
time_gap	Success if time distance to given vehicle (in Frénet, if vehicle is on the path) is between *min* and *max* (inclusive) in seconds.
gap	Success if distance to a given vehicle (in Frénet, if vehicle is on the path) is between *min* and *max* (inclusive) in meters. Note gap is negative if vehicle is behind.
lane_occupied	Success if current lane is occupied (i.e., there is a vehicle ahead). Limited to maximum *time* and *distance* as thresholds.
traffic_light_state	Success if state of the traffic light (applicable to current lanelet) matches the given color *state*. Example: traffic_light_state(color='RED').

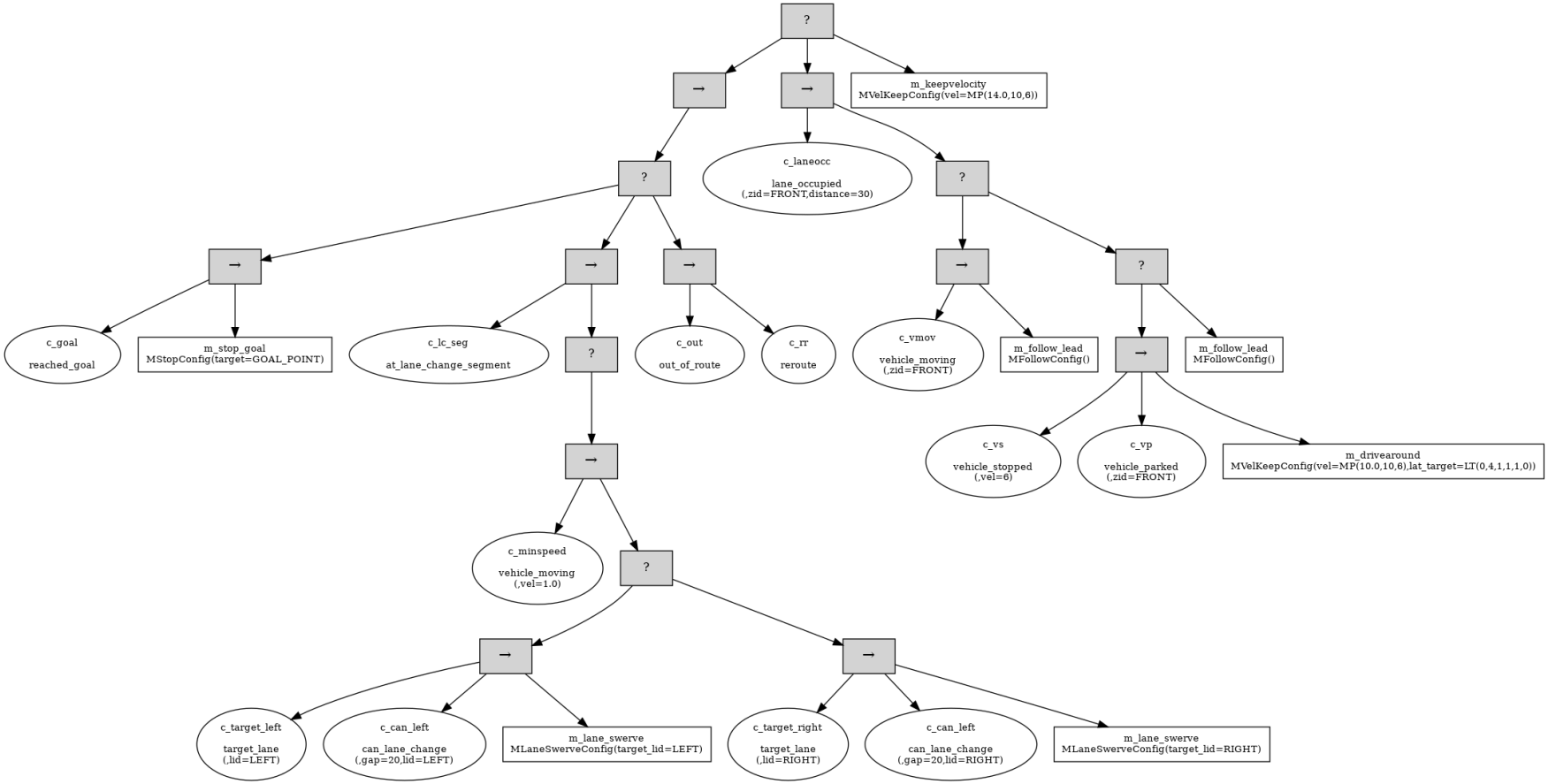


Figure 4.5: A sample SDV BT structuring the decision-making of a standard driver

4.8 Maneuver layer

This layer is responsible for the actual vehicle motion on the road. It receives a maneuver decision from the Behavior Layer and implements it by generating a feasible trajectory, which can be performed by a real vehicle. To achieve this, the model is bounded by a set of feasibility constraints respecting the vehicle dynamics. This is an important distinction from the behavior models assumed by the scenario definition languages and traffic simulators discussed in Chapter 2.

A maneuver is *"goal-oriented vehicle motion control behavior undertaken by a human driver or an ADS in order to achieve a specific result/outcome."* [62] A key element of the result is the target state of the vehicle, such as reaching a desired velocity or an adjacent lane and challenging Ego is a specific way. Furthermore, the maneuver must account for the road geometry, other traffic, including Ego, and the desired driving style, according to the test objectives.

Each maneuver is defined through a set of trajectory characteristics relative to the road environment. A maneuver exposes a set of parameters to control it according to scenario objectives. We use existing maneuver catalogs [62, 29] and implement a subset to support the evaluation in Chapter 6 velocity keeping, vehicle following, lane swerving (used for lane change and swerve-in-lane), merge-in-front, stop, and reverse. Note that these are elemental maneuvers, and composite maneuvers are implemented as BTs over the elemental maneuvers. For instance, lane maintenance composes velocity keeping, vehicle following, and stop. The maneuvers instantiate a general model (see Figure 4.6), which has three steps: (i) finding the target states for the maneuver, (ii) generating candidate trajectories, (iii) selecting an optimal trajectory. Each of these steps is controlled by a set of configurable parameters, allowing testers to realize a particular driving style or misbehavior. The generated trajectories are kept short (2 to 5 seconds), but some maneuvers, e.g., vehicle following, are performed over extended periods of time and, therefore, consist of a sequence of trajectories. The Behavior layer decides when to start, finish, or abort a maneuver.

Target Finding

Each maneuver has its own criterion to define a target state and a time to reach it. Target finding requires evaluating the road structure, traffic, and other objects. For example, the target for velocity keeping is to reach and keep a target velocity, while in the same lane; and the target for vehicle following is to reach and keep a certain target gap.

The maneuver configuration is used to adjust the desired behavior according to the

scenario goals by assigning target ranges to these parameters. Any lateral position relative to the current lane can be used, but if the maneuver is a lane swerve, the position is relative to the target lane. In the merge-in-front maneuver, the goal is to reach the same lane as the target vehicle, while achieving the target differences in position, velocity, and acceleration (δS). These target parameters allow simulating a dangerous cut-in maneuver by setting the gap to be small and closing. Our online documentation has a detailed description of maneuvers and target configuration options.

While defining the maneuver configuration, parameters can be set as a single value or a value range, e.g., a vehicle target speed of exactly 14 m/s, or within 20 % from 14 m/s. During execution, our model samples multiple values for each range parameter independently and creates a target state set as a Cartesian product over the parameter value sets. The sampling method of choice and the number of samples per parameter are configurable. The target state set is used to generate multiple trajectory candidates and select the best trajectory, filtering out configurations that may be infeasible or suboptimal.

Trajectory Generation

Given a target state set, trajectory generation computes a smooth motion profile between the current vehicle state and each target state in the Frénet frame. We use an approach that plans each trajectory as a pair of quintic polynomials, in longitudinal and lateral direction, respectively [79], which minimizes jerk to reflect smooth and comfortable driving. A quintic polynomial is a jerk-minimal connection between two points P_0 and P_T, in a one-dimensional problem with $p(t)$ as location and T as the motion duration [73]. The total accumulated jerk over the one-dimensional trajectory is given by 4.4:

$$J_{p,T} := \int_{t=0}^{t=T} \dddot{p}^2(t)dt \tag{4.4}$$

Trajectory generation creates a trajectory by computing the coefficients of two quintic polynomials, $S(t)$ for the longitudinal dimension as $p(t)$, and $D(t)$ for the lateral direction as $p(t)$, to fit the boundary conditions: the initial state $\mathit{VehicleState}_{\text{Frénet}}(t_0)$ and each of the target states $\mathit{VehicleState}_{\text{Frénet}}(t_0 + T)$ from the target-finding step. This results in a candidate set that respects the target constraints.

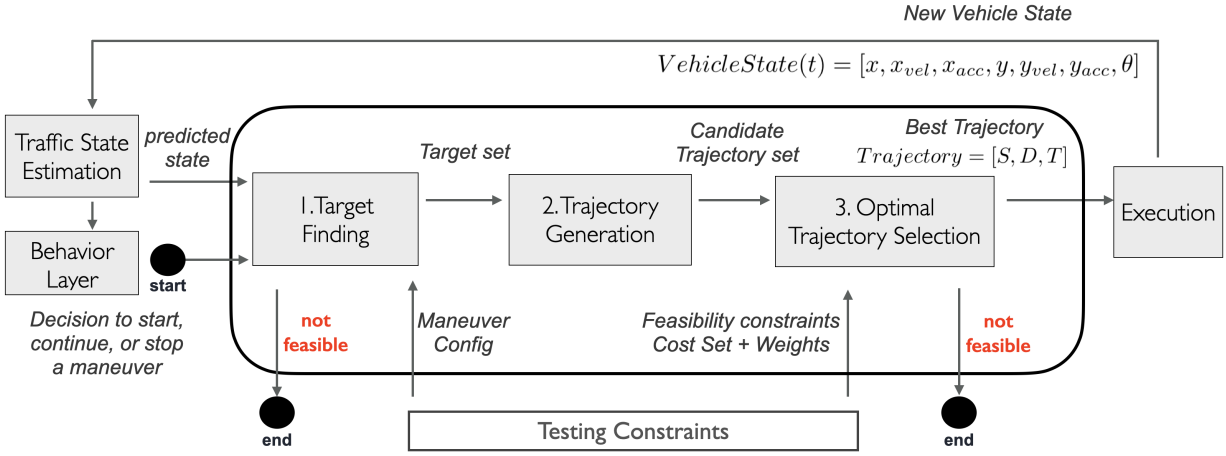


Figure 4.6: SDV general maneuver model

Optimal Trajectory Selection

This step selects a feasible and optimal trajectory from the candidate set, based on feasibility constraints and cost functions. *Feasibility constraints* reject trajectories with any collision, direction inversion, lane departure, and exceedance of maximum lateral/longitudinal jerk and acceleration. These are checked by sampling points over the S and D trajectories independently (see Figure 4.7) and using thresholds for accepted behavior (for instance, a more aggressive driving style can accept higher Jerk). For collision checking, we use the predicted movement from other dynamic agents (e.g., Ego), and static object locations as illustrated in Figure 4.8.

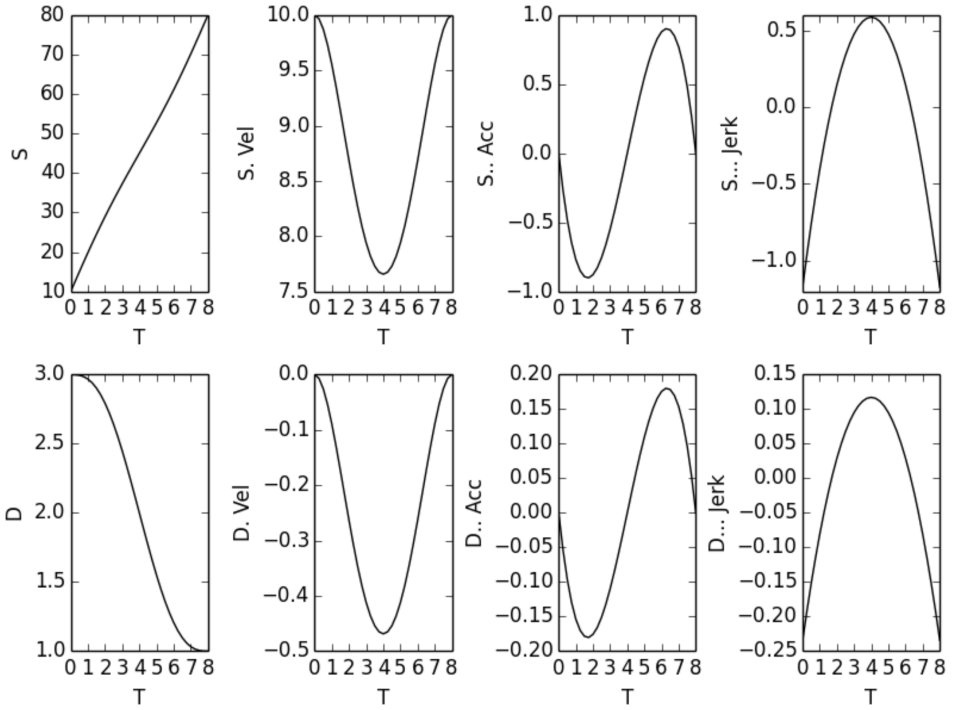


Figure 4.7: Sub trajectories in S and D for a swerve

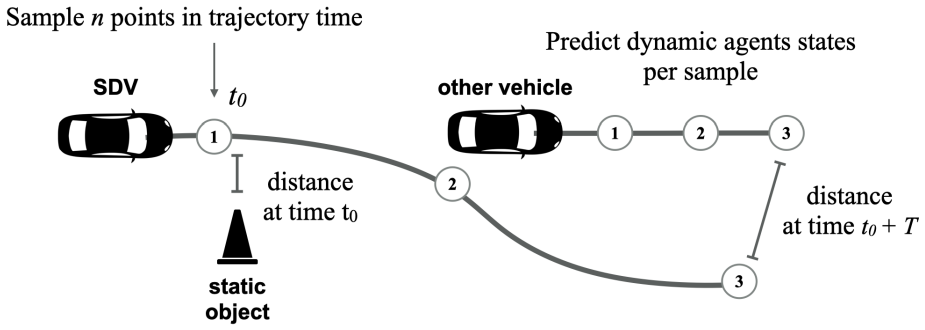


Figure 4.8: Checking for collisions with static objects and dynamic obstacles

The remaining candidate set is ranked using a weighted sum of *cost functions*:

- Time cost: Penalizes trajectories longer or shorter than the target time T.
- Efficiency cost: Penalizes low average velocity.
- Lane-offset cost: Penalizes distance from lane center during the entire trajectory.
- Jerk cost: Penalizes high longitudinal and lateral jerk over the entire trajectory ($J_{S,T}$ and $J_{D,T}$).
- Acceleration cost: Penalizes high longitudinal and lateral acceleration over the entire trajectory.
- Proximity cost: Penalizes proximity to obstacles (vehicles, pedestrians, or other objects).

The best trajectory is the lowest-cost feasible one. Weights can be adjusted per BT node according to scenario goals. For example, if a given scenario requires the vehicle to drive too close to Ego, the proximity cost weight for Ego must be lowered. The resulting trajectory respects realistic vehicle motion, balances conflicting qualities such as progress and comfort, while implementing the scenario goals.

4.9 Execution Layer

The selected trajectory is executed as a function of time. At each new planning cycle, the BTs either continue the trajectory or switch between maneuvers if a new condition is triggered. Figure 4.9 shows an example of trajectory planning for a cut-in maneuver to the right lane. The grey lines are the candidate trajectories eliminated by feasibility constraints or higher cost. The blue line is the best cut-in trajectory based on scenario goals (target and weight values) and motion constraints. The green line is the target vehicle (Ego) trajectory.

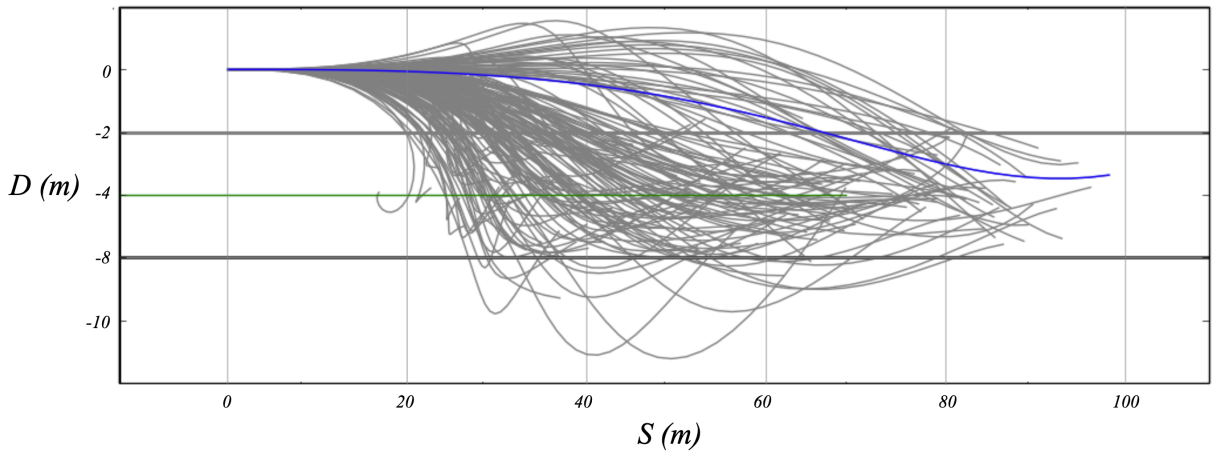


Figure 4.9: Trajectory planning by the SDV during a cut-in maneuver

4.10 Chapter Conclusion

We propose the SDV model to express and fully simulate realistic HV behavior in ADS scenario-based testing. The model encapsulates driver and vehicle as a single entity with a layered architecture that provides a user-oriented language to coordinate the vehicle behavior with a higher level of abstraction, and it also includes a planner for vehicle motion that optimizes for realism and the scenario test objective. The model fills the gap between scenario design with traditional DSLs for scenario representation and the execution in simulation with dynamic interactions between HVs and Ego, by allowing for a high-level, declarative description, but also improved controllability and consistency. We detail implementation decisions and performance considerations when using the model in the next chapter, followed by the Evaluation in Chapter 6.

Chapter 5

Reference Implementation, Performance, and Integration

In this chapter we discuss the GeoScenario and SDV Model reference implementations, performance and scalability considerations to achieve the target qualities of the design, how the implementation is integrated with simulators and an autonomy stack for testing, and examples of the model running in practice. This implementation and integration are the basis for experiments in the evaluation (Chapter 6). Additional details and a catalog of scenarios are available in the evolving documentation, [1] and the code repository. [2]

5.1 GeoScenario Server

The GeoScenario Server project contains the GeoScenario parser, the SDV model reference implementation, and other tools for running full scenarios in simulation. The server parses scenario definitions expressed using Lanelet2 map [57] and GeoScenario language [59] extended with the SDV BT definition format and creates a traffic simulation with the SDV model instances running concurrently. Vehicles that do not require complex reactive behaviors can use predefined trajectories rather than the SDV model, which improves performance. The server is implemented in Python 3.8, targets Ubuntu 20, and operates as a co-simulator to be interfaced with the simulation of the Ego vehicle, its sensors, and the ADS under test. Figure 5.1 shows the GeoScenario Server architecture. The components are described as follows.

[1] https://geoscenario2.readthedocs.io/en/latest/

[2] https://github.com/rodrigoqueiroz/geoscenarioserver

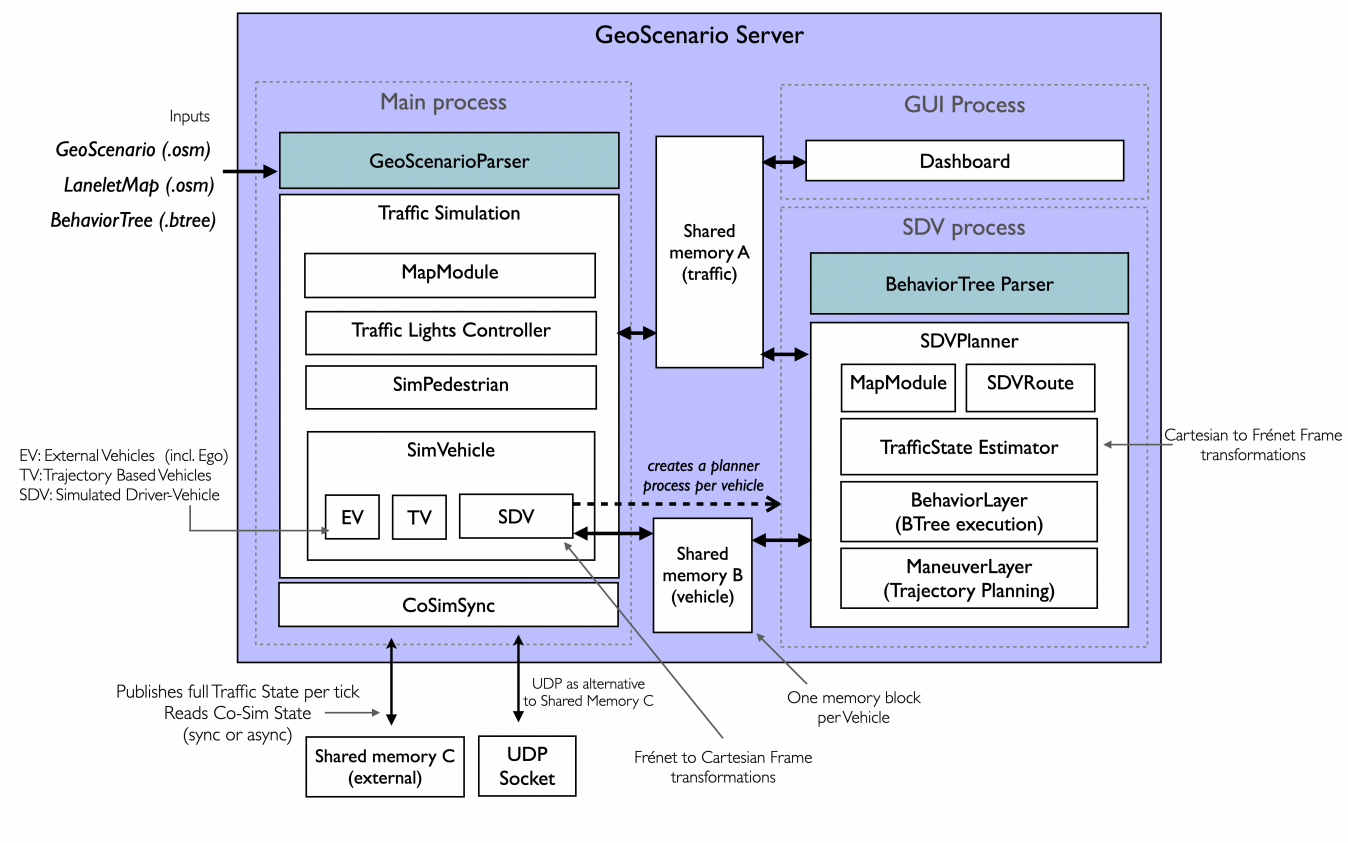


Figure 5.1: GeoScenario Server (GSServer) and the SDV Model reference implementation architectures

- *GeoScenario Parser*: The starting point of a scenario simulation. It reads and parses scenario definitions from GeoScenario files (.osm) using *libxml*, and the Lanelet2 map (.osm) using the *liblanelet* [57].

- *Traffic Simulation*: This component coordinates the full traffic simulation (vehicles, pedestrians, Ego and co-simulation interfaces) and monitors collisions. Additionally, it adds *StaticObjects* and *TrafficLights* (a light state controller) over the static Lanelet Map based on the scenario definition.

- *MapModule*: Interface for map related queries (identifying road and lane boundaries, network connectivity, and regulatory elements). This module is used by both the *SimTraffic* and the *SDVPlanner* module in the planning task.

- *SimPedestrian*: Pedestrian simulation based on GeoScenario (v1.0, as defined in Section 3.3) or the Social Force model as defined in [50].
- *SimVehicle*: vehicle simulation with three types:
  - *External Vehicle* (EV): Vehicles simulated outside of the server. These include Ego, as the vehicle is operated by the ADS under test and simulated with a separate model. Other vehicles not simulated by the GeoScenario Server are also represented with this type. For example, vehicles operated directly by a tester (manual drive) in real-time simulation, or operated by an alternative behavior model running in the Co-Simulator. The role of this vehicle is to synchronize the $VehicleState$ with the Co-Simulator or the Autonomy Stack directly.
  - *Trajectory-based Vehicle* (TV): vehicles that do not require intelligent behavior can use predefined trajectories rather than the SDV model. They are ideal for better performance with high traffic density, but limit the scenarios in which they can operate. They follow the Dynamic Agent model from the base GeoScenario, as defined in Chapter 3;
  - *Simulated-Driver Vehicle* (SDV): Vehicles running the SDV Model as defined in Chapter 4. They run in the *Main process*, while the planning task runs in a parallel process (detailed in Section 5.1.1).
- *CoSimSync* (The Co-Simulation Synchronization): This component is responsible for synchronizing the state of the simulation between the server and a client simulator. We use two alternative methods: Shared Memory (*shared memory B*) and UDP (User Datagram Protocol) Sockets. Shared memory is the preferred method for better performance, but requires both simulators and/or the Autonomy stack running in the same computer. The UDP Socket can synchronize the simulation from different computers over the network. The simulation can be asynchronous (default) or synchronous. On *asynchronous mode*, the server runs the simulation independently and updates the entire simulation state at the end of each tick (writing in the shared memory or sending a datagram), while reading the client state and updating the internal representation for Ego and all external vehicles in the next tick. On synchronous mode, the server waits for the client tick after writing the simulation state before it executes the next tick.
- *Dashboard*: This visualization module plots the map, vehicles, pedestrians, and trajectories in both Cartesian frame and Frénet frame. It is an optional module for

debug and demonstration purposes, running in a separate process. It is computationally expensive and runs at a lower rate (10Hz recommended). The update rate of the GUI does not affect the rate of the simulation.

5.1.1 SDV implementation

The SDV model runs in two separate processes: Planning and Execution. The separation of roles and hierarchical structure of the model design promotes this separation and allows parallelism. Execution is where the progress in the trajectory is computed and transformations from Frénet to Cartesian frame are performed. It requires a higher rate and must be compatible with the simulation rate of the entire traffic. Planning is a computationally expensive and time-critical task and needs to be executed with a fixed time since it is based on a future state prediction (self-state and traffic state). It includes the *Traffic State Estimation*, the Behavior Tree execution in the *Behavior Layer*, and the trajectory generation and optimization from the *Maneuver Layer*. For each vehicle running the SDV instance, the server creates a new process for the *SDV Planner* and an exclusive shared memory block (*shared memory B*) to exchange the updated *VehicleState* and Trajectories. This parallelism paired with a fast shared-memory interface between the two processes is fundamental to keep the performance of the model under target in real-time simulation. Further, non real-time simulation can be used to accelerate and scale the number of scenarios in testing with a strict time-budget.

- *BehaviorTree Parser*: This component parses Behavior Tree files (.btree) and creates an internal representation of the tree using the *py_tree* library. The root tree is assigned to a vehicle in the GeoScenario file, and subsequent sub-trees are parsed and composed in internal tree structure as a single Behavior Tree instance. The parser is based on *ANTLR 4.7.2* (ANother Tool for Language Recognition) [1].

- *TrafficState Estimator*: This component is responsible for transforming the state of the traffic simulation (static and dynamic elements, including Ego, that surround the reference path) into the SDV's reference frame. The traffic state is available in *shared memory A*. It predicts the state of the world after the planning cycle is complete to account for the moving traffic and self state along the previous trajectory. This predicted traffic snapshot in both Frénet frame and Cartesian frame represents the world and is used by the *BehaviorLayer* and *ManeuverLayer* for decision making and trajectory generation, respectively.

- *SDVRoute*: This component is responsible for creating a route from a GeoScenario route definition and tracking the vehicle progress along the route. This module can identify the goal, and other relevant road segments. For example, where a lane change is required to continue on route, or when the vehicle deviates from original route and a re-planning is required to generate a new *Reference Path*. Note that actions related to the route depend on the vehicle behavior defined in the Behavior Tree, as the behavior is constrained by the scenario design.

- *BehaviorLayer*: This component executes the Behavior Tree assigned for the vehicle. Each planning cycle restarts the execution from the root, and traverses the tree until a decision is made (an action or maneuver).

- *ManeuverLayer*: This component implements the trajectory generation and optimization task based on the Behavior Tree decision and maneuver configuration as inputs.

5.1.2 Balancing performance

Two major parameters impact the overall performance of the model: Traffic Rate and Planner Rate. They are particularly important when real-time simulation is required. For the *Traffic Rate* we recommend 20 Hz to 30 Hz. Note they will depend on the perception and tracking rates of the autonomy stack. A higher rate will generate smoother vehicle motion, but the additional frames might not be used in practice. For the *Planner Rate* we recommend 3 Hz to 5 Hz.

Additionally, planning parameters can affect the performance and must be used with caution when running simulations with multiple vehicles.

- *Cost sampling*: When optimizing the trajectory selection, two parameters define how many samples are extracted from each trajectory to compute feasibility and their cost:

  - *Samples per Trajectory* (SPT): The minimum number of samples extracted from each trajectory to compute the cost. We recommend a value between 10 and 20 SPT. However, scenarios that require high precision in reaching certain parameters might require more samples.
  - *Samples per Second* (SPS): The minimum number of samples per second extracted from each trajectory to compute the cost. This will allow trajectories configured to span a longer time horizon to have more samples than defined by

the SPT parameter. We recommend using 3 SPS. For example, a trajectory of four seconds will require 12 samples as a baseline.

The resulting number of samples is the highest between SPT and SPS. Cost functions based on integrals will use this parameter for approximation. These parameters apply to all trajectories in a vehicle instance, but they can be modified per maneuver node in the Behavior Tree.

- *Path Points per Meter*: The number of points-per-meter to be used along the vehicle's reference path as we generate the Spline. Higher density generates smoother curves. We recommend 3 points-per-meter.
- *Reference path length ahead*: Look-ahead distance of the Frenét Frame (max longitudinal distance). We recommend 100 m. Note the lookahead distance for conditions can not be higher than this parameter.
- *Reference path length behind*: Distance behind the vehicle to be included in the Frenét frame. We recommend 50 m (or higher for high speed merging scenarios).

External factors can also impact performance. For example, the total number of vehicles in traffic (regardless of running SDV models), pedestrians, and static objects will affect transformations in both Traffic Rate and Planner Rate, and impact collision checking during the trajectory optimization as they get closer to the SDV. We recommend scenarios with up to 10 vehicles along the reference path. The Reference path length ahead and behind the vehicle can be adjusted to reduce the number of transformations and fine-tune performance for a particular scenario or available computational resources.

5.2 SDV Model Examples

We show examples of the SDV Model running with the reference implementation. Note the examples are not complete test scenarios (there is no ADS under test, which is replaced by an SDV). However, they showcase some situations that vehicles running the SDV model can handle in a scenario.

Figure 5.2 (a) shows a typical *follow-and-stop scenario*. In $t(1)$, the lead vehicle ($v2$) starts moving and the following vehicle ($v1$) accelerates. In $t(2)$, both vehicles are driving at full speed (around $14\,m/s$) and $v1$ keeps a longer distance while projecting a trajectory ahead of the current $v2$ position to maintain the 2 s TTC with the predicted $v2$ motion.

In $t(3)$, $v1$ identifies the lead vehicle has stopped and starts decelerating until a complete stop behind $v2$ in $t(4)$.

Figure 5.2 (b) shows a *lane-change scenario*. In $t(1)$, the vehicle is driving on current lane. The trigger for a lane-change can be defined in the Behavior Tree as required by the scenario, or as an automatic action to keep the vehicle on route (for example, to take a turn using a parallel lane). In $t(2)$, the vehicle projects a trajectory to the right lane, and proceeds with the swerve in $t(3)$. In $t(4)$, the vehicle finishes the lane change and continues driving. Note how between $t(3)$ and $t(4)$, the reference path changes to a new lane (blue dotted line in the Cartesian frame), and the origin point in the Frenét frame ($D = 0$) is at the center of the new lane. The vehicle lateral state changes from $d < 0$ to $d > 0$.

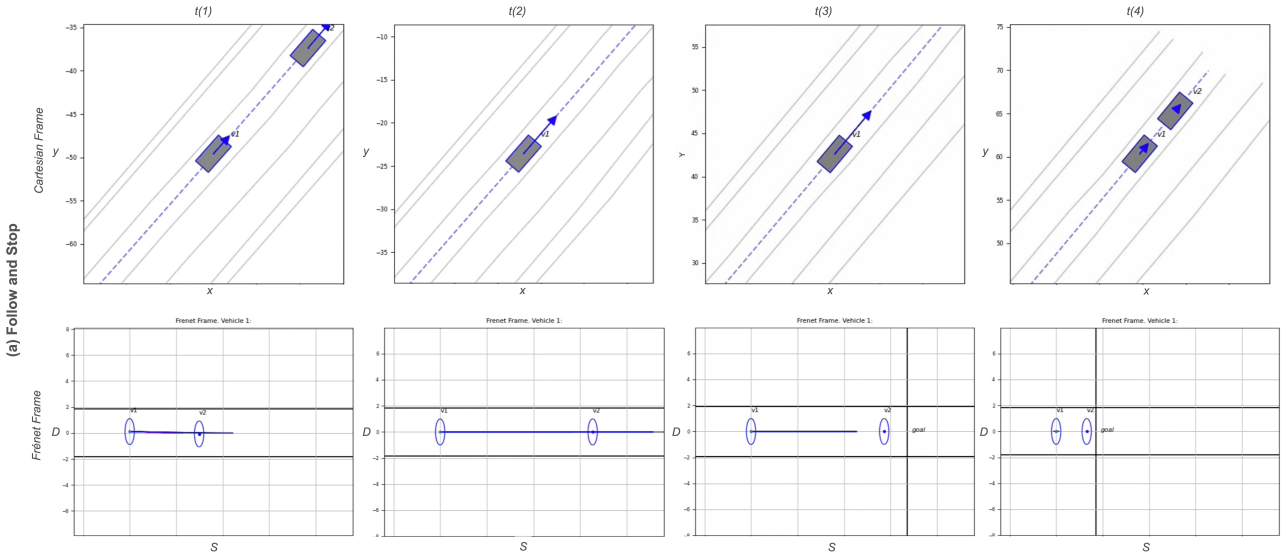
Figure 5.2 (c) shows an *obstacle avoidance scenario*. In $t(1)$, the vehicle identifies an obstacle on the road and begins sampling the lateral space (uniformly, by default). In ($t2$), the feasibility constraints from the projected trajectories identify a collision (red trajectories in Frénet frame) and the cost optimization results in the vehicle swerving away from the object while keeping the vehicle inside the lane boundaries. In ($t3$), the vehicle is safely away from the object and keeps its position on the right side of the lane. In $t(4)$ a new object arises, forcing the vehicle to immediately swerve to the left side of the lane.

Figure 5.2 (c) shows an *all-way-stop intersection scenario*. In $t(1)$, both vehicles $v1$ and $v3$ are approaching the intersection at the same time. Vehicle $v1$ projects a decelerating trajectory until the stop line. In ($t2$), two new vehicles arrive and all vehicles are waiting to cross the intersection in a deadlock. In $t(3)$, $v1$ decides to move first, while $v3$ waits and eventually crosses in $t(4)$. Note the grid in the Frénet frame represents vehicles in each *occupancy zone* surrounding the vehicle. The zones are part of the estimated *Traffic State* and can be used for conditions that compose decisions in the Behavior Tree.

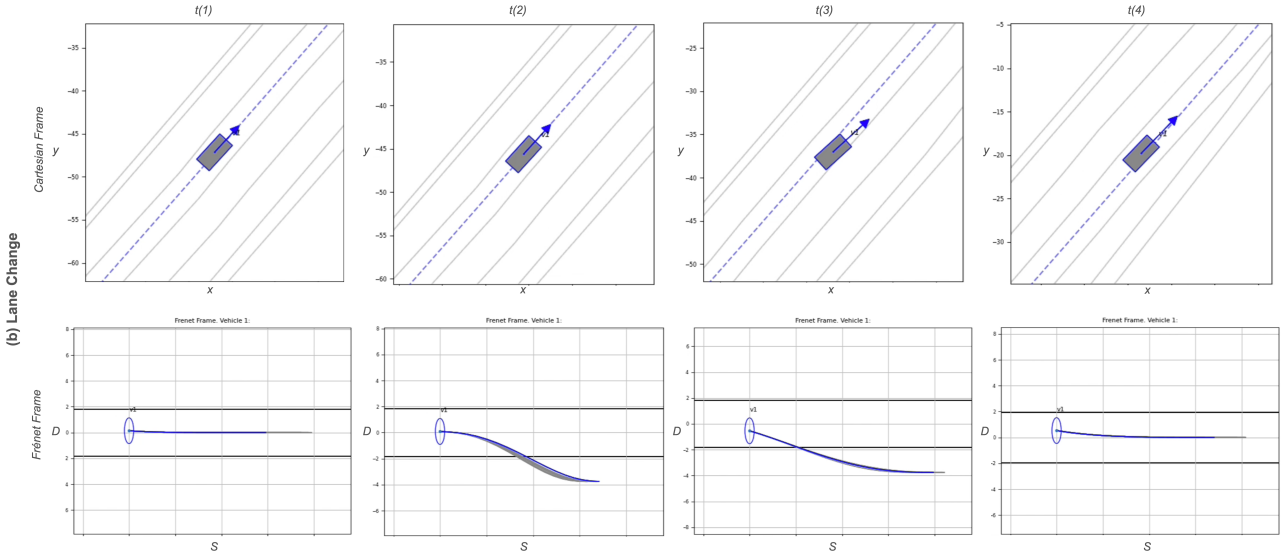
5.3 Integration and Co-Simulation

The implementation also provides a sample integration with an existing simulator, WISE Sim, and an ADS software stack, WISE ADS (see Figure 5.4). The GSClient component connects to the shared memory interface between the GeoScenario Server and WISE Sim, which runs within the Unreal game engine and provides lidar and camera simulation. The high-fidelity dynamics model of the Ego vehicle, a Lincoln MKZ, runs as a Robot Operating System (ROS) [9] module along with the WISE ADS.

The GeoScenario Server can be integrated into any other simulation environment, simply by customizing GSClient for the new environment (shown in the cut-in example in

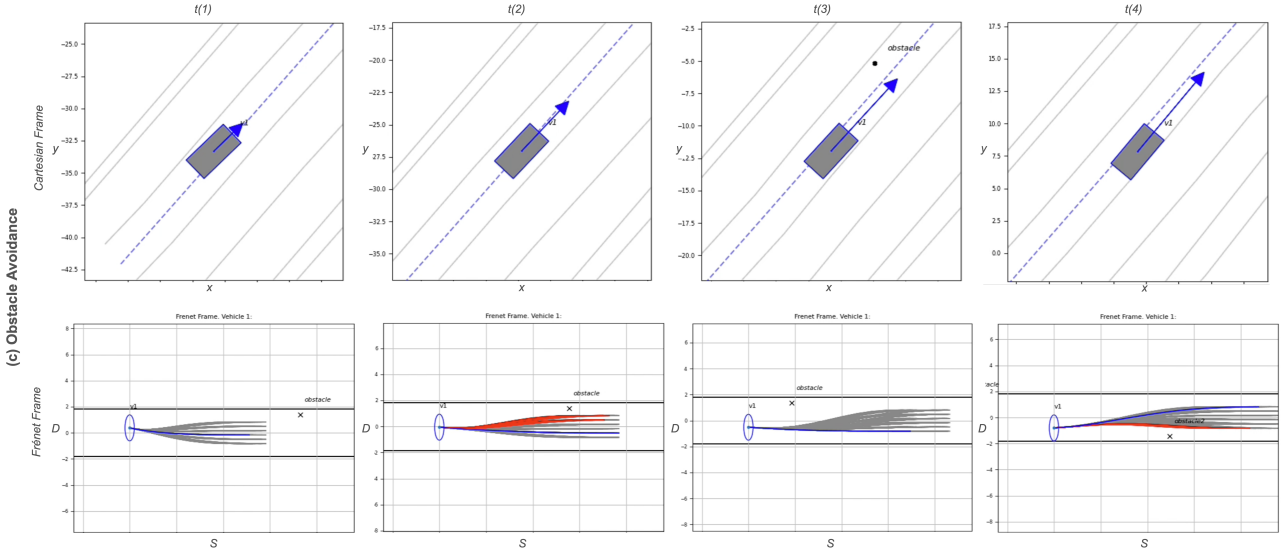


(a)

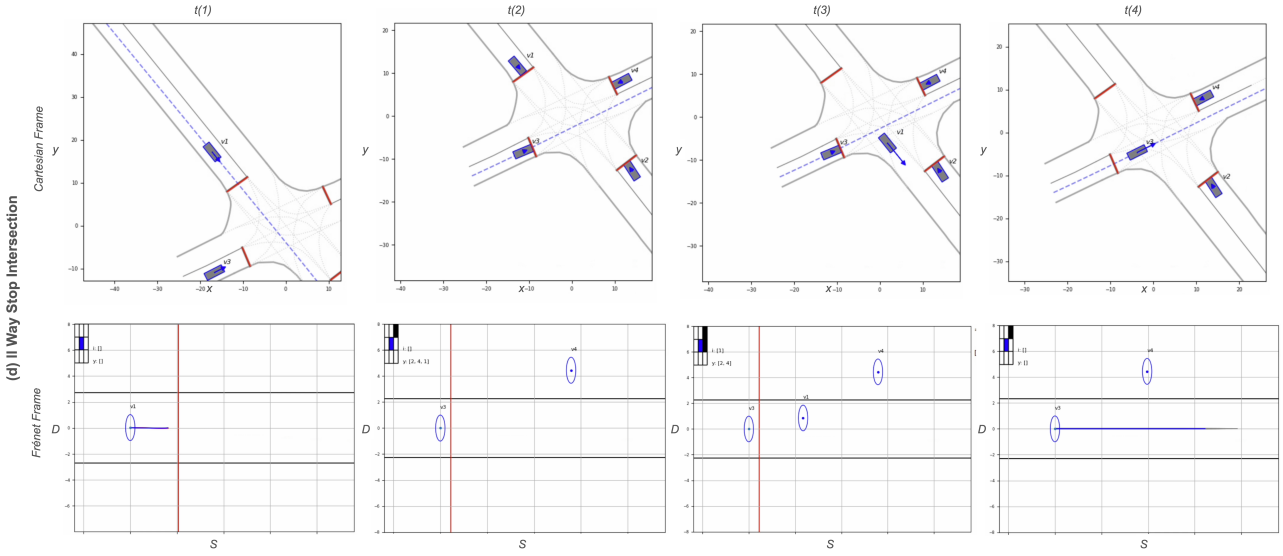


(b)

Figure 5.2: SDV Model in Simulation. Following a vehicle (a) and changing lanes (b).



(c)



(d)

Figure 5.3: SDV Model in Simulation. Maneuvering around static objects (c), and handling an All-Way-Stop Intersection with multiple vehicles (d).

Figure 6.5). The implementation provides a collection of sample scenarios, BTs, and maps covering different traffic situations. All tools are available to the research community and can run scenarios out-of-the-box. More details are available in the online documentation.

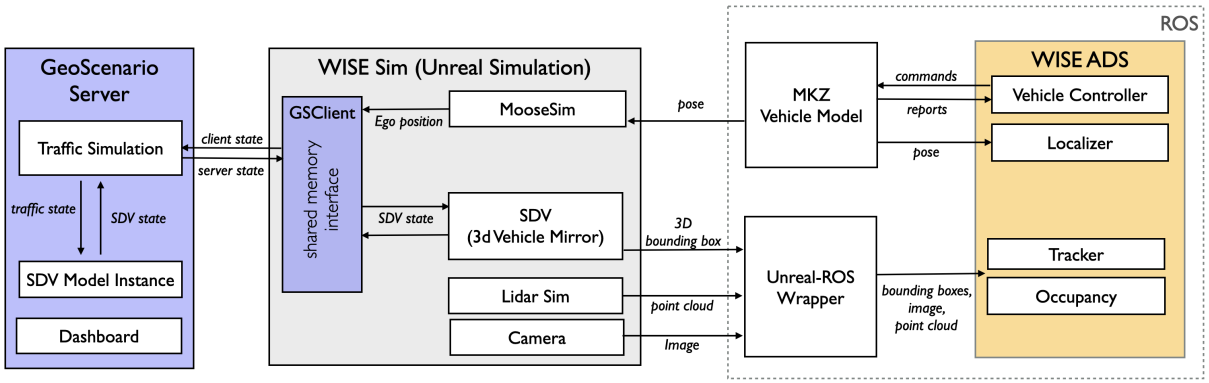


Figure 5.4: SDV Model integration with the co-simulation environment (GeoScenario Server + WISE Sim) and the subject system under test (WISE ADS). For simplicity, all vehicle-related modules are abstracted as SDV Model Instance

We also provide an experimental integration to run scenarios in co-simulation with CARLA [30]. The SDV model is able to drive with mixed traffic, when part of the vehicles are CARLA Agents. This is possible because the driving task do not require any knowledge of the underlying model driving the vehicle or any form of vehicle-to-vehicle communication. All traffic participants are interpreted based on their external state and visibility from the vehicle's point of view.

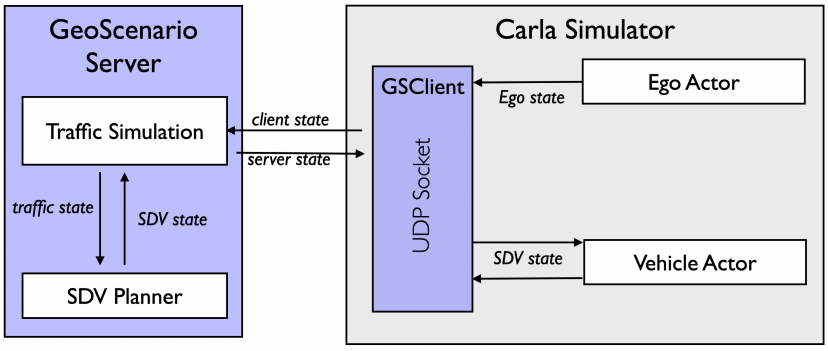


Figure 5.5: SDV Model integration in co-simulation (GeoScenario Server + CARLA)

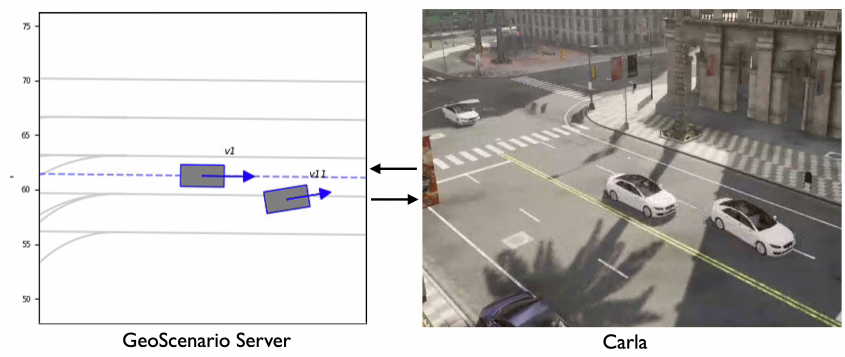


Figure 5.6: SDV Model running in CARLA using co-simulation

5.4 GeoScenario design

A custom JOSM can be used as a graphical editing tool to help designing and understanding Geoscenario definitions on top of the Road Network (Lanelet layer). Additional imagery map layers (e.g., Bing Maps, ESRI maps) can also be used if the scenarios are based on real roads. This customization is available as a set of presets and style sheets that can be added to the original JOSM tool (see Figure 3.5). We highly recommend using the graphical interface when creating new scenarios. Alternatively, scenarios can be created directly as XML files. The Behavior Trees for SDV models are written in separate files and used as catalogs of driving behavior that can be assigned to individual vehicles in a scenario.

5.5 Applications

Projects using GeoScenario integrated with WISE Sim and WISE ADS:

- *Autonomous Driving: Mapping and Behavior Planning for Crosswalks*: The project is a study exploring pedestrian crosswalk scenarios. The goal is to evaluate methods of balancing safety, assertiveness, caution, and obstruction to the traffic flow when interacting with pedestrians in a configurable driving policy. Using GeoScenario, a researcher manually designed ten pedestrian scenarios, and generated five additional scenarios using data collected directly from a a crosswalk located at the University of Waterloo campus Ring Road. This project explores the pedestrian paths with speed

profiles for realistic motion, and trigger features from GeoScenario. A configurable driving policy for the WISE ADS system is introduced with results [26].

- *WISE Bench: A Motion Planning Benchmarking Framework for Autonomous Vehicles*: This project aims at developing a benchmark for Motion Planning. The goal of this study is to provide a comprehensive set of scenarios covering a diverse range of road types (e.g., straight road, cul-de-sac, intersections) in interactions with both vehicles and pedestrians. The project explores scenarios in a composition format with increasing levels of difficulty, and relies on a metrics module to evaluate a systems overall performance. All scenarios from the benchmark are designed with GeoScenario language and executed in simulation with WISE Sim. This project is the most extensive use of the DSL, with more than 70 scenarios exploring many of the GeoScenario features [37].

- *AAA AV Test Methods project*: A collaboration between WISE Lab and the Autonomous Vehicles team of AAA. In this project, scenarios are performed in a testing track using the Autonomoose vehicle. GeoScenario is used to formalize the scenario choreography for the testing track execution, and to run the same scenarios in simulation. The differences between the real track and simulation executions are compared, and the causes of differences are used to suggest further development of the simulator towards more consistent results. The scenarios involve the subject vehicle (Ego), a second vehicle, and a pedestrian (in the test track, the pedestrian is a test dummy mounted on a robotic platform from Dynamic Research Inc. "Micro LPRV" [3]). The project explores six different scenarios imposing challenges to the subject vehicle. For example, a left turn with occlusion, a parking car close to an intersection causing Ego to yield, and a near-crash between the subject vehicle and a pedestrian. Results are available in the 2020 SAE Technical paper [20].

- *A Hierarchical Pedestrian Behavior Model to Generate Realistic Human Behavior in Traffic Simulation*: This project extends the SDV Model to create a realistic and highly controllable pedestrian simulation model. The hierarchical pedestrian behavior model generates high-level decisions through the use of behavior trees, and executed by a low-level motion planner using an adapted Social Force model. A full implementation of the work is integrated into GeoScenario Server, extending its vehicle simulation capabilities with pedestrian simulation. This integration allows simulating test scenarios involving both vehicles and pedestrians to assist in the scenario-based testing process of autonomous vehicles. The model is shown to

[3] http://www.dri-ats.com/lprv/

replicate the real-world pedestrians' trajectories with a high degree of fidelity and a decision-making accuracy of 98% or better, given only high-level routing information for each pedestrian. Results are available in the IV 22 paper [50].

Chapter 6

Evaluation

In this Chapter we evaluate GeoScenario and the SDV model in terms of scenario development effectiveness, realistic vehicle motion, practical applicability for scenario-based ADS testing, and finally scalability. The following research questions guide our evaluation:

- **RQ1**: Can realistic and interactive scenarios for ADS testing be effectively modeled and executed via GeoScenario and SDV models?
- **RQ2**: Can SDV models generate realistic vehicle motion?
- **RQ3**: Can using GeoScenario and SDV models improve the effectiveness of scenario-based testing of a real ADS?
- **RQ4**: How does the model performance scale with traffic density?

6.1 Effective Scenario Development (RQ1)

We evaluate the effectiveness of scenario development using GeoScenario and the SDV model to design and execute test scenarios from an independent catalog. We conduct an experiment where we manually design scenarios from the catalog, execute them in simulation, and observe the scenario evolution and vehicle behaviors. We evaluate the performance in terms of expressiveness, execution accuracy, and behavior reuse. Further, we analyze how the SDV model improves the original GeoScenario as the baseline DSL. Finally, we discuss limitations and challenges we found during the experiment, the types of

scenario that improved the most with the inclusion of SDV models, and how our approach can improve scenario based-testing with such scenarios.

> **Research Question 1**: Can realistic and interactive scenarios for ADS testing be effectively modeled and executed via GeoScenario and SDV models?

6.1.1 Metrics

We evaluate the effectiveness of scenario development using three metrics:

- (i) *expressiveness*: the breadth of scenarios that can be represented with the model. Given a set of scenarios, we classify each scenario according to whether the required behaviors for all vehicles in the scenario could be modeled. We assign *success* (S) when all behaviors are successfully expressed with no limitations, *partial* (P) when the behaviors for at least one variation of the scenario can be expressed, or *failure* (F) when the minimum behavior required for a scenario cannot be expressed.

- (ii) *execution accuracy*: accuracy of the scenario and vehicle behavior during simulation with respect to scenario objective. Even when a scenario can be represented with the language, the intent in design may not always translate to correct execution (this challenge is explored in [20]). After running a simulation, we classify the degree to which scenarios are correctly executed: *success* (S) when all vehicles behave as expected and the scenario objective is achieved; *partial* (P) when at least one variation of the scenario succeeds; and *failure* (F) when vehicles deviate from the design intent and thus the scenario execution fails.

- (iii) *reuse*: the ability to reuse behavior across scenarios to reduce design effort. Reuse in software is a key method for improving quality and productivity [32]. In GeoScenario, behavior is defined using BTs, which can be reused by importing from a shared library, composing using sub-tree references, and configuring using parameter values. We quantify reuse in a scenario based on the *internal reuse level* from Frakes and Terry [32]. When applying this metrics, BTs are considered as higher-level items, which consist of nodes as lower-level items. Given a scenario containing a set of BTs (higher-level items), the metric is defined as M/L, where M is the number of nodes (lower-level items) that are used more than once (i.e., used also in BTs of other scenarios) and L is the total number of nodes in the set of BTs. This metric assumes values between 0 and 1 and represents the percentage of internal reuse. The

remaining percentage represents custom BT code, such as custom sub-trees, required to implement behavior that is specific to the particular scenario. Note that each BT in the library is used by at least two scenarios. Since a scenario may not use all the nodes of the BTs it imports from the library, we also compute the internal reuse level for a given scenario accounting for only the nodes that are actually executed in a successful simulation.

6.1.2 GeoScenario PDT versus GeoScenario with SDV Model

Since the SDV model extends the capabilities of the original GeoScenario (Chapter 3), we use the latter as the baseline. In the original language, the vehicle behavior is composed of predefined trajectories (PDT) with speed profiles and triggers to change them at runtime [59]. This common approach is also supported by other simulation tools, including PreScan [4] and VTD [13], and scenario definition languages, including OpenScenario [8] and M-SDL [5].

6.1.3 Scenario Catalog and Test Set

SDV models can be used in a wide variety of scenario designs and test cases requiring Ego-to-HV interactions, which naturally leads to a large design space to explore and it becomes difficult to show effectiveness. In order to make the evaluation feasible, we focus on safety-critical scenarios that account for the majority of crashes in traffic. We use the Pre-Crash Scenario Typology from NHTSA [54] to compose this evaluation set. This scenario catalog provides interactive and realistic scenarios that can challenge the ADS capabilities in crash avoidance and are commonly used as a reference for ADS validation in other projects [16, 30]. They are *interactive* in the sense that they contain crash and near-crash events caused by vehicle-to-vehicle interactions. They are *realistic* in the sense that they are based on a large collection of actual crash reports. NHTSA scenarios are based on the 2004 General Estimates System (GES) crash database with 5 942 000 police-reported crashes occurred in the United States including at least one light vehicle.

Since the SDV models target scenarios that require Ego-to-HV interactions, we filter the original set for situations with vehicle-to-vehicle interactions, resulting in 18 scenarios. Therefore, scenarios based on a single vehicle crash (e.g., control loss), or involving other traffic participants (e.g., pedestrians, pedalcyclists, animals on the road) are not applicable. The list is not exhaustive in terms of coverage of the entire scenario design space, but together they represent 69.65% of all light-vehicle crashes [54]. Table 6.1 shows the final

scenario set classified into groups. The ID we use is the same as the original NHTSA Report. In Appendix C we show the complete list of 37 scenarios from NHTSA with additional details on the selection process.

6.1.4 Turning NHTSA Scenarios into Test Scenarios

We design each scenario using a combination of the original GeoScenario and multiple instances of SDV models. Each instance is based on a collection of BTs and maneuver configurations representing the behavior of one or more vehicles interacting with Ego. The original NHTSA set is based on reported events between HVs, but we assume that one of the HVs is Ego, operated the ADS (similar to how Waymo adapts NHTSA scenarios as tests [16]).

A test scenario must also have goals and a clear success/fail criteria. Ego's goal is to drive through the scenario (from start to goal point) and avoid a collision. The goal of an SDV is to interact with Ego using target parameters defined by the tester, e.g., achieving a certain time gap before braking. The overall scenario goal is to replicate the pre-crash events as described by NHTSA, leading to a crash or a near-crash. If execution differs by either a safe outcome (vehicles never interact or interact differently than intended) or another type of crash, the scenario execution fails. After modeling the scenarios, we execute them in simulation using the reference implementation (Chapter 5).

As part of the comparison of expressiveness with the baseline, we classify the type of SDV behavior required in each scenario as *static* or *dynamic* with respect to three elements: *path shapes*, *speed profiles*, and *behavior triggers*. Behavior triggers are conditions triggering the required changes in paths and speed profiles during the scenario (Table 6.1). Scenarios that involve static behavior for all three elements, i.e., fixed paths and speed profiles for each SDV and their starting triggers, can be easily designed with PDTs from start to finish and do not benefit significantly from a dynamic model (stat,stat,stat in Table 6.1). Scenarios that require dynamic behaviors, but the behaviors can be expressed as sets of static paths and velocity profiles with dynamic triggers to select among them (stat,stat,dyn in Table 6.1), can still be modeled using PDTs with reasonable effort. Finally, scenarios that require dynamic path or velocity profile or both (dyn,stat,*; stat,dyn,*; and dyn,dyn,* in Table 6.1) are impractical to be modeled using PDTs, but are enabled by the proposed SDV model. For example, the cut-in scenario has a continuous space of paths and speed profiles, and a dynamic trajectory needs to be planned based on the Ego behavior, which may vary from execution to execution.

We note that using the NHTSA descriptions of the scenarios as a source, many scenario

variants are possible. Our classification is based on the minimal behavior required to reproduce the critical event occurring immediately prior to a crash as described by NHTSA; however, added elements, such as additional vehicles, might change the static classification to a dynamic one, but not the other way.

6.1.5 Results

Expressiveness

All 18 scenarios except for one variant of #17 are successfully expressed using the SDV model. We identify 14 scenarios (78%) that depend on dynamic path or velocity profile or both and are thus impractical for the baseline. For instance, a vehicle leaving a parking position in scenario #17 must start this maneuver only when Ego is approaching and adjust its trajectory, in one of the variants, to merge ahead of Ego. While the vehicle must challenge the ADS, an unavoidable lateral crash into Ego would not be useful as a test scenario. To achieve the scenario goal, the vehicle must be able to generate a trajectory relative to Ego's motion at run time. The same requirement applies to all lane-change scenarios (#16-#19). For crossing-path scenarios #30 and #31, the velocity profile must be dynamically planned. The SDV models enable us to successfully express these dynamic behaviors, which are infeasible with the baseline, resulting in a higher expressiveness. One variant of Scenario #17 "Parked Vehicle SD" requires the parked vehicle to join traffic by making a U-turn, and this maneuver is currently not supported by the implementation of trajectory generation.

A total of four scenarios (22%) require only static trajectories (stat,stat,* in Table 6.1) and thus can be designed with the baseline. For instance, in the rear-end scenario #25 both path shape and speed profile can be generated offline and expressed as PDTs with only a trigger to activate the deceleration as Ego approaches. In such examples, the SDV model does not increase expressiveness. However, it adds two advantages: (i) conciseness, by defining the scenario at a higher level of abstraction using target parameters instead of detailed trajectories, and (ii) flexibility, by allowing the scenario to be replicated in different road geometries without changing the behavior definition. We also observed improvement in the conciseness of the scenario description in comparison with the use of PDT. All scenarios could be simplified by avoiding detailed trajectories and adopting a higher level of abstraction from Behavior Trees and Maneuver Configuration using targets.

Execution Accuracy

We run simulations to evaluate if the simulated vehicles perform as expected and if the scenario execution resulted in the pre-crash situation as described by NHTSA. In 17 scenarios, vehicles perform as expected, and the scenario ends with a crash or near-crash as described in the NHTSA report. The performance deviates from the design in the scenario #16 "Vehicle(s) Turning – Same Direction". The assigned behavior requires that vehicles perform a maneuver that violates the legal road-network connectivity. Since the current implementation relies on the Lanelet map to constrain the driving space, the map required an adaptation to execute the scenario correctly.

Behavior Reuse

The composable nature of BTs allows us to reuse most of them, i.e., use each BT in two or more scenarios, since there is significant commonality in the driving task for the different scenarios. In most scenarios, vehicles start by performing normal lane maintenance or vehicle following until an unexpected event occurs, such as a risky behavior of another vehicle. The differences among scenarios emerge in such events and are usually modeled at the highest levels of the main BT for the given scenario. We call them the "*scenario-trees*". The remaining tasks are reusable and performed using "*sub-trees*" (e.g., performing a lane-change). This reuse pattern is not part of the original BT concept, but it has emerged during this experiment when trying to maximize reuse.

In some instances, a simple overriding of parameters for conditions or maneuvers during the sub-tree composition is sufficient to adapt the behavior from one scenario to another and achieve the scenario objective with 100% reuse (see *Internal Reuse Level* in Table 6.1). Overall, the average internal reuse level (weighted by the size of behavior trees in each scenario) is 0.93. Considering only the nodes executed during a simulation, the average is 0.81. The results are summarized in Table 6.1.

Summary of Results

The experience modelling and running NHTSA scenarios reveals how effective the SDV model can be in ADS scenario development. The model enables expressing highly-dynamic behaviors, fosters reuse and can successfully execute most scenarios in simulation. Vehicle interactions involving lane changing, merging, and crossing paths are severely limited or impractical using the PDT baseline. Thus, such interactive scenarios benefit most from the SDV model.

Table 6.1: Scenarios and performance

ID	Group	Scenario	Path Shape	Speed Profile	Behavior Trigger	Expressiveness	Execution	IRL	IRL exec
4	CP	Running Red Light	stat	dyn	stat	S	S	0.83	0.60
5	CP	Running Stop Sign	stat	dyn	dyn	S	S	1.00	1.00
15	B	Backing Up Into Another Vehicle	stat	stat	dyn	S	S	0.91	0.60
16	LC	Turning SD	dyn	dyn	dyn	S	S*	0.87	0.63
17	LC	Parking SD	dyn	dyn	dyn	P	P	0.84	0.71
18	LC	Changing Lanes SD	dyn	dyn	dyn	S	S	0.89	0.79
19	LC	Drifting SD	dyn	dyn	dyn	S	S	0.79	0.71
20	OD	Making Maneuver OD	dyn	dyn	dyn	S	S	0.90	0.84
21	OD	Not Making Maneuver OD	dyn	dyn	dyn	S	S	0.76	0.50
22	RE	Following Vehicle Making Maneuver	dyn	dyn	dyn	S	S	1.00	1.00
23	RE	Lead Vehicle Accelerating	stat	stat	dyn	S	S	0.90	0.75
24	RE	Lead Vehicle at Lower Speed	stat	stat	stat	S	S	1.00	1.00
25	RE	Lead Vehicle Decelerating	stat	stat	dyn	S	S	0.90	0.75
27	CP	Left-Turn Across Path/OD at SJ	stat	dyn	dyn	S	S	0.90	0.75
28	CP	Vehicle Turning Right at SJ	stat	dyn	dyn	S	S	0.99	0.94
29	CP	Left-Turn Across Path/OD at NSJ	stat	dyn	dyn	S	S	0.98	0.93
30	CP	Straight Crossing Paths at NSJ	stat	dyn	dyn	S	S	0.94	0.81
31	CP	Vehicle Turning at NSJ	stat	dyn	dyn	S	S	0.94	0.81

Acronyms: B: Backing up, CP = Crossing Paths, LC = Lane Change, OD = Opposite Direction, RE = Rear-end, SD = Same Direction, SJ = Signalized Junction, NSJ = Non-Signalized Junction. Path Shape, Speed Profile, and Behavior Trigger are requirements for vehicle behavior that can be static (stat) or dynamic (dyn). Expressiveness and Execution show the degree in which a scenario is modeled and correctly executed, respectively (S=successfully, P=partially, F=Failed). The internal reuse level is computed with all Behavior tree nodes (IRL), and only for nodes that are executed in the simulation (IRL exec). *Scenario #16 required a map adaptation to perform correctly.

The limitations we identify are due to missing underlying maneuvers (such as a U-turn, which is difficult to implement when planning in Frénet frame of a lane) or the map constraints that prevent certain vehicle movements. We will address them in future work. Based on the NHTSA statistics, the scenarios expressed and executed successfully with the SDV model represent about 49% of all light-vehicle crashes in traffic.

6.2 Vehicle Motion Realism (RQ2)

In this section we evaluate the motion realism of vehicles running SDV models. As the primary goal is to simulate human-operated vehicles, a good model must reflect the human-driving behavior and how vehicles move in naturalistic traffic conditions. To evaluate the motion realism, we use SDV models to replicate scenarios collected from urban traffic and compare their behavior with real vehicles.

It is unreasonable to expect SDV models to drive exactly like the empirical vehicle, since not even humans drive equally. However, our model is designed to be highly configurable and adapt to different driving styles. With the proper configuration in the calibration process, we expect that SDV models can approximate the behavior of the empirical vehicles to a high degree given the same environment conditions. We evaluate the performance of the model in 100 scenarios based on a measure of trajectory distance, and discuss how different configurations can further approximate the synthetic behavior to real vehicles in traffic.

Research Question 2: Can SDV models generate realistic vehicle motion?

6.2.1 Naturalistic Dataset

We use data from a busy signalized intersection during mid-day traffic in Waterloo, Canada, which is part of the Waterloo Multi-Agent Traffic Dataset [14]. The "birds-eye" image was collected using a drone and processed to label and track pedestrians and vehicles (Figure 6.1).

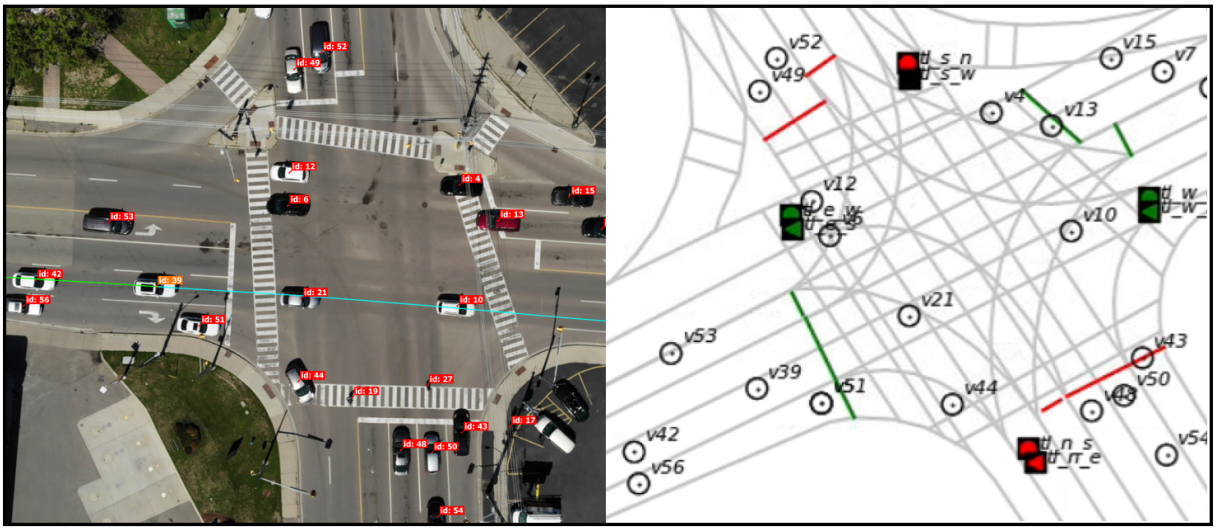


Figure 6.1: A snapshot of the signalized intersection used for experiments and its corresponding simulation on the right.

6.2.2 Experiment

This experiment follows four steps:

1. *Data preparation:* We classify the vehicle trajectories in the dataset into five scenario types based on the main maneuver they represent: (i) vehicle crossing intersection unconstrained (free), (ii) vehicle stopping (red light), (iii) vehicle resuming driving (green light), (iv) vehicle following a lead through the intersection (follow), and (v) vehicle partly following a lead when the lead merges or leaves mid-scenario (free/-follow). In cases where a vehicle stops at a signal light, we split the trajectory into two scenarios, namely (ii) and (iii), in order to eliminate the waiting state where a simulated trajectory can trivially match the empirical vehicle. Each such classified vehicle trajectory represents an individual experimental trial.

2. *Test generation*: For each classified vehicle trajectory, we identify the traffic conditions that may affect how the vehicle is driving, e.g., signal light states and all other vehicles and pedestrians that may affect it, to be reproduced in simulation. Each classified vehicle trajectory is used as a reference vehicle for a single test. We generate a new GeoScenario test replacing the reference vehicle with an SDV model instance with a standard driver BT and using the same start state (velocity and position in

the intersection), and replicate the traffic conditions to ensure the driving task is influenced by the same factors. The standard driver BT is capable of performing each of the five maneuvers. We also assign a route goal to the model based on the last known position of the empirical reference vehicle to ensure the simulated vehicle will navigate the intersection towards the same exit lane. All other relevant empirical vehicles and pedestrians are included in the test as agents with PDTs, and the signal light phases are also replicated. We generate 100 test scenarios and manually review the correctness of the identified traffic conditions.

3. *Calibration*: While each simulated reference uses the same standard-driver BT, it needs a BT configuration to replicate the driving style of its empirical counterpart. We use a set of rules to automatically analyze each empirical reference trajectory and generate a configuration for it by extracting a set of high-level driving-style parameter values and value ranges, including maximum and average velocities, lateral displacement on the lane, stopping distance to target, reaction times, and time gap to other vehicles. We adjust the SDV parameter ranges to target similar values.

4. *Simulation*: We run two simulations per scenario using the SDV model, one with a default configuration before the calibration and another one after the calibration, and export the resulting trajectories as a discrete set of the vehicle states in the simulation frequency at 30 Hz. The default configuration uses nominal naturalistic driving parameters, such as zero offset from the lane centerline and a time gap range of 1.8..2.2 s [28].

The SDV performance is assessed using a measure of distance between the simulated trajectory T_1 and the empirical reference trajectory T_2, which takes into account both their spatial and temporal characteristics. The shorter the distance, the more similar the motion behavior of the simulated and the empirical vehicle. We use the spatio-temporal Euclidean distance (STED) [56], which represents the average Euclidean distance between positions of the respective vehicles, $T_1(t)$ and $T_2(t)$, along their respective trajectories T_1 and T_2, over the interval l in which both trajectories exist:

$$d_{STED}(T_1, T_2) = \frac{\int_l d(T_1(t), T_2(t))\, dt}{l} \tag{6.1}$$

6.2.3 Results

Figure 6.2 shows the distribution of STED before and after calibration per scenario type. The majority of simulated trajectories are already fairly similar to their empirical reference even before the calibration with an average STED of 4.27 m. A review of the simulated trajectories shows a similar decision making patterns, such as reacting to traffic lights and vehicles ahead, to the empirical ones. However, the main differences are observed in the speed profiles, lateral placement on the lane, time gaps, and various delays and reaction times, all indicative of different driving styles. The calibration brings the simulated trajectories significantly closer to their empirical counterparts: average STED for all 100 scenarios reduces from 4.27 m to 1.24 m. At an individual level, calibration improves the performance in 82 scenarios. Although the performance is worse for 18 scenarios, it is only slightly worse for 16 of them, with less than 1 m deterioration.

Only two scenarios deteriorated more significantly, by 1.4 m and 1.9 m. The latter deviation is due to an erratic driving style of the empirical reference vehicle, which accelerates hard when resuming driving on green and then decelerates for no apparent reason. Such erratic behavior could be replicated by a dedicated maneuver. Further note that the improvement from calibration is most pronounced for (i) free driving by matching the average speed of the empirical vehicle, and (ii) signal light handling by matching the delay to resume driving on green.

Figures 6.3 and 6.4 show the paths and speed profiles of sample individual scenarios. Plot (a) shows the reference vehicle 5 reacting to a red light. The path before calibration shows the simulated vehicle stop at the stop line, but the empirical vehicle stops about 2.5 m before the line. After calibration, both the simulated and empirical paths match up almost perfectly, with an STED of 17 cm, and a maximum distance of 31 cm. The calibrated speed profile also closely matches the empirical one. Plot (b) shows vehicle 97 crossing the intersection southwards, while already following a lead vehicle. The black dashed line shows the lead vehicle's speed profile, which is fairly constant throughout the scenario. The initially slower reference vehicle accelerates to match the lead's speed. The calibration improves the default configuration to match the more aggressive time-gap of the empirical vehicle, resulting in closely matched speed profile and reducing the STED from 2.37 m to 17 cm.

In rare cases, the calibration does not improve performance, as shown in plot (c). A vehicle approaches the intersection with a red light and an already stopped vehicle ahead. After waiting for the green light, the reference vehicle can resume driving but needs to keep a following distance from the lead vehicle. The simulated vehicles resume with a smaller delay compared to the empirical one.

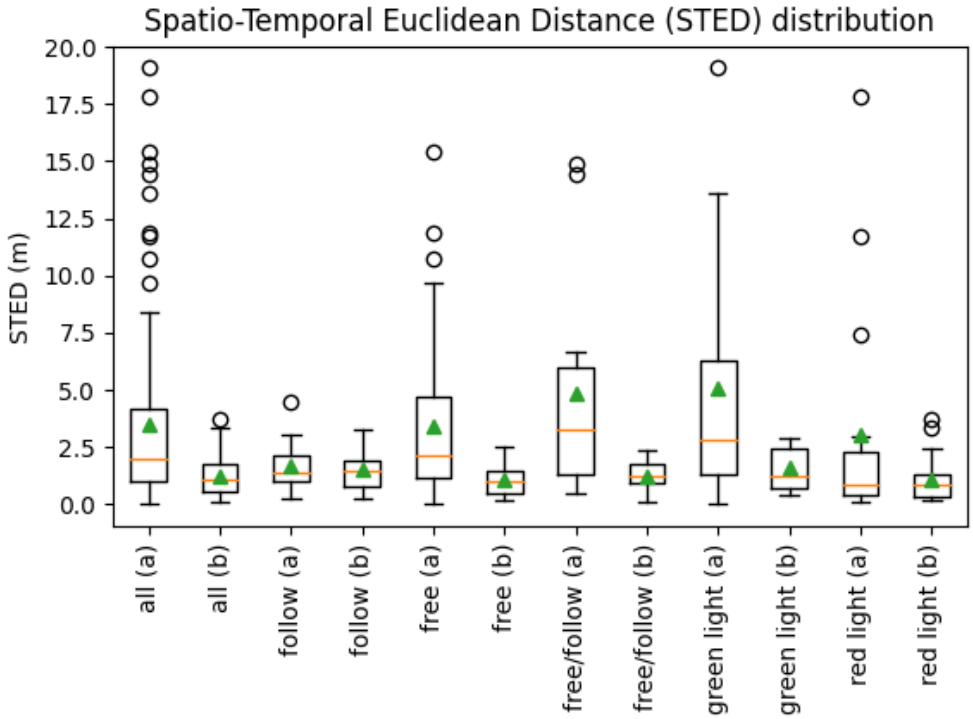


Figure 6.2: Performance for all scenarios and per type, before (a) and after (b) calibration, measured using STED in meters. Orange lines represent medians, and green triangles represent averages.

In summary, SDV models can closely reproduce the behavior of human-driven vehicles under the same traffic conditions. Overall, the model calibration can address varying driving styles and significantly increase the similarities in the trajectories. In some scenarios, such as in Figure 6.3 (a), the simulated trajectory after calibration is in essence indistinguishable from the empirical one, with maximum difference of 31 cm. In some scenarios the human behaves unexpectedly, however, and the current automatic calibration process cannot replicate such behaviors, but they could be modeled in the BTs as additional maneuvers. All results, trajectory logs, speed and trajectory plots are available in the project website.

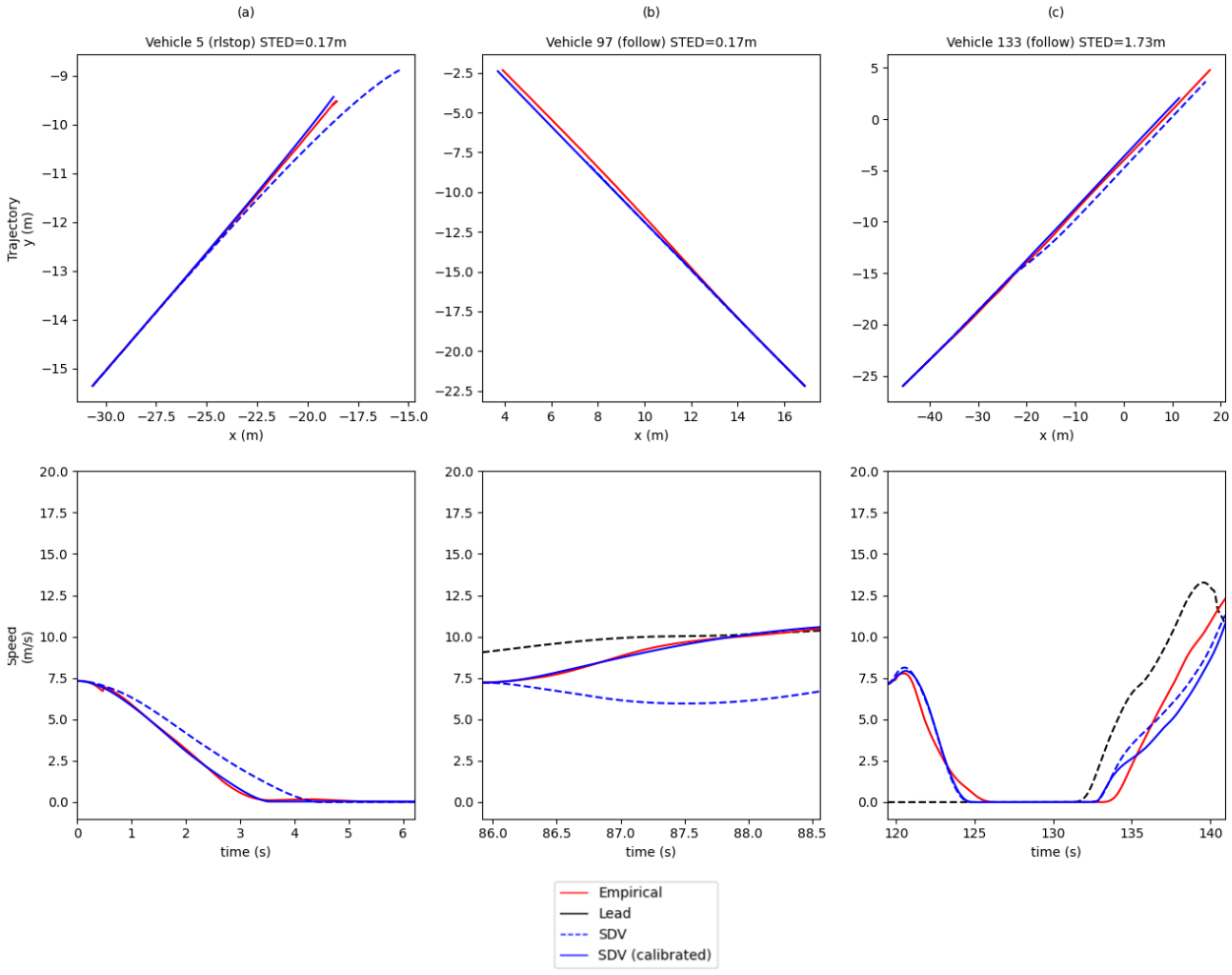


Figure 6.3: Paths and speed profiles for three sample scenarios. Empirical vehicles in red; SDV models in dashed blue (before calibration) and solid blue (after calibration).

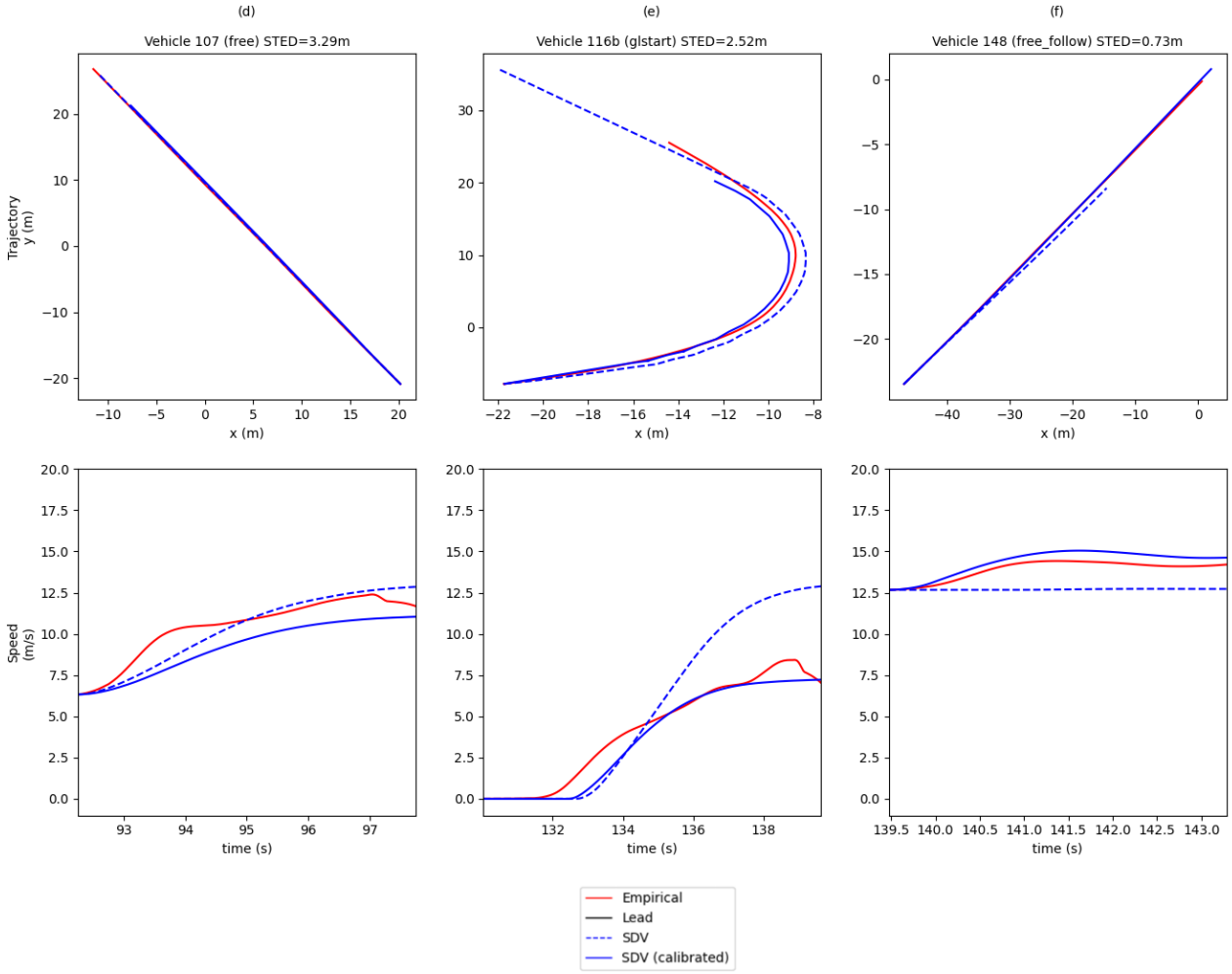


Figure 6.4: Paths and speed profiles for three sample scenarios. Empirical vehicles in red; SDV models in dashed blue (before calibration) and solid blue (after calibration).

6.3 Application (RQ3)

We run an in-depth case study to evaluate how the model performs in a real ADS testing environment and answer RQ3. We choose the cut-in lane change NHTSA scenario (#18 in Table 6.1) to test an actual ADS software as the subject system.

The case study has an explorative nature, with the objective to generate practical insights of applying the SDV model to test a real ADS, including identifying potential limitations.

> **Research Question**: Can using GeoScenario and SDV models improve the effectiveness of scenario-based testing of a real ADS?

6.3.1 The Cut-In Scenario

In this scenario, a vehicle changes lanes at a non-junction and merges closely in front of the Ego traveling in a adjacent lane in the same direction. After the maneuver, the lane-changing vehicle becomes the lead of the Ego. Cut-in maneuvers from other drivers pose challenges to the ADS, and if not handled properly can lead to crashes. In fact, this scenario accounts for 338 000 crashes or 6.69% of all the light-vehicle crashes in the NHTSA report [54]. Avoiding or mitigating them is an important goal for any ADS development program.

The goal of this test is to evaluate the ADS capabilities to handle cut-ins from other vehicles. Testers want to evaluate the impact of key vehicle interaction parameters, such as relative velocity and gap, on the likelihood and crash severity. The non-deterministic behavior of the subject ADS makes simulating this type of scenario challenging, however. Reaching the desired test parameter values while performing realistic vehicle interactions requires a reactive model, capable of adapting and re-planning trajectories as the scenario unfolds.

6.3.2 System Under Test

We test *WISE ADS*, developed at the University of Waterloo [15]. The ADS software consists of a set of ROS modules implementing object-detection and tracking, occupancy and high-definition mapping, localization and state estimation, maneuver and trajectory

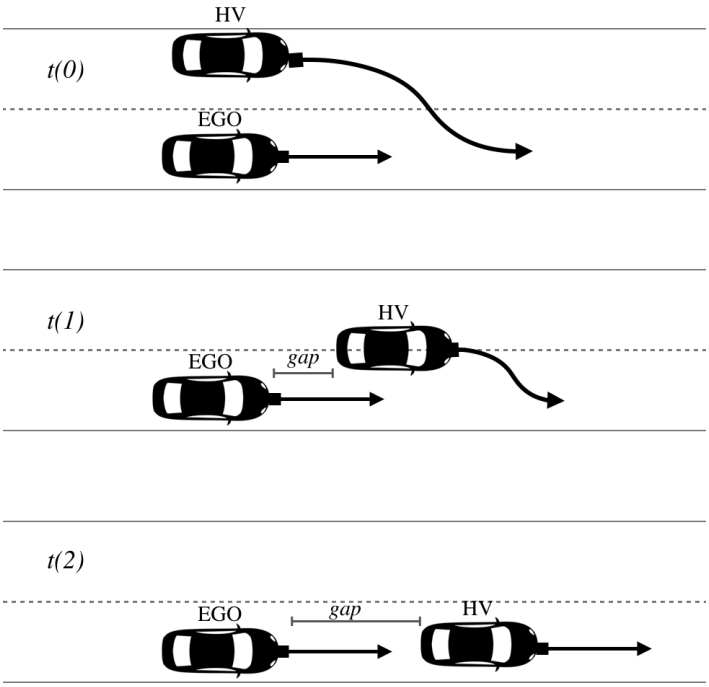


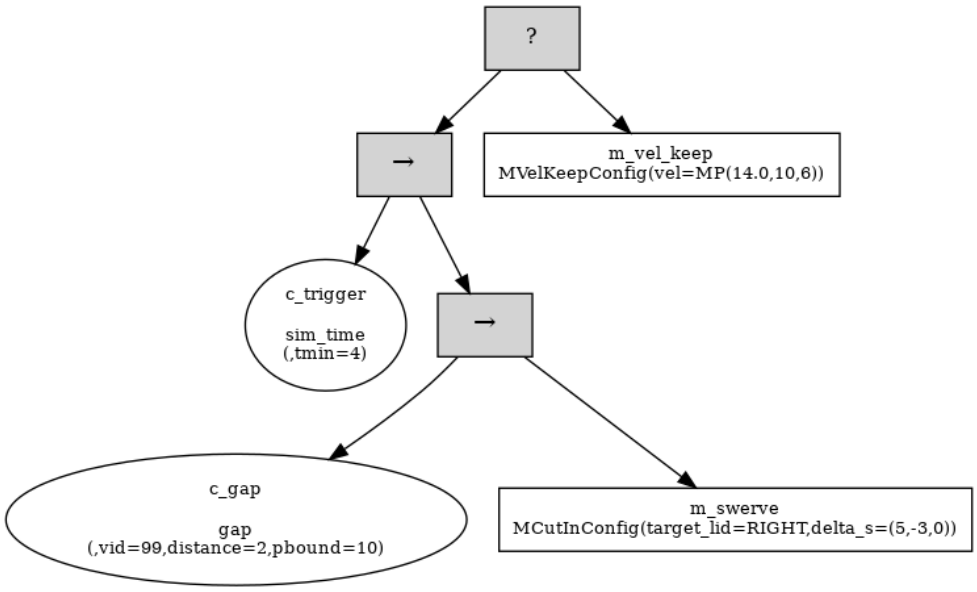
Figure 6.5: Scenario #18 - Vehicle Changing Lanes – Same Direction

planning, and control. The software can operate a Lincoln MKZ Hybrid, equipped with a drive-by-wire interface and a suite of lidar, camera, GPS, and inertial sensors, in automated mode at SAE level 3 (Figure 6.6). Note this is the same vehicle from the preliminary GeoScenario applicability demonstration in Section 3.4. However, the software under test is an updated version with improved mapping, occupancy, behavior planning, and local planning. We focus on testing the ADS software in simulation, using WISE Sim with the GeoScenario Server implementing the SDV model (see Figure 5.4). GeoScenario server simulates the SDV model instances and injects them into WISE Sim, with both environments operating in co-simulation. Then WISE Sim uses the ROS publish-subscribe mechanism to publish simulated sensor data from (GPS, IMU, cameras, and lidar) for the ADS and receives the Ego pose from the high-fidelity dynamics model of the Ego.

Figure 6.6: "UW Moose" research platform where the *WISE ADS* operates

6.3.3 Test Scenario

The cut-in behavior is expressed as a BT, such as the one in Fig. 6.7, and assigned to an SDV model instance. According to this BT, the vehicle must reach a certain acceptance (rear) gap before performing the cut-in maneuver and then achieve a certain target (rear) gap to Ego. The BT calls the standard drive BT (line 8) to maintain its current lane, parameterized with a target speed of 14 m/s (+-10%), which is slightly higher than the road speed limit. The simulation plans candidate trajectories by sampling 6 target velocities from this target range (uniformly, by default). After a delay to allow the vehicle to pick up pace (line 4), it starts checking for the acceptance distance gap (`distance`) of 5 m (+-10%) for a lane change to the right (`target_lane_id=RIGHT`), on which Ego drives at a speed matching the road speed limit (line 6). Once the acceptance gap is satisfied, the lane change is triggered (line 7), with a target distance gap of 5 m and a relative velocity of -3 m/s (`delta_s=(5,-3)`). The experiment repeats the scenario with different combinations of parameters to evaluate how Ego handles a variety of cut-in trajectories and find configurations that are more likely to lead to a crash.




```
behaviortree cut_in_driver:
    ?
        ->
            condition c_trigger( sim_time(t=4))
            ->
                condition c_gap(gap( target_lane=RIGHT, distance=5, pbound = 10) )
                maneuver m_swerve( MCutInConfig( target_lane=RIGHT, delta_s=(5, -3) ))
        subtree drive_tree(m_vel_keep=MVelKeepConfig( vel=MP(14.0,10,6) ))

```

Figure 6.7: Cut-in vehicle behavior using GeoScenario Behavior Tree DSL. Upper image is the graphical representation, while the lower image is the DSL form.

6.3.4 Results

As expected, more aggressive cut-ins are more likely to cause collisions, but the response of the ADS to different parameter combinations of the cut-in maneuver is non-obvious (see Table 6.2). Scenarios #7 and #8 are parameterized with the same short acceptance gap $\Delta d_a = 2\,m$ and the same target relative velocity $\Delta v_t = -5\,m/s$, but #8 has a smaller target distance gap, $\Delta d_t = -5\,m$, compared to $\Delta d_t = -2\,m$ for #7. As a result, #8 ends in a collision. Note that Δd_t and Δv_t are planned relative to the predicted Ego location at the end of the cut-in maneuver, assuming Ego continues at a constant velocity. Thus, although a negative Δd_t would guarantee a collision if Ego maintained its velocity, Ego is likely to brake and thus a negative Δd_t does not necessarily result in a collision. Scenarios #9-11 use a larger acceptance gap, with $\Delta d_a = 5\,m$. As a result, although #9 has the

same target parameters as #8, a collision is avoided, since the larger acceptance gap gives Ego more time to react. Increasing the target aggressiveness in #11 results in a collision, however.

Figure 6.8 shows scenario #8 with the SDV's trajectory generation (a-b), its ground-truth perspective (c), and the ADS's perception of the scenario (d). The ADS detects the SDV (yellow bounding box), and the ADS's tracker predicts the SDV's future trajectory (bold green line) as in conflict with the Ego's lane. Although the Ego initiates an emergency stop, the rear-end collision is not avoided.

Table 6.2: Simulation parameters for SDV behavior and results

	SDV Config			Observed				
#	Δd_a	Δd_t	Δv_t	Δd_a	Coll.	v_{SDV}	v_{Ego}	maneuver
7	2	-2	-5	2.07	n	-	-	-
8	2	-5	-5	2.05	y	10.89	13.16	emergency stop
9	5	-5	-5	5.49	n	-	-	-
10	5	-5	-10	5.50	n	-	-	stop
11	5	-10	-10	5.60	y	7.60	12.15	-

6.3.5 Summary

This experiment demonstrates how the the SDV model can be used with a real ADS to search for scenarios and parameters where the system may not be able avoid a collision. We found that using another SDV instance as placeholder for Ego enables a rapid iterative development of test scenarios. The iterations are needed to ensure the correct behavior of the cutting-in vehicle and select reasonable ranges of test parameters, before running the more time-consuming simulation with Ego controlled by the ADS. Finally, the experiment results also highlight the importance of being able to plan the SDV maneuver trajectories dynamically and influence their shape via parameters.

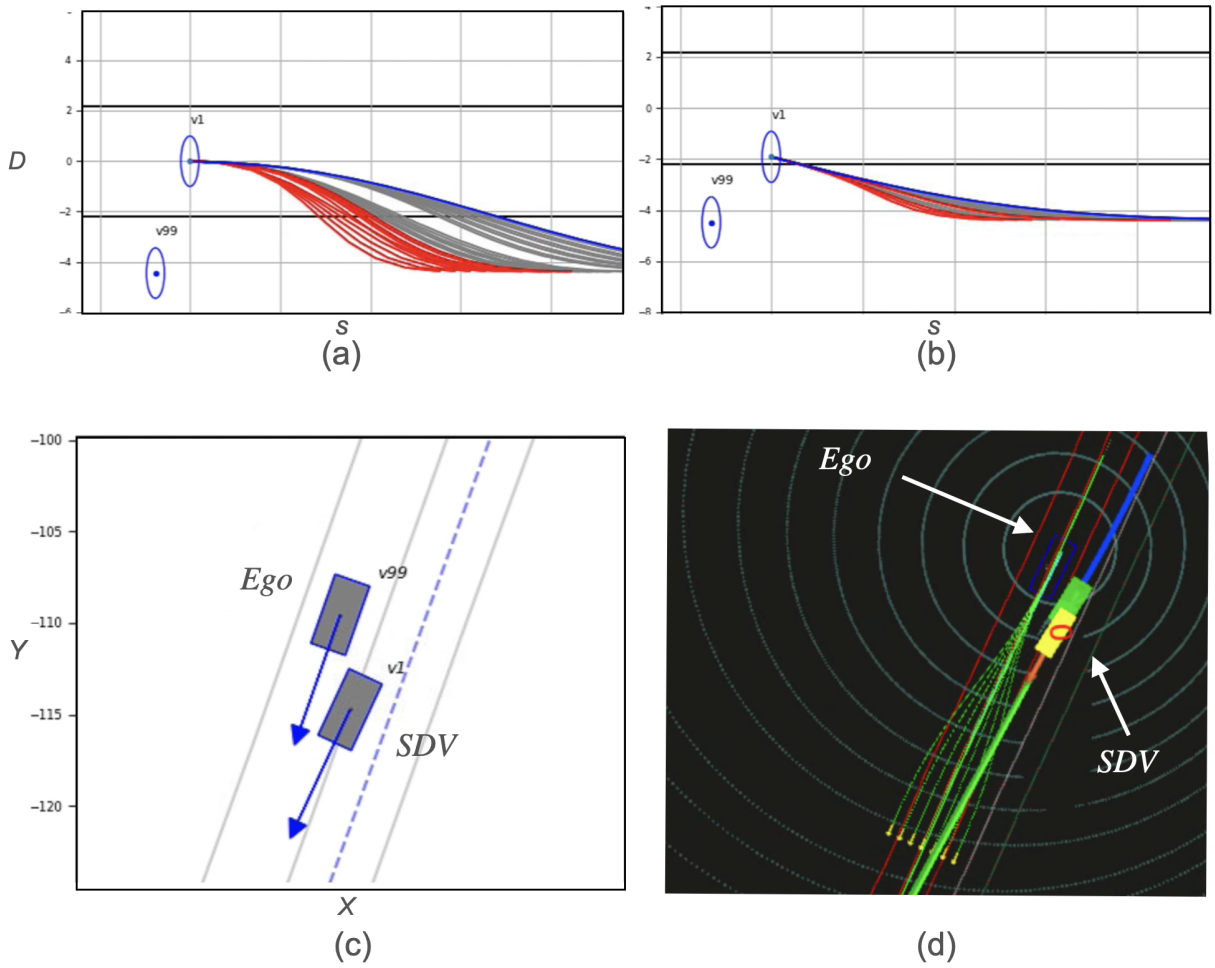


Figure 6.8: One of the simulation scenarios that results in a crash; in (a) and (b), the SDV trajectory generation in Frénet frame targeting Ego at two different moments (optimal trajectory in blue and infeasible ones in red); in (c) the SDV simulation view in Cartesian coordinates; and in (d) the ADS perception (circles represent the lidar simulation, with the Ego located at their center)

6.4 Scalability (RQ4)

We evaluate the SDV model scalability to see if it can support scenarios with heavy traffic. Although most scenarios rely on a small set of vehicles interacting with Ego, such as between one and three in the NHTSA pre-crash scenarios, other scenarios may require simulating heavy traffic. For example, we observed that a single vehicle in our intersection dataset may interact with up to six other vehicles. In order to support such scenarios, the model must be able to scale traffic density without any significant degradation of the simulation performance or the quality of the planned trajectories. We run this experiment with a long scenario (two minutes) and increasing traffic density where all vehicles are running SDV models.

Research Question 4: How does the model performance scale with traffic density?

6.4.1 Reference Implementation and Performance Requirements

The experiment uses the reference implementation (Chapter 5) running in real-time. To provide a sufficient simulation update rate, the Behavior and Maneuver Layers target a planning rate of 3 Hz, and the Execution Layer targets updating the position of all vehicles at 30 Hz.

Planning is a highly time-critical task, which needs to be executed within its target period of 333 ms (3 Hz). If a vehicle misses the target time to generate its plan, it likely affects the quality of its trajectory and the resulting motion. Furthermore, a long overrun can affect the SDV model's ability to predict the traffic state, resulting in sub-optimal trajectories and even unintended collisions.

The Execution Layer executes the trajectory of each vehicle from the previous planning cycle by (i) transforming the current target state in the planned trajectory from the Frénet frame to the Cartesian frame and (ii) updating the vehicle's position, velocity, acceleration, and yaw. The state transformation and update must be completed for all vehicles within 33 ms. A small exceedance, if consistent, may be acceptable, as it would slightly reduce the update frequency below 30 Hz without destroying the actual vehicle motion.

Note: The experiment is executed on an Intel Core i7-6800K at 3.40 GHz (6 cores), with 32 GB RAM and Ubuntu 18.04.5 (the implementation is CPU based).

6.4.2 Scenarios

We use two long-running scenarios, each with a two-minute duration, and vary the number of vehicles, up to 20. In each scenario, the vehicles travel in one lane and form a virtual platoon, simulating heavy traffic (see Figure 6.9). In scenario A, the vehicles travel without any disturbance, and in scenario B, they need to steer to avoid a static obstacle in their lane. When running scenario A, collision checking for obstacles is inactive; and it is activated when running scenario B. The purpose of scenario B is to show the impact of collision checking on scalability, since it is computationally expensive to plan around objects. Each vehicle travelling behind another one is expected to observe a safe following distance in both scenarios.

Figure 6.9: Scenario setup in GeoScenario. Up to 20 vehicles are travelling in one lane on the University of Waterloo Ring Road (lanelet map on the left). On the right the image is zoomed-in over the starting lane. The starting distance between vehicles is defined as 5m in Frénet frame.

6.4.3 Metrics

We evaluate the adherence to the target rates using the following metrics:

- *Target Rate Compliance* (TRC), defined as the % of simulation (execution) ticks from all vehicles that adhere to the target tick time of 33 ms (30 Hz);
- *maximum tick time*, defined as the longest tick in simulation;
- *Target Planning Rate Compliance* (TPRC), defined as the % of planning cycles from all vehicles that adhere to the target time of 333 ms (3 Hz);
- the *maximum planning time*, defined as the longest planning time among all vehicles (worst-case);

The compliance metrics allow us to observe the simulation cycles in which the vehicles can maintain the target rate in both planning and execution, and the maximum tick time and maximum planning time reveal the severity of the worst cases. Additionally, we observe if there is any collision between vehicles or between vehicles and the obstacle during the entire scenario.

6.4.4 Results

Both scenarios with up to 20 vehicles execute successfully, without any collisions or lane boundary violations. The planning adheres to the target rate with almost 100%, with 99.8% being the worst case (Table 6.3). On the other hand, execution deteriorates significantly between 10 and 15 vehicles, especially when the collision checking is active, plunging from 98.49% to 78.58%.

Such a deterioration of the target rate to update the state of all vehicles may introduce inconsistencies and confuse the ADS under test, such as inducing significant errors in its object tracking system. However, reducing the update rate from 30 Hz to 20 Hz results in near perfect adherence for up to 20 vehicles when no collision checking is used and up to 15 vehicles with the collision checking active (Figure 6.10). Thus, scenarios with up to 10 SDV instances are easily handled by the reference implementation, and scaling to 20 instances requires reducing the update rate.

For scenarios requiring even more vehicles, the traffic can consist of a mix of vehicles, with the more expensive SDV instances used for interactions with Ego, and the remaining vehicles following PDTs, which have a negligible computing cost.

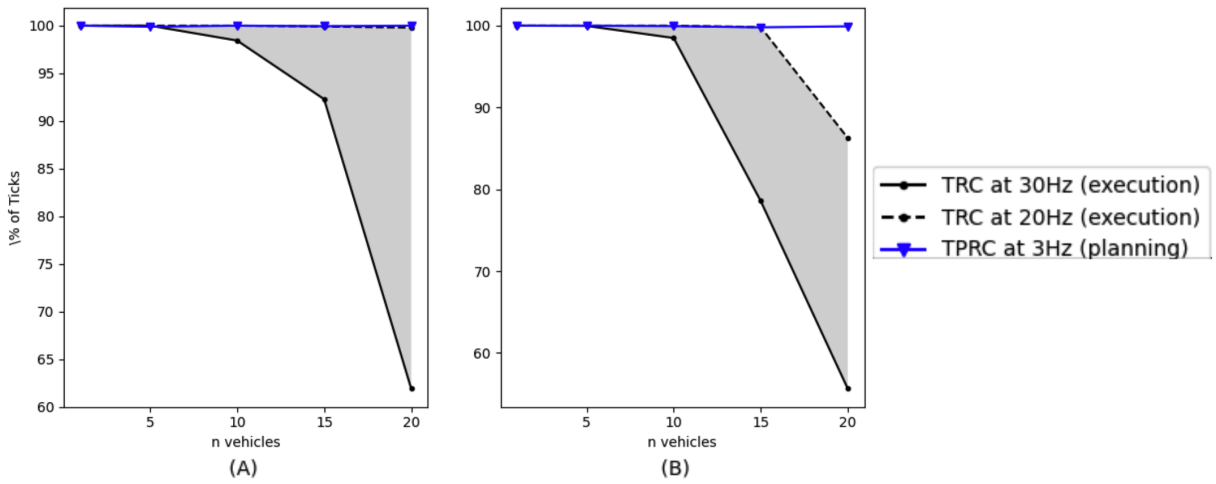


Figure 6.10: Performance with increasing number of vehicles. (A) is without obstacle avoidance, and (B) is with obstacle avoidance. The grey area represents the performance range between 20 Hz to 30 Hz.

Table 6.3: Performance with multiple scenario configurations in real-time simulation

id	vehicles	obstacle	coll.	TRC	max tick	TPRC	max plan
1	1	inactive	0	100.00%	0.033s	100.00%	0.333s
2	5	inactive	0	100.00%	0.033s	99.93%	0.337s
3	10	inactive	0	98.44%	0.042s	100.00%	0.333s
4	15	inactive	0	92.28%	0.055s	99.94%	0.338s
5	20	inactive	0	61.90%	0.052s	100.00%	0.333s
6	1	active	0	100.00%	0.033s	100.00%	0.333s
7	5	active	0	99.97%	0.034s	100.00%	0.333s
8	10	active	0	98.49%	0.041s	99.94%	0.338s
9	15	active	0	78.58%	0.052s	99.80%	0.340s
10	20	active	0	55.65%	0.065s	99.91%	0.343s

When real-time simulation is not required, the tick time in the worst cases do not affect the quality of the simulation and even more vehicles can be added. However, the total time in simulation is still relevant to scale the number of scenarios. We run Scenario B 35% faster with 20 vehicles, and 7% faster with 30 vehicles (see Table 6.4). Additional vehicles in the simulation take longer than real-time to finish, but the growth is linear.

Table 6.4: Performance with faster-than-real-time simulation

id	vehicles	obstacle	time	% from sim time
12	20	active	77.8s	65%
13	30	active	112.5s	93%

6.5 Threats to Validity

A threat to external validity of our conclusions is the selection of scenarios to answer RQ1. We acknowledge that the scenario set does not allow us to conclude that our findings are applicable to other safety-critical scenarios, or other scenarios with Ego-to-HV interactions. However, we attempt to increase external validity by using NHTSA Pre-Crash Typology as a source to compose a realistic and diverse evaluation set. The NHTSA typology is constructed with data aggregated from 6,170,000 police reported crashes involving at least one light vehicle. We believe they should be of first priority to evaluate safety in ADS development. Moreover, after filtering this set to applicable scenarios, the resulting evaluation set (18 scenarios) still account for a substantial percentage of all reported crashes (50%). They cover different types of vehicle-to-vehicle interactions with varying road locations (traffic light intersections, stop sign intersections, single and double lane roads, same and opposite traffic lanes, etc.).

Another threat to external validity is the Motion Realism from RQ2. The data is limited to a single road location, and it does not cover all driving styles from HVs. It does not support us to conclude the SDV performance will be similar when naturalistic data collected from different locations is used. However, it shows the potential of using Behavior Trees to express high-level behavior combined with Maneuver Models as low-level behavior. Further, the performance improvement after calibration shows the configurability of the model and how it can approximate synthetic and empirical vehicles. This calibration process can be improved with learning approaches applied to new data sources and further approximate simulated vehicles to naturalistic driving from new traffic observations.

A threat to internal validity arises when computing the metric *behavior reuse* per scenario in the RQ1. We acknowledge that the order in which the scenarios are created can affect the amount of reuse, since there is an additional pool of behaviors to compose new scenarios. We mitigate this threat by designing the scenarios as a collection to maximize reuse across all scenarios. This way, we are not favouring one scenario over the other. Another threat arises from the different styles used by testers to create a scenario and write the behavior trees. For example, a scenario can be expressed with a high level of abstraction and encode intention while maximizing reuse, but it might be difficult to reach the desired behavior during execution. We mitigate this threat by using three conflicting metrics as objectives, and a good performance requires a balance that avoids design styles that only maximizes one or the other.

6.6 Chapter Conclusion

We evaluate GeoScenario and the SDV model in terms of: scenario development effectiveness (RQ1), which includes expressiveness, execution accuracy, and reuse, using NHTSA pre-crash scenarios; its motion realism (RQ2) in comparison to naturalistic urban traffic; its practical applicability (RQ3) to test an actual ADS, and its scalability with traffic density (RQ4);

The results show that the model is able to successfully express and accurately execute all eighteen NHTSA vehicle-to-vehicle pre-crash scenarios, except one scenario variant. This exception only applies to a variant that requires the U-Turn (a limitation in our trajectory planner), while the main scenario is based on a vehicle leaving the parking area and could successfully be performed. In comparison, only four scenarios are effectively expressible using PDTs, which is our baseline. In particular, lane-change and crossing-path scenarios are highly unpractical with the baseline and benefit the most from the new model. The SDV model also results in high-levels of internal reuse, achieving over 80% on average for the NHTSA scenarios.

We use naturalistic traffic data from a busy urban intersection and show how the motion of the simulated vehicles is similar to that of the real vehicles when operating under the same conditions. In the best performing scenarios (first quartile), the synthetic trajectories from the model are almost indistinguishable from empirical vehicles, with an average spatio-temporal trajectory distance of less than 55 cm (while the average from all scenarios is 1.24 m).

We demonstrate the model's applicability in ADS scenario-based testing with a real subject system, and its ability to reveal collision scenarios that cannot be expressed using

the baseline. Finally, we show that the model scales in scenarios with up to 10-20 simultaneous highly-interactive vehicles, while maintaining simulation quality and consistency in the driving task. Given that Ego rarely interacts with more than 8 vehicles at a time, the achieved scalability should be sufficient for most test scenarios.

Chapter 7

Conclusion

Scenario-based testing is one of the primary methods to verify and validate the behavioral safety of automated vehicles, as mandated by industry standards (e.g., ISO26262 and ISO21448) and safety frameworks (e.g., Waymo's [16]). Supporting such testing requires both a formal representation to express scenarios, and simulation models for execution. Many solutions have emerged in recent years, but they focus on the high-level scenario definition and lack the features to precisely and reliably control the required micro-simulation. This thesis proposes GeoScenario as a tool-independent DSL for scenario representation that covers both structure and behavior with a model for simulated human-operated vehicles (SDV model). Our approach improves scenario modeling and execution for ADS scenario-based testing, allowing researchers and engineers to develop tool-independent test scenarios with sufficient behavior controllability, realistic movements, and interactive planning of the participating road users to achieve the test objectives.

The proposed SDV model uses a combination of BTs and dynamic trajectory planning. The model encapsulates driver and vehicle as a single entity with a layered architecture that provides a user-oriented language to coordinate the vehicle behavior, and vehicle motion planning that optimizes for realism and achieving the scenario test objective. In particular, BTs provide a high-level description of discrete decisions, with a high-level of abstraction and parameterization to support controllability and reuse. Further, dynamic trajectory planning allows for flexible adaptation of the SDV trajectories to different road geometries and achieving the test objective despite varying and unpredictable Ego behaviors.

The evaluation shows that GeoScenario supports effective test scenario development in design and execution. All eighteen NHTSA vehicle-to-vehicle pre-crash scenarios are successfully expressed and executed using the SDV model, except for one variant due to

unsupported U-turns. The scenario analysis also shows that their majority (78%) require dynamic trajectory planning, and thus cannot be effectively handled using the predefined-trajectory agent baseline. The dynamic trajectory planning also allows for easy adaptation of the tests to different road geometries. The ability to reuse sub-trees and override parameters support high levels of internal reuse, achieving over 80% on average for the NHTSA scenarios. In other words, on average, over 80% of a scenario's content is also used in one or more other scenarios.

The evaluation also shows the ability of the SDV model to reproduce real-world vehicle behavior and scale sufficiently. In one of the experiments, the average STED between the simulation and the real trajectories for 100 traversals through a busy urban intersection is 4.27 m before calibration, and it improves to 1.24 m after calibration. In particular, the simulation faithfully reproduces different driving styles by adjusting parameters and can accommodate custom behaviors, including misbehaviors, as additional conditions and maneuvers. The reference implementation demonstrates that the SDV model scales to execute scenarios with 10-20 highly interactive vehicles, and additional optimizations, such as reducing the number of sampled trajectories for vehicles farther away from Ego, allow for further scaling.

The application of GeoScenario to test WISE ADS in the cut-in scenario confirms the usefulness of the model and offers practical insights. Among others, the ability to control the shape of the cut-in trajectories uncovers the varied response of the ADS to different trajectories, showing that not only the target gap and velocity, but also the acceptance gap impact the likelihood of a collision. Further, using an SDV model instance in place of Ego helps accelerate the development of the test scenario and parameter selection to tune the trajectories of the agent that challenges Ego.

Additionally, this thesis contributes with an open catalog of executable test scenarios, designed with GeoScenario and SDV model (from the Evaluation and other GeoScenario applications), and a complete toolset to support scenario-based testing that includes the language parser, SDV Model reference implementation, and integration tools. This toolset can be integrated in co-simulation with any existing simulation environment and any autonomy stack. With an open format and open-source tools, we hope it encourages the adoption by the research community. The toolset and evolving documentation are available online.[1]

[1] https://geoscenario.readthedocs.io

7.1 Limitations and Future Work

GeoScenario does not cover environmental conditions (e.g., precipitation, fog, snow) and the effects of such phenomena on sensor performance and vehicle dynamics. We plan to address them in both scenario design and their effects on the SDV model for a comprehensive traffic simulation.

The SDV model addresses the dynamic behavior of human operated light vehicles, while other vehicle types are not covered (e.g., motorbikes, trucks and other heavy vehicles, multi-body vehicles). While the concept of Behavior Tree in scenario design can be applied to any simulated "driver", the vehicle motion must be designed for the mechanical constraints of such vehicles. We plan to extend the Maneuver Layer to account for the differences in vehicle types. Similarly, we plan to extend the model for pedestrian behavior. Preliminary results with BTs and the Social Force Model are available in [50]. Our next goal is to integrate both models for scenarios with interactions between vehicles and pedestrians [50].

The model cannot reproduce irregular maneuvers that do not follow the structure of the road or lane connectivity. For example, U-turns, 3-point-turns, and driving through a parking lot are not supported. We plan to address this limitation with a new method to generate the reference path and Frénet frame in such conditions, allowing the vehicle to alternate driving modes.

We plan several other model extensions and new capabilities that exploit the model. We plan to expand the model with new maneuvers and configuration options based on additional scenarios, harvested from a wider range of naturalistic data, such as the additional locations in the Waterloo dataset [14] and the multi-country INTERACTION dataset [80]. To improve the maneuver parameter sampling method, we plan to explore probabilistic distribution modelling to represent joint distributions in the maneuver configuration space. This can accelerate the planning task by reducing the number of unfeasible trajectories in the candidate set. Finally, we plan to improve the auto-calibration process and further automate creation of BTs and their parameterization to approximate the naturalistic traffic. We plan to exploit the model in generating new scenarios by injecting road-user misbehaviour into BTs, such as simulating distraction [78] and ignoring occlusions [41].

References

[1] ANTLR - ANother Tool for Language Recognition. .

[2] Government Technology. https://www.govtech.com/fs/transportation/poll-nearly-half-of-us-drivers-skeptical-of-autonomous-cars.html.

[3] Java Open Street Map Editor. https://josm.openstreetmap.de.

[4] MathWorks PreScan. .

[5] Measurable Scenario Description Language (M-SDL). https://www.foretellix.com/open-language/.

[6] Open Street Map (OSM). https://www.openstreetmap.org.

[7] OpenDRIVE. https://www.opendrive.com.

[8] OpenScenario. https://www.asam.net/standards/detail/openscenario.

[9] Robot Operating System (ROS). https://www.ros.org/.

[10] Simulation of Urban MObility. https://www.eclipse.org/sumo/.

[11] State of California, Department of Motor Vehicles. https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/.

[12] SVL SIMULATOR. https://www.svlsimulator.com.

[13] Virtual Test Drive (VTD). https://vires.com/vtd-vires-virtual-test-drive.

[14] Waterloo Multi-Agent Traffic Dataset. http://wiselab.uwaterloo.ca/waterloo-multi-agent-traffic-dataset.

[15] WISE ADS. https://uwaterloo.ca/waterloo-intelligent-systems-engineering-lab/projects/wise-automated-driving-system.

[16] Waymo safety report. Technical report, 09 2020.

[17] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. Testing advanced driver assistance systems using multi-objective search and neural networks. In *31st IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*, pages 63–74, Sep 2016.

[18] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *Proc. 40th Int. Conf. on Software Engineering*, pages 1016–1026. ACM, 2018.

[19] M. Althoff, M. Koschi, and S. Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 719–726, June 2017.

[20] Michal Antkiewicz, Maximilian Kahn, Michael Ala, Krzysztof Czarnecki, Paul Wells, Atul Acharya, and Sven Beiker. Modes of automated driving system scenario testing: Experience report and recommendations. In *SAE World Congress Experience*. SAE, 2020.

[21] J. Andrew Bagnell, Felipe Cavalcanti, Lei Cui, Thomas Galluzzo, Martial Hebert, Moslem Kazemi, Matthew Klingensmith, Jacqueline Libby, Tian Yu Liu, Nancy Pollard, Mihail Pivtoraiko, Jean-Sebastien Valois, and Ranqi Zhu. An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962, 2012.

[22] P. Bender, J. Ziegler, and C. Stiller. Lanelets: Efficient map representation for autonomous driving. In *IEEE Intelligent Vehicles Symposium*, pages 420–425, June 2014.

[23] Ewin R Boer, Marika Hoedemaeker, et al. Modeling driver behavior with different degrees of automation: A hierarchical decision framework of interacting mental models. In *Proc. 17th European annual conference on human decision making and manual control*, pages 63–72, 1998.

[24] Oliver Bühler and Joachim Wegener. Automatic testing of an autonomous parking system using evolutionary computation. *SAE Technical Papers*, 2004.

[25] Qianwen Chao, Huikun Bi, W. Li, Tianlu Mao, Zhaoqi Wang, Ming C. Lin, and Z. Deng. A survey on visual traffic simulation: Models, evaluations, and applications in autonomous driving. *Computer Graphics Forum*, 39, 2020.

[26] Chao, Edward. Autonomous driving: Mapping and behavior planning for crosswalks. Master's thesis, 2019.

[27] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[28] Krzysztof Czarnecki. Automated Driving System (ADS) Task Analysis - Part 1: Basic Motion Control Tasks. Technical report, 07 2018.

[29] Krzysztof Czarnecki. Automated Driving System (ADS) Task Analysis - Part 2: Structured Road Maneuvers. Technical report, 07 2018.

[30] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proc. 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[31] Martin Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[32] William Frakes and Carol Terry. Software reuse: Metrics and models. *ACM Comput. Surv.*, 28:415–435, 06 1996.

[33] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and scene generation. In *Proc. 40th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, page 63–78, New York, USA, 2019. ACM.

[34] S. Geyer, M. Baltzer, B. Franz, S. Hakuli, M. Kauer, M. Kienle, S. Meier, T. Weissgerber, K. Bengler, R. Bruder, F. Flemisch, and H. Winner. Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance. *IET Intelligent Transport Systems*, 8(3):183–189, 2014.

[35] P.G. Gipps. A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological*, 15(2):105–111, 1981.

[36] Kentaro Go and John M. Carroll. The blind men and the elephant: Views of scenario-based system design. *Interactions*, 11(6):44–53, November 2004.

[37] Ilievski, Marko. Wisebench: A motion planning benchmarking framework for autonomous vehicles. Master's thesis, 2020.

[38] ISO/FDIS 26262:1994. *Road vehicles – Functional safety.* ISO, Geneva, Switzerland, 2011.

[39] ISO/PAS-21448:2019. *Road vehicles – Safety of the intended functionality.* ISO, Geneva, Switzerland, 2019.

[40] Matthias Jarke, X. Tung Bui, and John M. Carroll. Scenario management: An interdisciplinary approach. *Requirements Engineering*, 3(3):155–173, Mar 1998.

[41] Maximilian Kahn, Atrisha Sarkar, and Krzysztof Czarnecki. I know you can't see me: Dynamic occlusion-aware safety validation of strategic planners for autonomous vehicles using hypergames. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[42] Nidhi Kalra and Susan M. Paddock. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, Santa Monica, CA, 2016.

[43] Cem Kaner, James Bach, and Bret Pettichord. *Lessons Learned in Software Testing.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[44] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *Int. J. Robot. Res*, pages 583–602, 2003.

[45] Arne Kesting, Martin Treiber, and Dirk Helbing. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record*, 1999(1):86–94, 2007.

[46] Richard Knoblauch, Martin Pietrucha, and Marsha Nitzburg. Field studies of pedestrian walking speed and start-up time. *Transportation Research Record: Journal of the Transportation Research Board*, 1538:27–38, 1996.

[47] Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE Int. J. Trans. Safety*, 4:15–24, 04 2016.

[48] Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125. IEEE, 2018.

[49] Robert Krajewski, Tobias Moers, Dominik Nerger, and Lutz Eckstein. Data-driven maneuver modeling using generative adversarial networks and variational autoencoders for safety validation of highly automated vehicles. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2383–2390. IEEE, 2018.

[50] Scott Larter, Rodrigo Queiroz, Sean Sedwards, Atrisha Sarkar, and Krzysztof Czarnecki. A hierarchical pedestrian behaviour model to generate realistic human behaviour in traffic simulation. In *IEEE Intelligent Vehicles Symposium (IV22)*. IEEE, 2022.

[51] Till Menzel, Gerrit Bagschik, and Markus Maurer. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, Suzhou, China, June 26-30, 2018*, pages 1821–1827, 2018.

[52] John A Michon. A critical view of driver behavior models: what do we know, what should we do? In *Human behavior and traffic safety*, pages 485–524. Springer, 1985.

[53] Sara Moridpour, Majid sarvi, and Geoff Rose. Modeling the lane changing execution of multi class vehicles under heavy traffic conditions. *Transportation Research Record: Journal of the Transportation Research Board*, 2161, 12 2010.

[54] W. G. Najm, John D. Smith, and Mikio Yanagisawa. Pre-Crash Scenario Topology for Crash Avoidance Research. Technical report, U.S. Department of Transportation, NHTSA, April 2007.

[55] W. G. Najm, S. Toma, and J. Brewer. Depiction of Priority Light-Vehicle Pre-Crash Scenarios for Safety Applications Based on Vehicle-to-Vehicle Communications. Technical report, U.S. Department of Transportation, NHTSA, April 2013.

[56] Mirco Nanni and Dino Pedreschi. Time-focused clustering of trajectories of moving objects. *J. Intell. Inf. Syst.*, 27:267–289, 11 2006.

[57] Fabian Poggenhans, Jan-Hendrik Pauls, Johannes Janosovits, Stefan Orf, Maximilian Naumann, Florian Kuhnt, and Matthias Mayr. Lanelet2: A high-definition map framework for the future of automated driving. In *Proc. IEEE Intell. Trans. Syst. Conf.*, Hawaii, USA, November 2018.

[58] Vincenzo Punzo, Maria Teresa Borzacchiello, and Biagio Ciuffo. On the assessment of vehicle trajectory data accuracy and application to the next generation simulation

(NGSIM) program data. *Transportation Research Part C: Emerging Technologies*, 19(6):1243 – 1262, 2011.

[59] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. GeoScenario: An open DSL for autonomous driving scenario representation. In *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[60] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[61] SAE. Operational definitions of driving performance measures and statistics (sae j2944). Technical report, SAE International, 2015.

[62] SAE. Taxonomy and Definitions for Terms Related to Automated Driving System Behaviors and Maneuvers for On-Road Motor Vehicles (SAE J3164). Technical report, SAE International, 2018.

[63] SAE. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles(SAE J3016). Technical report, SAE International, 2021.

[64] SAE. Summary Report: Standing General Order on Crash Reporting for Automated Driving Systems. Technical report, NHTSA, 2022.

[65] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. An analysis of iso 26262: Machine learning and safety in automotive software. *SAE Technical Papers*, 2018.

[66] Barbara Schütt, Thilo Braun, Stefan Otten, and Eric Sax. Sceml: A graphical modeling framework for scenario-based testing of autonomous vehicles. In *Proc. 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, page 114–120, New York, NY, USA, 2020. ACM.

[67] Chris Schwarz. On computing time-to-collision for automation scenarios. *Transportation Research Part F: Traffic Psychology and Behaviour*, 27:283 – 294, 2014.

[68] J. Sewall, David Wilkie, Paul C. Merrell, and Ming C. Lin. Continuum traffic simulation. *Computer Graphics Forum*, 29, 2010.

[69] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017.

[70] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars, 2018.

[71] Ravi Shanker, Adam Jonas, Scott Devitt, Katy Huberty, Simon Flannery, William Greene, Benjamin Swinburne, Gregory Locraft, Adam Wood, Keith Weiss, Joseph Moore, Andrew Schenker, Paresh Jain, Yejay Ying, Shinji Kakiuchi, Ryosuke Hoshino, and Andrew Humphrey. Autonomous Cars Self-Driving the New Auto Indåustry Paradigm (Blue Paper). Technical report, Morgan Stanley, 2013.

[72] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors, 2021.

[73] A. Takahashi, T. Hongo, Y. Ninomiya, and G. Sugimoto. Local path planning and motion control for agv in positioning. In *Proc. IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, pages 392–397, 1989.

[74] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, Sept 2015.

[75] UMTRI. Safety Pilot Model Deployment. Technical report, The University of Michigan Transportation Research Institute (UMTRI), 2017.

[76] Richard van der horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. 01 1994.

[77] Van Gennip, Matthew. Vehicle dynamic modelling and parameter identification for an autonomous vehicle, 2018.

[78] J.W.C. van Lint and S.C. Calvert. A generic multi-level framework for microscopic traffic simulation—theory and an example case in modelling driver distraction. *Transportation Research Part B: Methodological*, 117:63–86, 2018.

[79] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a Frenét Frame. *IEEE Int. Conference on Robotics and Automation*, pages 987–993, 2010.

[80] Wei Zhan, Liting Sun, Di Wang, Haojie Shi, Aubrey Clausse, Maximilian Naumann, Julius Kümmerle, Hendrik Königshof, Christoph Stiller, Arnaud de La Fortelle, and Masayoshi Tomizuka. INTERACTION Dataset: An INTERnational, Adversarial and

Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps. *arXiv:1910.03088 [cs, eess]*, 2019.

[81] Xizhe Zhang, Siddartha Khastgir, and Paul Jennings. Scenario description language for automated driving systems: A two level abstraction approach. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 973–980, 2020.

[82] D. Zhao, H. Lam, H. Peng, S. Bao, D. J. LeBlanc, K. Nobukawa, and C. S. Pan. Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques. *IEEE Transactions on Intelligent Transportation Systems*, 18(3):595–607, March 2017.

Appendix A

Behavior Tree Conditions

The condition node set that evaluates the estimated Traffic State. They are building blocks to create and modify the vehicle behavior using Behavior Trees. This set is available in the reference implementation, but it can be expanded by the user with access to he Traffic State. For some conditions, the target vehicle can be identified by *vid* (vehicle with id), *lid* (vehicle in lane with id), or *zid* (vehicle in zone with id). If no vehicle is given, assumes $self$ (reference to vehicle running this tree). For example, *vehicle_stopped* means if self is not moving, *vehicle_stopped*($lid = RIGHT$) means if vehicle on the right lane is not moving. The units for distance, time, and velocities are respectively seconds, meters, and meters per second.

Basics

- **sim_time**
  t, min, max
  Success if simulation time matches given time t or is within min and max values (in seconds). Using exact time will have a tolerance of up to the planner rate (3ms if 3Hz). If max is not given, is assumed to be $infinity$.

- **vehicle_state**
  vid, lid, zid
  Success if vehicle state has reached longitudinal and lateral values at minimum. If no *vid* or *zid* is assigned, defaults to $self$.

- **delta_vehicle_state**
  $s\_pos, s\_vel, s\_acc, d\_pos, d\_vel, d\_acc, vid, lid, zid$

Success if vehicle state has reached delta values at minimum. Comparison is between $self$ and the given vehicle.

- **wait**
  t
  Success if time since node is first visited is $\geqslant t$. Resets clock after returning success. On other words, wait given time before moving to next node (used to delay reactions).

- **interrupt**
  Returns Failure. Can be used to interrupt a sequence of conditions with success.

Condition *sim_time* can be useful to coordinate certain actions at the beginning of the scenario. However, as the timeline advances, using simulation time to control the flow is prone to error and scenario drift (when scenario deviates from intended design). We recommend using Metrics and Interaction Conditions as the main form of control.

The conditions *vehicle_state* and *delta_vehicle_state* are low level conditions that can be used when Traffic Interactions and Metric conditions are not adequate. We recommend conditions with higher level of abstraction whenever possible.

Routing and Driving Mission

- **reached_goal**
  distance
  Success if the vehicle has reached or passed the goal point (see Routing in Section 4.5.1)

- **at_lane_change_segment**
  Success if the vehicle is inside a road segment where a lane change is required to continue on route.

- **target_lane**
  lid
  Success if given lane is the target lane. Target lane is the default lane if a lane change is requested without specifying direction.

- **out_of_route**
  Success if vehicle is outside planned route.

Traffic Interactions

- **vehicle_stopped**
  vid, lid, zid
  Success if vehicle is not moving.

- **vehicle_moving**
  vid, lid, zid
  Success if vehicle is moving.

- **vehicle_yielding**
  $vid, lid, zid, distance, vel$
  Success if vehicle is stopped and at yielding position (stop line or right before conflicting lanelet). *vel* and *distance* (from the stop line) are thresholds to define the yielding state. A small velocity threshold can used to account for noise in the estimation.

- **vehicle_parked**
  $vid, lid, zid, distance, vel$
  Success if vehicle has stopped and at parking position. *vel* and *distance* (offset from the center of the lane) are thresholds to define the parked state. A small velocity threshold can be used to account for noise in the estimation.

- **lv_stopped**
  Same as vehicle_stopped, but targets the lead vehicle.

- **lv_moving**
  Same as vehicle_moving, but targets the lead vehicle.

- **lv_parked**
  Same as vehicle_parked, but targets the lead vehicle.

- **can_lane_change**
  $lid, gap, time\_gap$
  Success if there a lane change to *lid* or $target\_lane$ (set by a previous Action) can be performed. *gap* and $time\_gap$ can be used to configure the acceptance conditions.

- **is_ego**
  vid, lid, zid
  Success if vehicle is Ego.

Metrics

- **distance**
  vid, lid, zid, min, max
  Success if distance to given vehicle (in Cartesian) is between min and max (inclusive).

- **time_gap**
  vid, lid, zid, min, max
  Success if time distance to given vehicle (in Frénet, if vehicle is on the path) is between min and max (inclusive) in seconds.

- **gap**
  vid, lid, zid, min, max
  Success if distance to given vehicle (in Frénet, if vehicle is on the path) is between min and max (inclusive) in meters. Note gap is negative if vehicle is behind.

- **longitudinal_distance**
  vid, lid, zid, min, max
  Success if absolute longitudinal distance to given vehicle (in Frénet) is between min and max (inclusive). Note this is the absolute difference, and is always positive (unlike gap).

- **lateral_distance**
  vid, lid, zid, min, max
  Success if absolute lateral distance to given vehicle (in Frénet) is between min and max (inclusive). Note this is the absolute difference, and is always positive (unlike gap).

Road and Regulatory Elements

- **approaching_intersection**
  distance
  Success if vehicle is approaching a regulated intersection. Threshold is given in distance to intersection.

- **approaching_stop_sign**
  distance
  Success if vehicle is approaching any stop sign (useful when map is incomplete and does not contain a proper regulatory element). Threshold is given in distance to intersection.

- **intersection_type**
  type
  Success if vehicle is approaching intersection from given type. Type can be RIGHT_OF_WAY, ALL_WAY_STOP, TRAFFIC_LIGHT, or PEDESTRIAN_CROSS.

- **yield_role**
  Success if vehicle approaching intersection and has the yielding role.

- **intersection_occupied**
  vid
  Success if intersection of any type (RightOfWay, TrafficLight or AllWayStop) has crossing vehicles (moving). With *vid*, returns success only if given vehicle is present.

- **row_occupied**
  vid, *distance*
  Success if RightOfWay intersection has approaching vehicles in the lanelets with right of way, limited to *distance* from the intersection. With *vid*, returns success only if given vehicle is present.

- **aws_occupied**
  vid, *distance*
  Success if AllWayStop intersection has vehicles in yielding lanelets, limited to *distance* from the intersection. With *vid*, returns success only if given vehicle is present.

- **aws_yield**
  vel, *wait_time*, *risk_probability*
  Success if AllWayStop intersection has vehicles yielding with priority. Failure otherwise (meaning self has priority). With *vid* it returns success only if given vehicle is present, *vel* is the velocity threshold to assume a vehicle is moving (account for noise), *wait_time* is the max wait time on a deadlock before moving, and *risk_probability* is the probability of taking the risk and moving first from 0 (never move) to 1 (always move), after wait_time in deadlock has passed.

- **lane_occupied**
  time, *distance*
  Success if current lane is occupied (if there is a vehicle ahead). Limited to maximum time and distance as thresholds.

- **traffic_light_state**
  color

Success if state of the traffic light (applicable to current lanelet) matches the given color state. Example traffic_light_state(color='RED').

Actions

Action conditions modify the state of the vehicle and return *Success*.

- **action_set_target_lane**
  lid
  change target lane.
- **action_reroute**
  recalculate a route and generate a new reference path.
- **action_turn_signal_left**
  state
  Change left turn signal state. Will toggle on/off if no state is given.
- **action_turn_signal_right**
  state
  Change right turn signal state. Will toggle on/off if no state is given.
- **action_head_light**
  state
  Change headlight state. Will toggle on/off if no state is given.

Appendix B

Behavior Tree Grammar

SDV Behavior Tree Grammar in ANTLR4 format [1].

Listing B.1: Behavior Tree Grammar

```
grammar BTreeDSL;
/* Parser Rules */
behaviorTree    : ('behaviortree' name ':' INDENT rootNode NL? DEDENT?)+EOF;

rootNode        : node;
node            : leafNode | nodeComposition;
nodeComposition : OPERATOR name? INDENT node+ DEDENT;
leafNode        : (maneuver | condition | subtree) NL;

condition       : 'condition' name '(' cconfig ')';
maneuver        : 'maneuver' name '(' mconfig ')';
subtree         : 'subtree' name '(' (midconf (',' midconf)*)? ')';
midconf         : mid '=' mconfig;
mconfig         : name '(' params* ')';
cconfig         : name '(' params* ')';
mid             : name;
params          : bexpr (',' bexpr)* ;
bexpr           : name (BOP|ATT) value ;
value           : FLOAT | name | func | tupl;
func            : name '('FLOAT (',' FLOAT)*')';
tupl            : '('FLOAT (',' FLOAT)*')';
name            : WS* WORD WS*;
```

```
/* Lexer Rules */
OPERATOR        : '?' | '->' | '||';
BOP             : '<'|'>'|'=='|'>='|'<='|'!=';
ATT             : '=';
FLOAT           : [+-]?([0-9]*[.])?[0-9]+;
WORD            : ([a-z] | [A-Z] | '_')+ ;
WS              : (' ' | '\t') -> skip;
```

Appendix C

NHTSA Scenarios

The National Highway Traffic Safety Administration (NHTSA), in conjunction with the Research and Innovative Technology Administrarion's Volpe Nation Transportation System Center (Volpe Center) conduct a series of vehicle safety research in crash avoidance. They published an analysis of the 2004 General Estimates System (GES) crash database where they describe a typology of pre-crash scenarios involving light vehicles [54] with aggregated data. They show statistics of the frequency of occurrence, severity, and number of vehicles involved for all light-vehicle police reported crashes on 2004 GES database. The database contains a total of 6,170,000 crashes, with 5,942,000 reports involving at least one light vehicle, and a total of 10,695,000 light vehicles and 15,027,00 people involved.

Since our goal is to explore interactions between the ADS and human-operated vehicles simulated by the SDV model, we focus on the statistics from Two-Vehicle Pre-crash Scenarios (they also account for the majority of crashes). This thesis also do not cover scenarios involving one vehicle (e.g., vehicle failure, control loss) or animals. We understand how these scenarios are still important for ADS testing, but they do not benefit from our model and are not used during the Evaluation in Section 6.1. Table C.1 shows the 37 pre-crash scenarios that represent 99.3% of all two-vehicle crashes involving at least one light vehicle. As we can see, the most frequent scenarios are #26 Lead Vehicle Stopped (16.41%), #31 Vehicle(s) Turning at Non-Signalized Junctions, and #18 Vehicle(s) Changing Lanes – Same Direction (5.69%). Although the most frequent crash type, #26 was excluded from the testing set in Section 6.1 because it does not require any dynamic behavior from the interacting vehicle. However, we can still use the SDV model to (i) simulate Ego as a placeholder in order to evaluate the scenario before executing simulations with the autonomy stack, and (ii) simulate additional vehicles in traffic to increase the complexity and difficulty of the test.

Table C.1: Pre-Crash Scenario Typology from NHTSA with Relative Frequency [54] and selected scenarios with x.

#	Sel.	Group	Scenario	Frequency
1		Run-Off-Road	Vehicle Failure	0.71%
2		Run-Off-Road	Control Loss With Prior Vehicle Action	1.73%
3		Run-Off-Road	Control Loss Without Prior Vehicle Action	8.90%
4	x	Crossing Paths	Running Red Light	4.27%
5	x	Crossing Paths	Running Stop Sign	0.81%
6		Run-Off-Road	Road Edge Departure With Prior Vehicle Maneuver	1.14%
7		Run-Off-Road	Road Edge Departure Without Prior Vehicle Maneuver	5.62%
8		Run-Off-Road	Road Edge Departure While Backing Up	1.11%
9		Animal	Animal Crash With Prior Vehicle Maneuver	0.39%
10		Animal	Animal Crash Without Prior Vehicle Maneuver	5.13%
11		Pedestrian	Pedestrian Crash With Prior Vehicle Maneuver	0.29%
12		Pedestrian	Pedestrian Crash Without Prior Vehicle Maneuver	0.66%
13		Pedalcyclist	Pedalcyclist Crash With Prior Vehicle Maneuver	0.31%
14		Pedalcyclist	Pedalcyclist Crash Without Prior Vehicle Maneuver	0.41%
15	x	Backing	Backing Up Into Another Vehicle	2.20%
16	x	Lane-change	Vehicle(s) Turning – Same Direction	3.73%
17	x	Lane-change	Vehicle(s) Parking – Same Direction	0.81%
18	x	Lane-change	Vehicle(s) Changing Lanes – Same Direction	5.69%
19	x	Lane-change	Vehicle(s) Drifting – Same Direction	1.65%
20	x	Opposite Direction	Vehicle(s) Making a Maneuver – Opposite Direction	0.26%
21	x	Opposite Direction	Vehicle(s) Not Making a Maneuver – Opposite Direction	2.08%
22	x	Rear-End	Following Vehicle Making a Maneuver	1.44%
23	x	Rear-End	Lead Vehicle Accelerating	0.32%
24	x	Rear-End	Lead Vehicle Moving at Lower Constant Speed	3.53%
25	x	Rear-End	Lead Vehicle Decelerating	7.20%
26		Rear-End	Lead Vehicle Stopped	16.41%
27	x	Crossing Paths	LTAP/OD at Signalized Junctions	3.71%
28	x	Crossing Paths	Vehicle Turning Right at Signalized Junctions	0.59%
29	x	Crossing Paths	LTAP/OD at Non-Signalized Junctions	3.19%
30	x	Crossing Paths	Straight Crossing Paths at Non-Signalized Junctions	4.44%
31	x	Crossing Paths	Vehicle(s) Turning at Non-Signalized Junctions	7.32%
32		Run-Off-Road	Evasive Action With Prior Vehicle Maneuver	0.22%
33		Run-Off-Road	Evasive Action Without Prior Vehicle Maneuver	0.95%
34		Other	Non-Collision Incident	0.77%
35		Object	Object Crash With Prior Vehicle Maneuver	0.51%
36		Object	Object Crash Without Prior Vehicle Maneuver	0.92%
37		Other	Other	0.60%