

Trajectory planning based on adaptive model predictive control: Study of the performance of an autonomous vehicle in critical highway scenarios

Catarina Isabel Vaz Gonçalves

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores
(2^o ciclo de estudos)

Orientador: Prof. Doutor Nuno Gonçalo Coelho Costa Pombo
Coorientador: Prof^a. Doutora Maria do Rosário Alves Calado

Covilhã, janeiro de 2022

Acknowledgements

At the end of this master's thesis, I would like to express my sincere acknowledgements to all those who directly or indirectly contributed to make it happen.

I would first like to thank my supervisor, Prof. Doutor Nuno Gonçalo Coelho Costa Pombo, for his availability, suggestions, criticisms, support and encouragement and for the valuable guidance during the realization of this work. I want to leave a special thanks for all the professionalism and for the sharing of knowledge and constructive ideas that made the realization of this project a more enriching and educational journey.

I would like to express my gratitude to my co-supervisor, Prof^a. Doutora Maria do Rosário Alves Calado, for all her availability, guidance, support and expertise, especially in the practical section. I would like to acknowledge all of the valuable contributions provided to me throughout this project.

In addition, I would like to thank my parents and brother for their wise counsel and unconditional support, and for showing me that hard work pays off.

Finally, I could not have completed this dissertation without the support of my friends, who provided stimulating discussions as well as happy distractions outside of my research.

Abstract

Increasing automation in automotive industry is an important contribution to overcome many of the major societal challenges. However, testing and validating a highly autonomous vehicle is one of the biggest obstacles to the deployment of such vehicles, since they rely on data-driven and real-time sensors, actuators, complex algorithms, machine learning systems, and powerful processors to execute software, and they must be proven to be reliable and safe.

For this reason, the verification, validation and testing (VVT) of autonomous vehicles is gaining interest and attention among the scientific community and there has been a number of significant efforts in this field. VVT helps developers and testers to determine any hidden faults, increasing systems confidence in safety, security, functional analysis, and in the ability to integrate autonomous prototypes into existing road networks. Other stakeholders like higher-management, public authorities and the public are also crucial to complete the VTT process.

As autonomous vehicles require hundreds of millions of kilometers of testing driven on public roads before vehicle certification, simulations are playing a key role as they allow the simulation tools to virtually test millions of real-life scenarios, increasing safety and reducing costs, time and the need for physical road tests.

In this study, a literature review is conducted to classify approaches for the VVT and an existing simulation tool is used to implement an autonomous driving system. The system will be characterized from the point of view of its performance in some critical highway scenarios.

Keywords

Autonomous vehicles, safety-critical, verification, validation, testing, simulation.

Resumo

O aumento da automação na indústria automotiva é uma importante contribuição para superar muitos dos principais desafios da sociedade. No entanto, testar e validar um veículo altamente autónomo é um dos maiores obstáculos para a implantação de tais veículos, uma vez que eles contam com sensores, atuadores, algoritmos complexos, sistemas de aprendizagem de máquina e processadores potentes para executar softwares em tempo real, e devem ser comprovadamente confiáveis e seguros.

Por esta razão, a verificação, validação e teste (VVT) de veículos autónomos está a ganhar interesse e atenção entre a comunidade científica e tem havido uma série de esforços significativos neste campo. A VVT ajuda os desenvolvedores e testadores a determinar quaisquer falhas ocultas, aumentando a confiança dos sistemas na segurança, proteção, análise funcional e na capacidade de integrar protótipos autónomos em redes rodoviárias existentes. Outras partes interessadas, como a alta administração, autoridades públicas e o público também são cruciais para concluir o processo de VVT.

Como os veículos autónomos exigem centenas de milhões de quilómetros de testes conduzidos em vias públicas antes da certificação do veículo, as simulações estão a desempenhar cada vez mais um papel fundamental, pois permitem que as ferramentas de simulação testem virtualmente milhões de cenários da vida real, aumentando a segurança e reduzindo custos, tempo e necessidade de testes físicos em estrada.

Neste estudo, é realizada uma revisão da literatura para classificar abordagens para a VVT e uma ferramenta de simulação existente é usada para implementar um sistema de direção autónoma. O sistema é caracterizado do ponto de vista do seu desempenho em alguns cenários críticos de autoestrada.

Palavras-chave

Veículos autónomos, *safety-critical*, verificação, validação, teste, simulação.

Contents

Chapter 1: Introduction	1
1.1 Objective	3
1.2 Structure.....	3
Chapter 2: Connected and autonomous vehicles	4
2.1 Autonomous vehicles	4
2.1.1 Standards	5
2.1.2 Evolution from ADAS to AD.....	15
2.1.3 Artificial intelligence in automotive software	17
2.1.4 Architecture	21
2.2 Connected vehicles	38
2.2.1 Communication networks	38
2.2.2 Types of V2X connectivity.....	44
Chapter 3: Verification, validation and testing of connected and autonomous vehicles	47
3.1 Software development models	48
3.1.1 V-model vs agile model.....	48
3.1.2 V-model	49
3.2 Verification, validation and testing approaches	52
3.2.1 Formal-based techniques	53
3.2.2 Model-based techniques	54
3.2.3 Ontology-based techniques.....	57
3.2.4 Scenario-based techniques.....	57
3.2.5 Search-based techniques.....	58
3.3 Verification, validation and testing tools.....	58
3.3.1 Verification tools.....	59
3.3.2 Test and validation tools	60
3.4 Challenges of SAE level 3+	62
3.4.1 Sensors.....	62
3.4.2 Communication.....	63
3.4.3 Machine learning	63
3.4.4 Testing.....	63
3.4.5 Regulation	64
Chapter 4: Trajectory planning based on adaptive model predictive control: Study of the performance of an autonomous vehicle in critical highway scenarios.....	65
4.1 Coordinate systems	67

4.2 Vehicle dynamics	68
4.3 Scenario setup.....	74
4.4 Vehicle path planning	79
4.4.1 States conversion.....	79
4.4.2 Polynomials trajectories.....	81
4.4.3 Formulation of local trajectory planning	82
4.5 Vehicle control	101
4.5.1 Adaptive model predictive control	101
4.5.2 Path following control	102
Chapter 5: Results and discussion	107
Chapter 6: Conclusions and future works	131
References.....	133

List of Figures

Figure 2.1 - Cyber-physical system of an autonomous vehicle (adapted from [7]).	4
Figure 2.2- The SAE J3016 levels of driving automation chart, issued in 2014 [9].	6
Figure 2.3 - The SAE J3016 levels of driving automation chart, updated in 2018 ⁴ .	7
Figure 2.4 - IEC 61508 and related standards (adapted from [11]).	8
Figure 2.5 - The ISO 26262:2011 structure [14].	9
Figure 2.6 - The ISO 26262:2018 structure.	9
Figure 2.7 - Determination of an ASIL.	10
Figure 2.8 - The ISO/SAE 21434 standard structure [20].	12
Figure 2.9 - AUTOSAR reference structure [14].	13
Figure 2.10 - Automotive SPICE process reference model [23].	15
Figure 2.11 - From ADAS to AD.	16
Figure 2.12 - Deep neural network and characteristics of a neuron.	17
Figure 2.13 - CNNs processes.	19
Figure 2.14 - Convolutional layer of a CNN.	20
Figure 2.15 - Pooling layer of a CNN, through the two most used processes: max pooling and average pooling.	21
Figure 2.16 - General architecture of autonomous vehicles [34].	22
Figure 2.17 - Strengths and weaknesses of vehicle sensors.	23
Figure 2.18 - Ultrasonic sensor scheme [38].	23
Figure 2.19 - Block diagram of a radar system [42].	25
Figure 2.20 - The different characteristics for radar in autonomous vehicles [38].	25
Figure 2.21 - Lidar sensor scheme [39].	26
Figure 2.22 - Stereo cameras for creating 3D images [43].	27
Figure 2.23 - Most commonly used GNSS [39].	29
Figure 2.24 - Principle of trilateration.	30
Figure 2.25 - Inertial Measurement Unit ¹⁰ .	31
Figure 2.26 - Planning system.	32
Figure 2.27 - Dijkstra's algorithm: A) the red points are the way-points planned by the route planner, B) the blue are matched points and the brown are interpolated points, C) generated trajectory [55].	34
Figure 2.28 - RRT search with increasing iterations [53].	35
Figure 2.29 - Bézier curve for the trajectory generation - the blue curve is the optimal trajectory for the vehicle to follow [56].	35
Figure 2.30 - Control Strategies for Autonomous Vehicles.	36

Figure 2.31 - Multi-hop routing, using dual paths, between the source node and the destination node (adapted from [61]).	39
Figure 2.32 - Dissemination models: (a) the same-direction model, and (b) the opposite-direction model [65].	41
Figure 2.33 - General characteristics of VANETs [68].	43
Figure 2.34 - V2X connectivity [71].	45
Figure 2.35 - Bidirectional electric vehicle charger: a) grid to vehicle mode; b) vehicle to grid mode; and c) Vehicle-to-Home [74].	46
Figure 3.1 - Possible models to configure for testing and validation of CAVs.	47
Figure 3.2 - V-model (left) and agile model (right) diagrams [78].	49
Figure 3.3 - Waterfall model (adapted from [81]).	50
Figure 3.4 - V-model phases (adapted from [78]).	51
Figure 3.5 - Approaches for VVT and architecture of CAVs.	53
Figure 3.6 - XIL simulation and Test drive and the relationship between the virtual/real components.	54
Figure 3.7 - Classification of tools for VVT.	58
Figure 4.1 - Planning system and control system used in the model.	65
Figure 4.2 - Axis systems for the vehicle coordinate system and the world coordinate system.	68
Figure 4.3 - Bicycle model.	69
Figure 4.4 - Parameters defined for the ego vehicle dynamics.	73
Figure 4.5 - Signal conversions.	74
Figure 4.6 - Driving scenario 1.	75
Figure 4.7 - Driving scenario 2.	75
Figure 4.8 - Driving scenario 3.	75
Figure 4.9 - Scenario reader block parameters.	76
Figure 4.10 - Scheme of the scenario setup of the Simulink model.	78
Figure 4.11 - Comparison of a planned trajectory in Cartesian and Frenet coordinates.	79
Figure 4.12 - Bus Editor showing the fields of the Simulink bus "BusEgoAndTargetStates".	80
Figure 4.13 - Vehicle trajectory.	81
Figure 4.14 - Bus Editor showing the fields of the Simulink bus "BusPlannerParams".	83
Figure 4.15 - Bus Editor showing the fields of the Simulink bus "BusMapInfo".	83
Figure 4.16 - Scheme of the local planning system for the ego vehicle to select the optimal trajectory at every sampled time.	84
Figure 4.17-Bus Editor showing the fields of the Simulink bus "BusMIOFrenetStates".	85

Figure 4.18 - Illustration of the relative velocity angle (relative yaw angle) and the relative distance angle (adapted from [120]).	86
Figure 4.19-Bus Editor showing the fields of the Simulink bus "BusTerminalStatesCombination"	88
Figure 4.20 - Bus Editor showing the fields of the Simulink bus "BusTrajectories".	91
Figure 4.21 - Bus Editor showing the fields of the Simulink bus "BusGlobalTrajectory".	91
Figure 4.22 - Bus Editor showing the fields of the Simulink bus "BusFutureTrajectory".	93
Figure 4.23 - Geometry of a capsule-based collision object.	93
Figure 4.24 - Vehicle dimensions of a sedan vehicle.	94
Figure 4.25 - Bus Editor showing the fields of the Simulink bus "BusTrajectoriesInfo".	96
Figure 4.26 - Bus Editor showing the fields of the Simulink bus "BusRefPointOnPath".	96
Figure 4.27 - Scheme of the planning system of the Simulink model.	98
Figure 4.28 – "Planner parameters" subsystem.	99
Figure 4.29 – "Generate terminal states for the ego vehicle" subsystem.	99
Figure 4.30 - "Pulse generator" subsystem.	100
Figure 4.31 - Schematic of lane curvature estimation for LKA ²⁹ .	101
Figure 4.32 - Structure of cruise control system [122].	102
Figure 4.33 – Lower level controller of the Simulink file.	103
Figure 4.34 - Calculation of previewed curvature: 1) curvature derivative, 2) longitudinal velocity and 3) curvature.	104
Figure 4.35 - Values of the path following control system.	105
Figure 4.36 - Scheme of the control system of the Simulink model.	106
Figure 5.1 - Lane centers for: A) 2 lanes, B) three lanes and C) four lanes.	107
Figure 5.2 - The ego vehicle performing its first lane change in each scenario: (1) driving scenario 1; (2) driving scenario 2 and (3) driving scenario 3.	109
Figure 5.3 - Ego vehicle characteristics for the 12 scenarios considered for this study.	110
Figure 5.4 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1A.	111
Figure 5.5 - Ego vehicle at instant $t = 7.6$ sec, when the vehicle starts to slow down to avoid a collision with the target vehicle ahead.	112
Figure 5.6 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1B.	113

Figure 5.7 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1C.....	114
Figure 5.8 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1D.	115
Figure 5.9 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2A.	116
Figure 5.10 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2B.	117
Figure 5.11 - Scenario 2B for instants $t = 11$ sec.	118
Figure 5.12 - Scenario 2B for instant $t = 13.5$ sec.	118
Figure 5.13 - Scenario 2B for instant $t = 15.5$ sec.	118
Figure 5.14 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2C.	119
Figure 5.15 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2D.....	120
Figure 5.16 - Lane change to the right lane at instant $t = 5.2$ sec.....	121
Figure 5.17 - Ego vehicle in cruise control mode at instant $t = 13$ sec.....	121
Figure 5.18 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3A.	122
Figure 5.19 - Ego vehicle in following the lead vehicle mode at instant $t = 15.3$ sec.....	123
Figure 5.20 - Ego vehicle in following the lead vehicle mode at instant $t = 30$ sec.	123
Figure 5.21 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3B.	124
Figure 5.22 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3C.	125
Figure 5.23 – The ego vehicle at instant $t = 11.8$ sec when it is unable to track a feasible trajectory.....	126
Figure 5.24 – The ego vehicle at instant $t = 14.5$ sec when it is finally able to track a feasible trajectory, making a lane change maneuver.....	126
Figure 5.25 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3D.....	127
Figure 5.26 – The ego vehicle searching for a better trajectory at instant $t = 4.2$ sec.	128
Figure 5.27 - Lane change in the instant $t = 5.2$ sec.	128

List of Tables

Table 2.1 - Safety integrity levels of the IEC 61508 standard (adapted from [12]).	8
Table 2.2 - Process Capability Levels [23].	14
Table 2.3 - ANN learning processes [32].	18
Table 2.4 - Speed of sound in dry and humid air (adapted from [40]).	24
Table 2.5 - Usage and applications of vehicle sensors.	28
Table 2.6 - Motion prediction models [52].	33
Table 3.1 - Characteristics between V-model and agile model.	49
Table 3.2 - Test levels of the V-model.	52
Table 3.3 - Model checkers.	59
Table 3.4 - Difference between tools.	60
Table 4.1 - Fixed ego vehicle parameters.	72
Table 4.2 - Actor pose structure in world coordinates ¹⁹ .	77
Table 4.3 - Vehicle profile structure.	95
Table 5.1 - Values defined for the target vehicles for the driving scenario 1.	108
Table 5.2 - Values defined for the target vehicles for the driving scenario 2.	108
Table 5.3 - Values defined for the target vehicles for the driving scenario 3.	108
Table 5.4 - Simulation results.	129

List of Acronyms

VVT	Verification, Validation and Testing
CPS	Cyber-Physical Systems
AVs	Autonomous Vehicles
ADAS	Advanced Driver Assistance Systems
AD	Autonomous Driving
AI	Artificial Intelligence
DL	Deep Learning
V&V	Verification and Validation
ODD	Operational Design Domains
SDLC	Systems Development Lifecycle
XIL	X-In-the-Loop
MIL	Model-In-the-Loop
SIL	Software-In-the-Loop
HIL	Hardware-In-the-Loop
VIL	Vehicle-In-the-Loop
ECU	Electronic Control Unit
CAN	Controller Area Network
CAVs	Connected and Autonomous Vehicles
SAE	Society of Automotive Engineers
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
E/E/PE	Electrical, Electronic, or Programmable Electronic
E/E	Electrical/Electronic
SIL	Safety Integrity Level
ASIL	Automotive Safety Integrity Level
QM	Quality Management
AUTOSAR	Automotive Open System Architecture
ASPICE	Automotive Software Process Improvement and Capability Determination
ANN	Artificial Neural Network
DNN	Deep Neural Networks
CNN	Convolutional Neural Networks
ReLU	Rectified Linear Unit
IoT	Internet-of-Things
RH	Relative Humidity
RaDAR	Radio Detection and Ranging
SRR	Short-Range Radar
LRR	Long-Range Radar
LiDAR	Light Detection and Ranging
CD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
INS	Inertial Navigation Systems
IMU	Inertial Measurement Unit

RRT	Rapidly-exploring Random Tree
VSS	Variable Structure Systems
SMC	Sliding Motion Control
MPC	Model Predictive Control
CVs	Connected Vehicles
ITS	Intelligent Transport System
V2X	Vehicle-to-everything
V2V	Vehicle-to-Vehicle
V2I	Vehicle-to-Infrastructure
V2P	Vehicle-to-Pedestrian
V2N	Vehicle-to-Network
V2G	Vehicle-to-Grid
V2H	Vehicle-to-Home
NHSTA	National Highway Traffic Safety Administration
VANET	Vehicular Ad hoc Network
MANET	Mobile Ad hoc Network
RSU	Roadside Unit
BSM	Basic Safety Message
DSRC	Dedicated Short Range Communication
C-V2X	Cellular V2X
3GPP	3 rd Generation Partnership Project
LTE	Long-Term Evolution
5G NR	Fifth-Generation New Radio
UT	Unit Tests
IT	Integration Tests
ST	System Tests
AccT	Acceptance Tests
FB	Formal-Based
MB	Model-Based
ScB	Scenario-Based
OB	Ontology-Based
SB	Search-Based
FV	Formal Verification
RV	Run-time Verification
TA	Timed Automata
CC	Cruise Control
FLV	Follow the Leading Vehicle
LC	Lane Change
CG	Center of Gravity
DOF	Degrees Of Freedom
LTI	Linear-Time-Invariant

Chapter 1

Introduction

Autonomous Vehicles (AVs) is an emerging topic which increasingly gained attention in recent years, with significant efforts and high-performance resources invested by academia and companies around the world. The focus of the automotive industry has shifted from Advanced Driver Assistance Systems (ADAS), classified by the Society of Automotive Engineers (SAE) as levels 1 and 2 of automation, to Autonomous Driving (AD) of SAE level 3+ [1].

Indeed, autonomous vehicles use AD that enables the vehicle to respond to external conditions by itself based on intelligent agents, such as deep learning. The architecture of these innovative systems is becoming progressively more complex at both hardware and software level which increases the ability of the AV functions. Due to technological advances in these fields, AVs are capable of sensing its environment (through a multitude of sensors), making decisions, and moving safely with reduced or even without human interaction [2].

In this way, AD is the next frontier of the automotive industry and the future of the new road vehicles with the potential to spark a revolution in the transportation sector. Indeed, AVs may lead to disruptive changes in urban mobility and city design. In addition, its adoption can improve traffic safety, significantly reducing accident rates, reduce traffic congestion and solve economic problems. In summary, AVs may pave the way to mitigate human errors and, at the same time, it may provide a mean to optimize the driving process [3].

However, the impressive results achieved by the AV technology challenges the adoption of adequate models (new methods) for verifying, validating and testing those safety-critical systems to ensure their safety and reliability, since a failure in their systems can result in loss of life. AVs must be trained and tested to respond to all driving situations, and testing millions of scenarios extends their development lifecycles. In addition, due to the lack of automation, the software verification and validation (V&V) tends to swallow up 40% to 50% of the total development cost [4], which means that testing these systems is a complex, time-consuming, and cost-intensive task.

All of this represents an obstacle in the deployment of these vehicles and that's why despite the significant progress made in the last decade, only AV prototypes without safety drivers are being tested. Furthermore, tech companies and automotive manufacturers only test these types of vehicles using Operational Design Domains

(ODDs), where the vehicle can operate safely in autonomous mode. An ODD defines when, where, and under what conditions an autonomous vehicle is designed to operate, including geographic, environmental, traffic-related, and temporal limitations, that is, the ODD comprises the static and dynamic attributes within which an Av is designed to function safely.

In contrast to traditional test methods, simulations are undoubtedly the most practical and effective way to test AV systems. Real-world testing requires a minimum total driving distance of 100.8 million km with costs over 100 million euros, and because simulation is conducted in a virtual environment, it is faster (reduces time to market) and less expensive to test millions of driving situations [5]. In addition, simulation enables to test the AVs in dangerous situations, failure modes and based on conditions that rarely happen in the real traffic scenarios. Besides that, as simulations are performed since early stages of the systems development lifecycle, from the design phase, it is easier and cheaper to find and to fix defects, reducing the number of errors in the AV functions. The V-Model is widely used as basis for the development and integration of electric/electronic systems, such as autonomous vehicles.

Virtual driving tests are performed through x-in-the-loop simulation which comprises model-in-the-loop simulation, software-in-the-loop simulation, hardware-in-the-loop simulation and vehicle-in-the-loop simulation, generally performed sequentially. From hardware-in-the-loop simulations, testing includes real physical hardware as electronic control units, which are distributed all over the vehicle and interconnected through different types of communication networks such as the controller area networks (CANs) [6].

As soon as AVs reach higher levels of connectivity, a network of autonomous vehicles capable of communicating with each other (vehicle-to-vehicle communication), with infrastructures (vehicle-to-infrastructure communication) and with pedestrians (vehicle-to-pedestrian communication), has the potential to face the challenges of the new cities.

1.1 Objective

The main objective of this dissertation is the proposal of an autonomous vehicle model. For the development of the model, a literature review is carried out on the technology of autonomous and connected vehicles and on how to test these systems, in order to acquire sufficient knowledge of this matter. In addition, the simulation of the model aims to analyze the performance of the model (driverless vehicle) through critical highway scenarios, and to analyze the decision making made by the model in each scenario.

1.2 Structure

The present work comprises six chapters, whose organization is described as follows.

In chapter 1, Introduction, an introduction to the topic is made, where the objectives and structure of the dissertation are presented.

In chapter 2, Connected and Autonomous vehicles, a definition of autonomous and connected vehicles is presented, as well as all the technology that involves them. In the autonomous vehicles subsection, the several functions for autonomous driving, the architecture of an autonomous vehicle and the artificial intelligence used in the software of these vehicles are introduced. In the connected vehicles subsection, the different types of communication networks are identified and the types of connectivity available for CAVs are characterized.

In chapter 3, Verification, Validation and Testing, the methods used in the autonomous vehicle industry to test their prototypes are introduced. In this chapter, a survey on the techniques and tools used to verify, validate and test the CAVs is also presented.

In chapter 4, Trajectory planning based on adaptive model predictive control: Study of the performance of an autonomous vehicle in critical highway scenarios, the detailed description of the project is presented: the vehicle dynamics, the planning system and the control system.

In chapter 5, Results and discussion, the results obtained are exposed and analyzed. The analyzed results are discussed in detail.

Chapter 6 concludes the dissertation and indicates possible future works and developments around this topic.

Chapter 2

Connected and autonomous vehicles

The introduction of Connected and Autonomous Vehicles (CAVs) will bring changes in the driving environment as we know it. Connected and autonomous vehicles incorporate many different technologies: all the technology of Autonomous Vehicles (AVs), which allows vehicles to operate without human intervention, and all the technology of Connected Vehicles (CVs), which allows autonomous vehicles to be connected with each other and with everything around them.

2.1 Autonomous vehicles

Due to the rapid development of artificial intelligence, its application to the automotive industry has drawn wide attention all over the world. Vehicles are no longer mechanical systems only, but intelligent systems, where the complexity of the embedded systems in today's vehicles is increasing.

Autonomous vehicles are cyber-physical systems in which at least some aspect of a safety-critical control function (e.g., steering, throttle, or braking) occurs without direct driver input. Cyber-physical systems, like AVs, are complex heterogeneous distributed systems, which consists of large number of sensors and actuators connected to a pool of computing nodes [7], as shown in Figure 2.1.

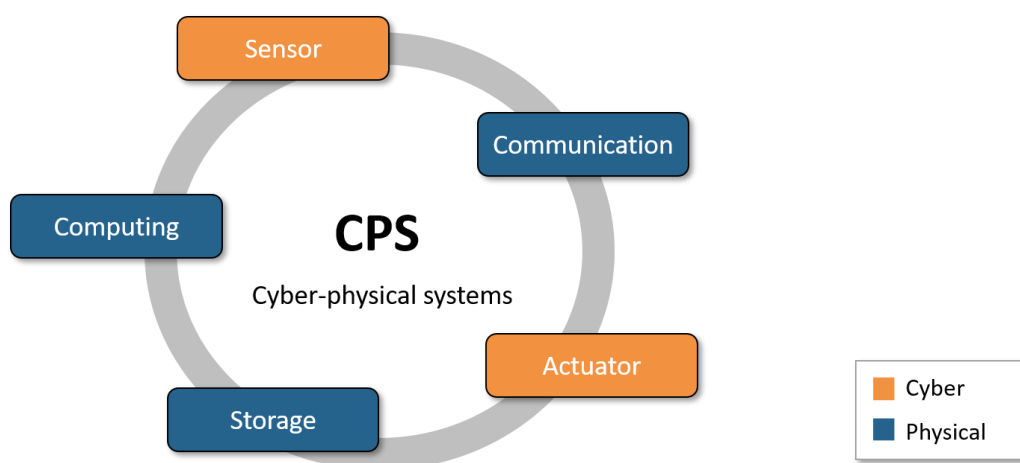


Figure 2.1 - Cyber-physical system of an autonomous vehicle (adapted from [7]).

Through the fusion of information from sensors, computing nodes and actuators (connected through several communications), the AV system perceives and understands changes in the physical environment, analyzes the impacts of such changes, and makes intelligent decisions to respond to the changes by issuing commands to control the physical objects in the system, moving the car safely on the road. Such decisions in conventional vehicles are made by humans and according to the World Health Organization¹, 1.35 million people die every year on roads where 94% of the accidents are caused by human error (e.g., distracted driving, impaired driving, speeding, driver inexperience, driver fatigue) [8].

In this perspective, autonomous vehicles are expected to reduce the number of traffic accidents and fatalities by removing the most common cause of traffic accidents (human errors), as well as improving driving efficiency, such as travel time reduction by reducing traffic congestion and better parking possibilities, and improving the efficiency of energy and/or fuel consumption, increasing environmental sustainability. These social, economic and environmental benefits are the most wanted skills for the AV industry.

2.1.1 Standards

Providing guarantees about the behavior of safety-critical systems, as autonomous vehicles, is imperative and proper certifications are indispensable. Because these systems use advanced software to interact with the physical world, security and safety are primary concerns. In this way, the Society of Automotive Engineers (SAE), the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) are working on new and improved regulations for AV.

The automotive industry follows its own international safety standard, which was designed especially for road vehicles: the well-established ISO 26262 standard². Regarding the cybersecurity of road vehicles, the standards are relatively new (such as ISO/SAE 21434³) since vehicles with advanced electronic technology are still an evolving reality.

¹ <https://www.who.int/data/gho/data/themes/road-safety>

² <https://www.iso.org/standard/68383.html>

³ <https://www.iso.org/standard/70918.html>

Besides that, as a result of reaching new levels of automation in modern vehicles, new taxonomies and definitions are needed. In this way the SAE International standard J3016⁴ establishes a terminology for the levels of automation of road vehicles.

Other important efforts in the automotive industry are in the development of the software architecture of an autonomous vehicle, in which the AUTOSAR standard⁵ is used, and in the quality management of that software, in which ASPICE standard is used. In addition to the standards presented here, there are several other standards for some particular functions (e.g., ISO 11270⁶: Intelligent transport systems - Lane keeping assistance systems).

A. SAE J3014

In 2014, SAE International designed a standard for consumers - the SAE J3016 (“Levels of Driving Automation”), which is the Cybersecurity Guidebook for Cyber-Physical Vehicle Systems and sets the foundation for cybersecurity standards. The standard provides a common taxonomy and definitions (e.g., dynamic driving task, driving mode, request to intervene) for automated driving with full descriptions and examples for each level, as illustrated in Figure 2.2. The standard defines the six levels of driving automation, from SAE Level Zero (no automation) to SAE Level 5 (full vehicle autonomy) [9].

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/Deceleration	Monitoring of Driving Environment	Fallback Performance of Dynamic Driving Task	System Capability (Driving Modes)
Human driver monitors the driving environment						
0	No Automation	the full-time performance by the <i>human driver</i> of all aspects of the <i>dynamic driving task</i> , even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the <i>driving mode</i> -specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the <i>driving mode</i> -specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	System	Human driver	Human driver	Some driving modes
Automated driving system (“system”) monitors the driving environment						
3	Conditional Automation	the <i>driving mode</i> -specific performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> with the expectation that the <i>human driver</i> will respond appropriately to a <i>request to intervene</i>	System	System	Human driver	Some driving modes
4	High Automation	the <i>driving mode</i> -specific performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> , even if a <i>human driver</i> does not respond appropriately to a <i>request to intervene</i>	System	System	System	Some driving modes
5	Full Automation	the full-time performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> under all roadway and environmental conditions that can be managed by a <i>human driver</i>	System	System	System	All driving modes

Figure 2.2- The SAE J3016 levels of driving automation chart, issued in 2014 [9].

⁴ <https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic>

⁵ <https://www.autosar.org/standards/>

⁶ <https://www.iso.org/committee/54706/x/catalogue/>

The automotive industry recently reached SAE levels 2 to 3 [10] and the SAE committee saw the need to explain more clearly each of the six levels and how they relate to increasing consumer safety and convenience. In this way, in 2018 SAE International unveiled the new visual chart for the standard (Figure 2.3).

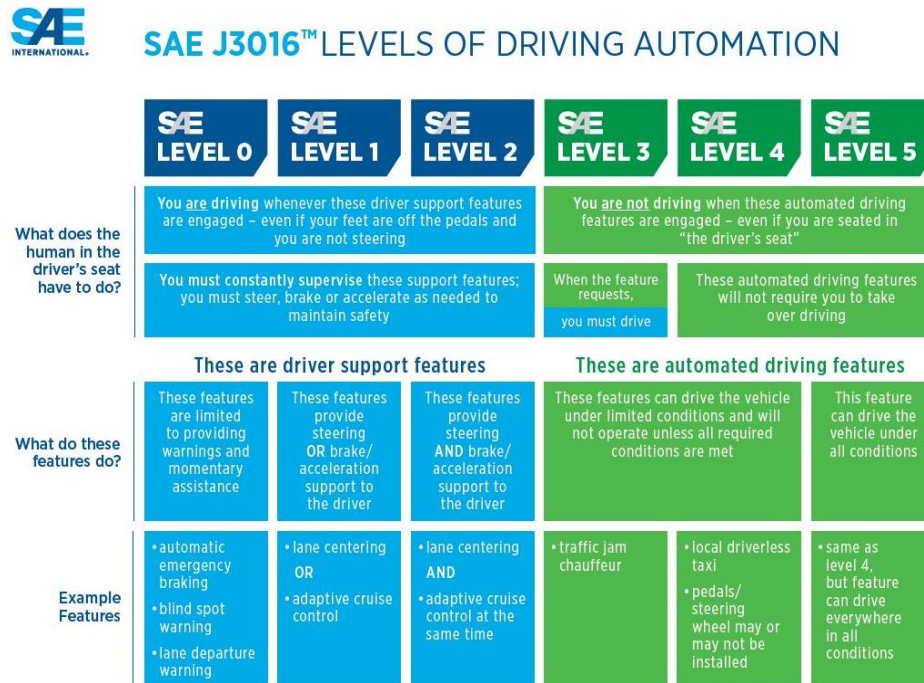


Figure 2.3 - The SAE J3016 levels of driving automation chart, updated in 2018⁴.

B. ISO 26262

Written specifically for automotive, the ISO 26262 (“Functional Safety: Road Vehicles”) is the international standard for the development of safety-critical electronic systems in the automotive industry and is based on the general IEC 61508 standard.

IEC 61508 (“Functional Safety of Electrical/Electronic/Programmable Electronic (E/E/PE) Safety-related Systems”) is a generic functional safety standard for industry-specific standards, especially for safety-critical industries, in the development of safety-critical E/E/PE systems. The standard's main goal is to reduce the risk of failure to a tolerable level when safety functions fail. Therefore, it specifies an allowable frequency of dangerous failure (see Table 2.1) for each Safety Integrity Level (SIL). The SILs are a measurement of performance that correlate with the frequency and severity of the hazards [11], and comprise four levels where SIL 1 represents the lowest level of safety integrity (least dependable) and SIL 4 represents the highest (most dependable).

Table 2.1 - Safety integrity levels of the IEC 61508 standard (adapted from [12]).

Safety Integrity Level	Probability of Failure (on demand)
SIL 4	$\geq 10^{-5}$ to $< 10^{-4}$
SIL 3	$\geq 10^{-4}$ to $< 10^{-3}$
SIL 2	$\geq 10^{-3}$ to $< 10^{-2}$
SIL 1	$\geq 10^{-2}$ to $< 10^{-1}$

There are several industry-specific adaptations of this generic safety standard (IEC 61508 related standards), as shown in Figure 2.4, in which ISO 26262 has specifications for the sector of electrical/electronic (E/E) systems within road vehicles.

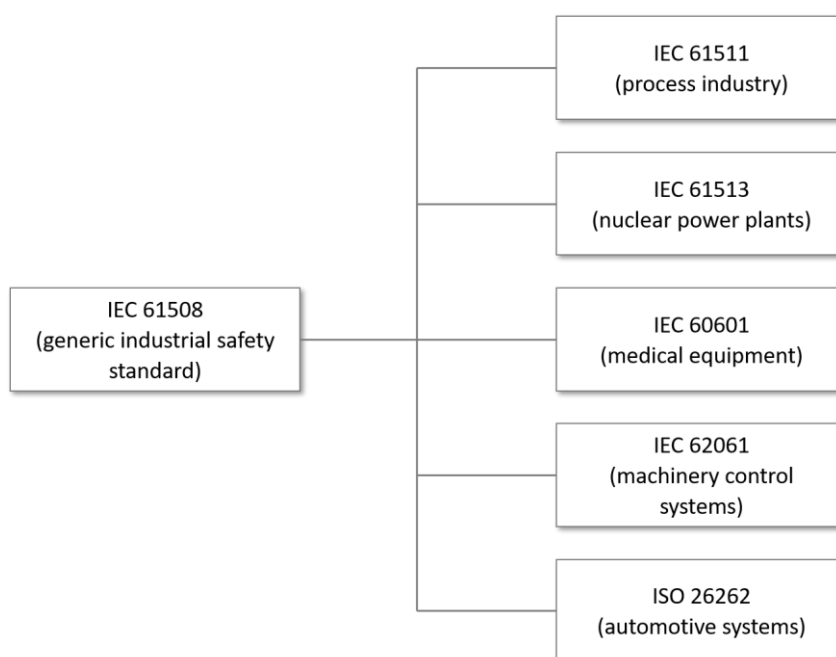


Figure 2.4 - IEC 61508 and related standards (adapted from [11]).

ISO 26262 is a risk-based safety standard where the goal is to ensure safety throughout the lifecycle of automotive systems, covering all of the functional safety aspects of the entire development process (requirements specification, design, implementation, verification and validation).

The first edition of the standard was published in 2011 (ISO 26262:2011) and it was limited to passenger cars. In 2018 was updated and the second edition was released (ISO 26262:2018) with an extended scope from passenger cars to all road vehicles except mopeds, including the requirements and the supporting processes for trucks, buses, trailers and semi-trailers [13]. As shown in Figure 2.5, the first edition of ISO 26262 consists of ten parts, nine normative parts and one guideline. In the second edition, two

parts were added: part 11 (guideline) and part 12 (normative part) making a total of twelve parts, and some parts have been renamed: Part 7 was changed from “Production and operation” to “Production, operation, service and decommissioning” and Part 10 changed from “Guideline” to “Guidelines”, as illustrated in Figure 2.6.

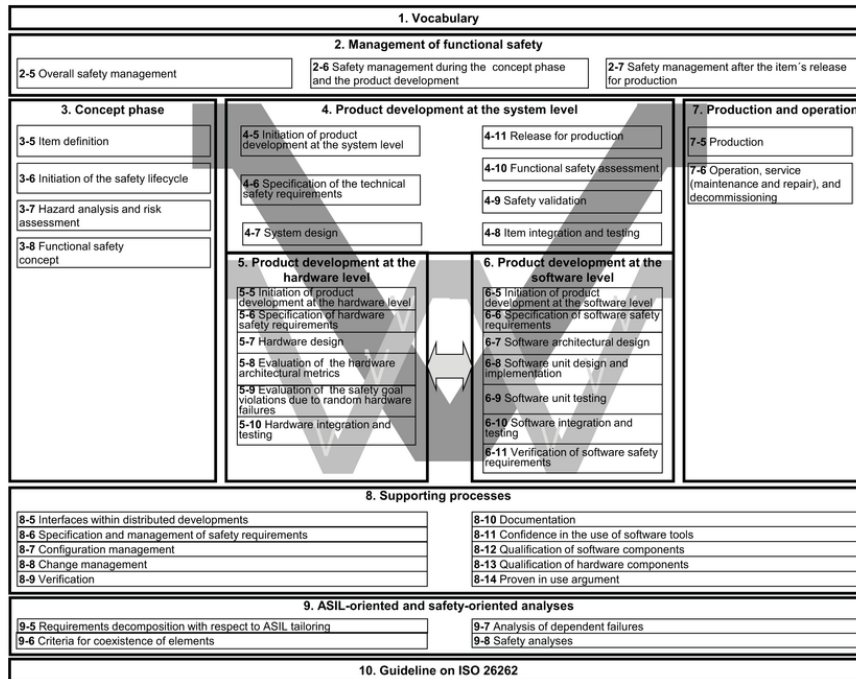


Figure 2.5 - The ISO 26262:2011 structure [14].

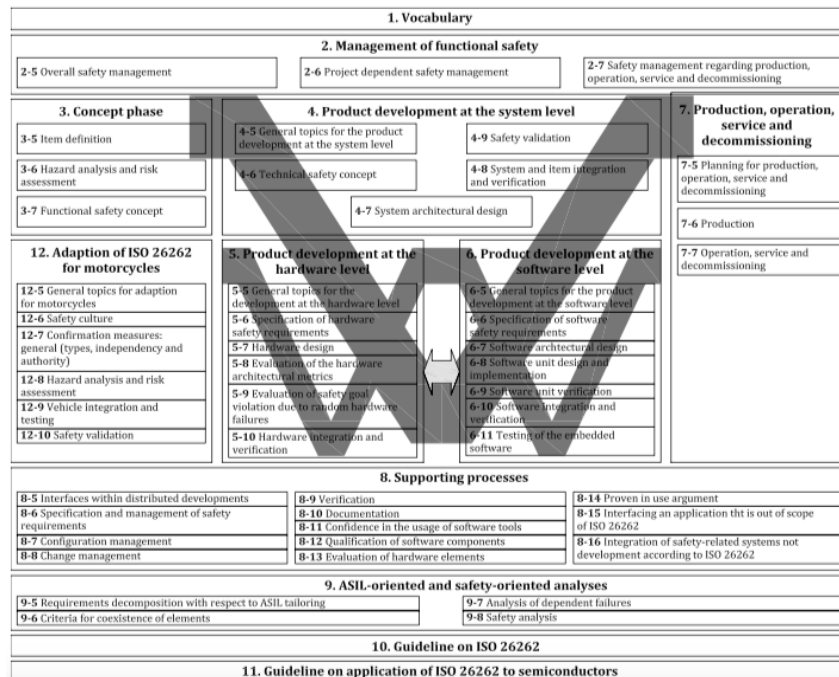


Figure 2.6 - The ISO 26262:2018 structure⁷.

⁷ <https://semiengineering.com/iso-262622018-2nd-edition-what-changes/>

In both editions, the shaded “V”s represent the interconnection between parts since Part 3-7 is the ISO 26262 safety lifecycle and addresses specifically the hardware and the software development lifecycles. The second edition of the standard includes a new part (Part 3) defining motorcycles specific requirements in the safety lifecycle [13].

One of the key components of ISO 26262 is the Automotive Safety Integrity Level (ASIL) for risk classification. The ASILs range from ASIL A (represents the lowest risk level) to ASIL D (represents the highest risk level) and there is also the Quality Management (QM) level that indicates a risk level below than the ASIL A (there is no need to implement additional risk reduction measures) [14]. As illustrated in Figure 2.7, the parameters for the ASILs classification are based on three main classes: classes of severity (the type of injuries to the driver and passengers in an incident), classes of probability of exposure (how often the vehicle is exposed to the hazard in the operational scenario) and classes of controllability (how much the driver can do to prevent the injury) [15].

Severity	Exposure	Controllability		
		C1 (Simple)	C2 (Normal)	C3 (Difficult, Uncontrollable)
S1 LIGHT AND MODERATE INJURIES	E1 (Very low)	QM	QM	QM
	E2 (Low)	QM	QM	QM
	E3 (Medium)	QM	QM	A
	E4 (High)	QM	A	B
S2 SEVERE AND LIFE THREATENING INJURIES – SURVIVAL PROBABLE	E1 (Very low)	QM	QM	QM
	E2 (Low)	QM	QM	A
	E3 (Medium)	QM	A	B
	E4 (High)	A	B	C
S3 LIFE THREATENING INJURIES, FATAL INJURIES	E1 (Very low)	QM	QM	A
	E2 (Low)	QM	A	B
	E3 (Medium)	A	B	C
	E4 (High)	B	C	D

QM (Quality Management)
Development supported by established Quality Management is sufficient.

A lowest ASIL
Low risk reduction necessary

B

C

D highest ASIL
High risk reduction necessary

Figure 2.7 - Determination of an ASIL⁸.

Each of these classes is divided into sub-classes. Severity has four classes ranging from “no injuries” (S0) to “life-threatening (survival uncertain), fatal injuries” (S3). Exposure has five classes covering the “incredible” (E0) to the “highly probability” (E4). Controllability has four classes ranging from “controllable in general” (C0) to “difficult to control or uncontrollable” (C3). All of the sub-classifications are analyzed and combined to determine the ASIL level [16].

⁸ <https://www.apiv.com/en/insights/article/what-is-asil-d>

C. ISO/SAE 21434

One of the pillars in the development of safety-critical systems is security and the SAE J3061 standard emphasizes that safety cannot be guaranteed without securing the system. Some discussions during the development of ISO 26262 addressed this topic and the ISO 21434 (“Road vehicles: cybersecurity engineering”) is under development [17].

SAE J3061 standard (“Cybersecurity Guidebook for Cyber-Physical Vehicle Systems”) provides the guiding principles for implementing a complete cybersecurity process into cyber-physical vehicle systems and is a predecessor of ISO/SAE 21434. The guidebook states that the new ISO/SAE 21434 standard requires an appropriate vehicle development lifecycle process for E/E systems within road vehicles – the ISO/SAE 21434 standard uses the characteristic V-model workflow of ISO 26262 for product development [18]. SAE J3061 relates security and safety processes to each other, determining whether there are cybersecurity threats that can potentially lead to safety violations. However, ISO/SAE 21434 security process is decoupled from safety.

In 2016, SAE International and ISO started a collaboration to develop joint standards and established four areas of common interest, where one of them is the automotive cybersecurity [17]. In this way, ISO/SAE 21434 standard introduces a set of guidelines for securing high-level processes in connected vehicles and is defined similarly to the structure described in ISO 26262 [19].

ISO/SAE 21434 structure identifies the guidelines for cybersecurity processes at all stages of the vehicle's life cycle: cybersecurity process overview and interdependencies, continuous cybersecurity activities (continuous risk assessments and vulnerability management), risk assessment, concept phase (item definition and cybersecurity concepts), product development and post-development phases (production and security operations) and distributed cybersecurity activities [20], as illustrated in the Figure 2.8.

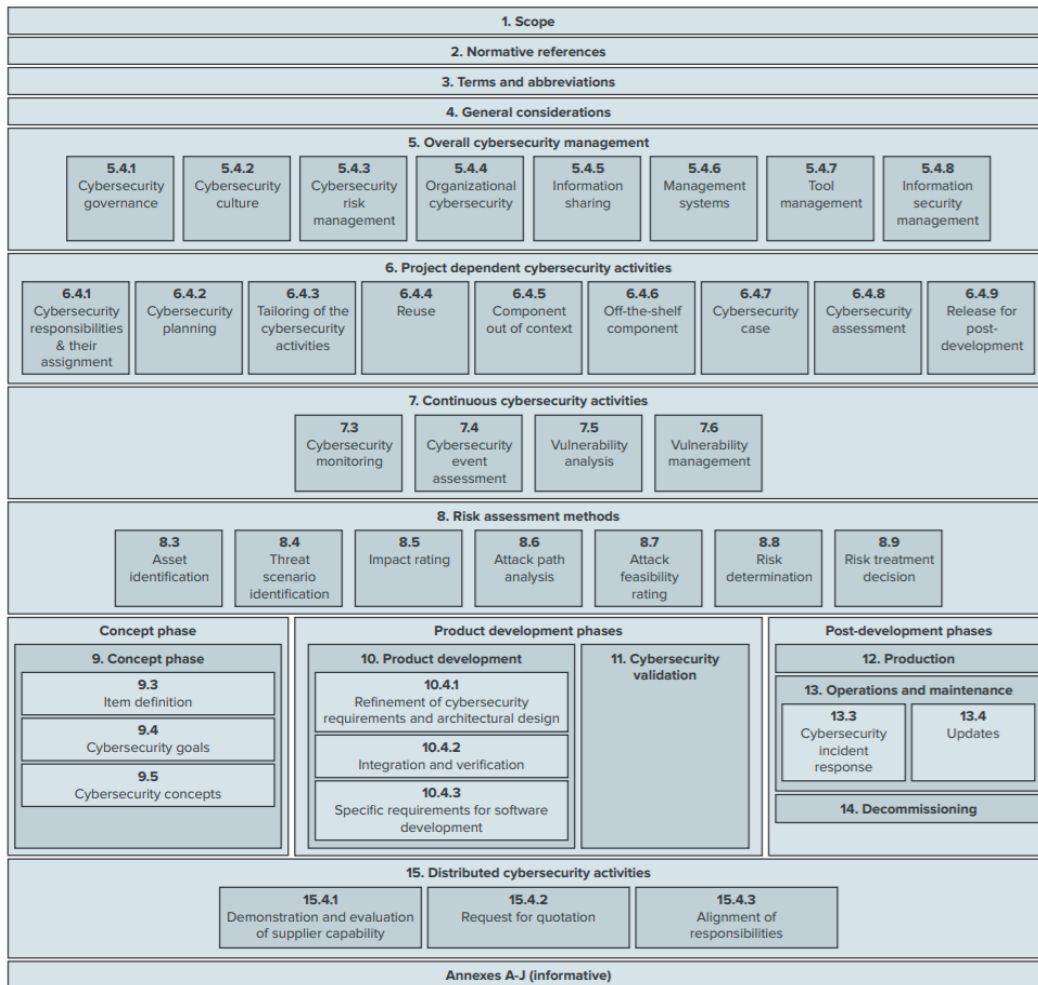


Figure 2.8 - The ISO/SAE 21434 standard structure [20].

D. AUTOSAR

Autonomous vehicles are extremely complex vehicles and part of that complexity involves the number of internal subsystems found within the vehicle's own electronic system. Modern vehicles have more than 100 electronic control units (ECUs) and each of them contains thousands of functions.

AUTOSAR (AUTomotive Open System Architecture) is the most popular industrial standard in the automotive field for automotive E/E architectures, especially because one AUTOSAR application can be deployed on multiple ECUs. AUTOSAR establishes an open and standardized software architecture for software development of automotive ECUs with components to improve interoperability [21] – with the AUTOSAR, it is possible to develop the software independent from the ECU and this software can be transferred or used in different systems or ECUs.

As is presented in Figure 2.9, the AUTOSAR architecture consists of three main layers of software: application layer, basic software layer and the runtime environment

layer. The application layer consists of software components that are mapped on the ECU. The interaction between them and between the application layer/basic software layer is routed through the runtime environment which is a middleware that provides a communication abstraction. The basic software is the standardized software layer and is necessary to run the functional part of the software as well as providing services for accessing the hardware layer [14].

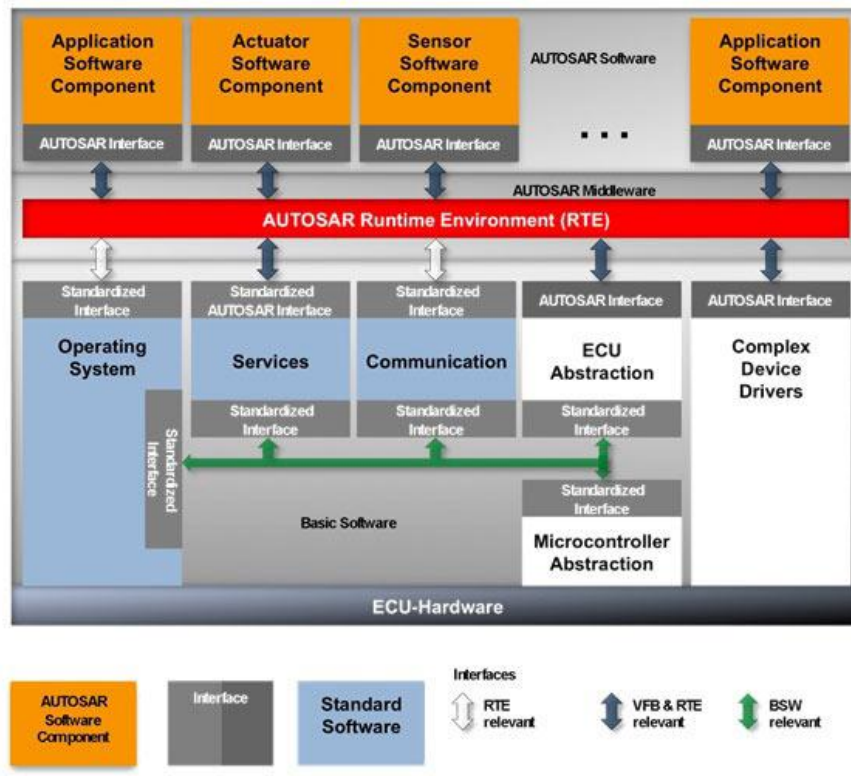


Figure 2.9 - AUTOSAR reference structure [14].

As AUTOSAR is not a security standard, it cannot guarantee the security of the system by itself.

E. ASPICE

Software development is in a constant state of improvement, especially in the automotive industry. In this way, the international ASPICE standard provides a framework for establishing and evaluating the processes required for the automotive software development.

ASPICE represents a variant of the international standard developed by the ISO and IEC joint subcommittee: ISO/IEC 15504. The ISO/IEC 15504 standard (“SPICE - Software Process Improvement and Capability Determination”) represents an

international standard that establishes a framework to build evaluation processes and improve software development processes [22].

The Automotive SPICE (ASPICE) consists of a process assessment model and a process reference model. The process assessment model is used to perform the evaluation of the process capability (capability determination) on the development of automotive software and embedded systems. The process capability defines six levels (see Table 2.2) that constitute a rational way of progressing through improvement of the capability of any process [23].

Table 2.2 - Process Capability Levels [23].

Capability Levels	Description
Level 0: Incomplete process	The process is not implemented or fails to achieve its process purpose.
Level 1: Performed process	The implemented process achieves its process purpose (however, there may be gaps).
Level 2: Managed process	The previously process is now implemented in a managed fashion (planned, monitored and adjusted) and its work products are appropriately established, controlled and maintained.
Level 3: Established process	The previously process is now implemented using a defined process that is capable of achieving its process outcomes.
Level 4: Predictable process	The previously process now operates predictively within defined limits to achieve its process outcomes. Quantitative management needs are identified, measurement data are collected and analyzed to identify assignable causes of variation.
Level 5: Innovating process	The previously process is now continually improved to respond to organizational change.

ASPICE has its own process reference model (Figure 2.10) that describes the lifecycle of automotive electronic products with three different process, categorized as follows: Primary Life Cycle Processes, Organizational Life Cycle Processes and Supporting Life Cycle Processes and it is essential to assess the maturity level of these processes. ASPICE purpose is to provide a scheme for evaluating the capability of software processes and a path for their improvement (quality of the software refinement) [23].

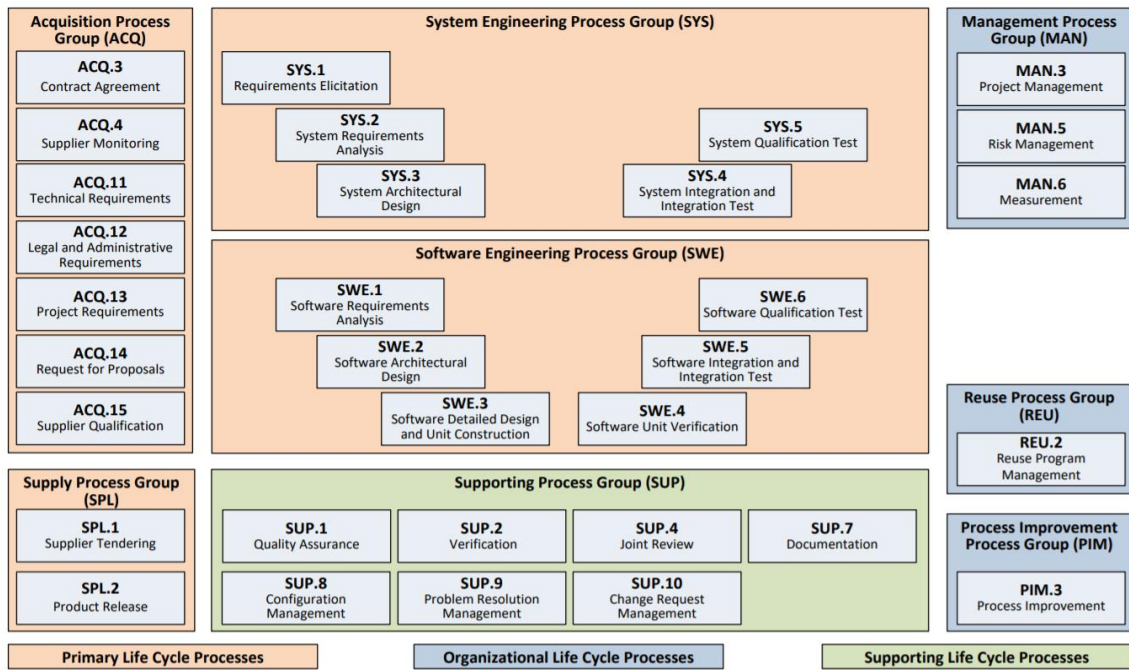


Figure 2.10 - Automotive SPICE process reference model [23].

2.1.2 Evolution from ADAS to AD

In recent years, there has been a worldwide demand for higher levels of automation and, therefore, huge progress has been made towards the development of automated driving technologies around the world [24].

Automated driving is a driving enhanced by the existence of autonomous (sub)systems that support the driver (dedicated control) while he is in control of the vehicle or is able to timely get back in control of the vehicle. These (sub)systems are called Advanced Driving Support Systems (ADAS) and are positioned in SAE levels 1 and 2. On the other hand, the extreme end result of automated driving is autonomous driving (SAE Level 3+), where at this stage no human driver needs to be active in the control of the vehicle. An Autonomous Driving (AD) system is able to take the entire vehicle's control and be in charge of driving the vehicle when it is authorized [25]. Indeed, the AD system is integrated in vehicles that are already equipped with several ADAS [26].

Current progress in the development of ADAS and AD is based on a wide range of technological advances in the field of artificial intelligence (AI). More and more ADAS are entering the market to improve both safety and comfort by assisting the drivers in their driving task, and the continuous integration of these driver assistance systems enables vehicles to handle more and more traffic situations autonomously until they fully

reach autopilot operations (SAE level 5) - transition from assisted driving to fully autonomous driving [27], as shown in Figure 2.11.

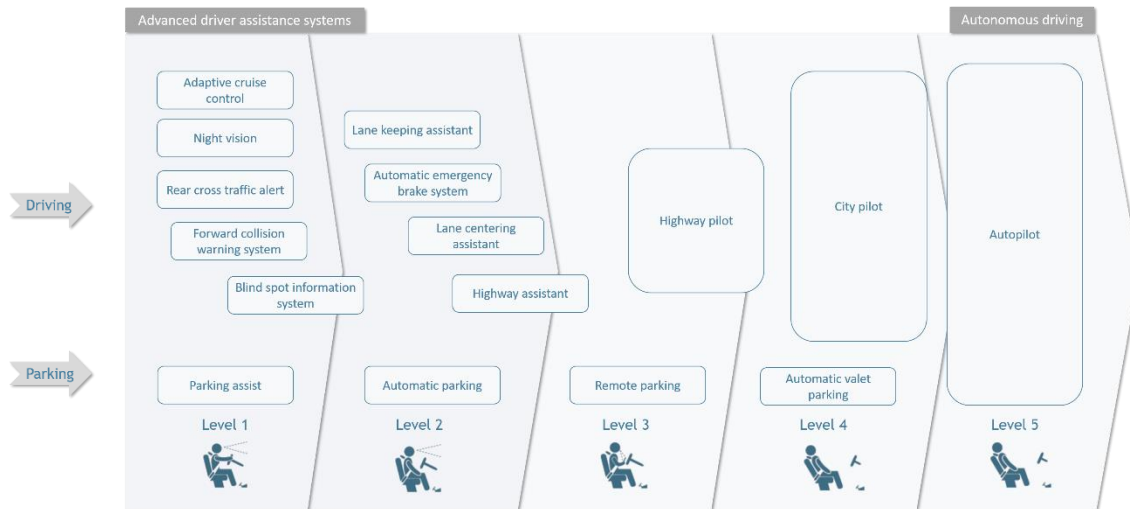


Figure 2.11 - From ADAS to AD.

SAE levels 1 and 2 include assisted driving technologies, but they differ from each other according to the number of technologies working together simultaneously - SAE level 1 only can take control over one functionality at a time. At SAE levels 3, 4 and 5, the driving tasks are no longer performed by the human driver and are carried out by the automated system. While SAE level 2 vehicles require drivers to have at least fingertips in the wheel, SAE level 3 vehicles enable drivers to take their hands off the wheel and feet off the pedals in specific circumstances (conditional automation capabilities). SAE level 4 vehicles provide full automation under most circumstances, and SAE level 5 vehicles provide full automation under all circumstances [28]. Some of these systems have not yet been fully embedded into commercial vehicles and, therefore, currently the highest level of automation available to the public is a SAE level 3.

ADAS and AD extract useful information from various sensors, such as in-vehicle cameras and radars, which sense information from the outside world. On the one hand, ADAS use environment sensors to improve driving comfort and traffic safety by assisting the driver in recognizing and reacting to potentially dangerous traffic situations. On the other hand, AD systems, due to their complex architecture of advanced software and hardware components (in Section 2.1.4), use environment sensors to sense the surrounding environment, plan a path, and implement that path into concrete driving actions [29]. The interaction between human and machine (Human-Machine interface) lowers as the vehicle's capability increases.

2.1.3 Artificial intelligence in automotive software

Autonomous vehicles use AI systems that employ machine learning techniques, specifically deep learning, in order to make decisions that in conventional vehicles are taken by humans. Deep learning is becoming crucial for the development of automotive software for autonomous vehicles, since one of the main tasks of the autonomous driving is the detection and identification of objects in the surrounding environment. An autonomous vehicle needs to accurately detect cars, pedestrians, cyclists, road signs, and other objects in real-time in order to make the right control decisions that ensure safety [30].

A. Deep learning

Deep Learning (DL) is a branch of machine learning based on artificial neural networks with multiple layers between the input and output layers (deep neural networks). An Artificial Neural Network (ANN) is a computing system designed to mimic the human brain (biological neural networks) in information processing, which is capable of learning.

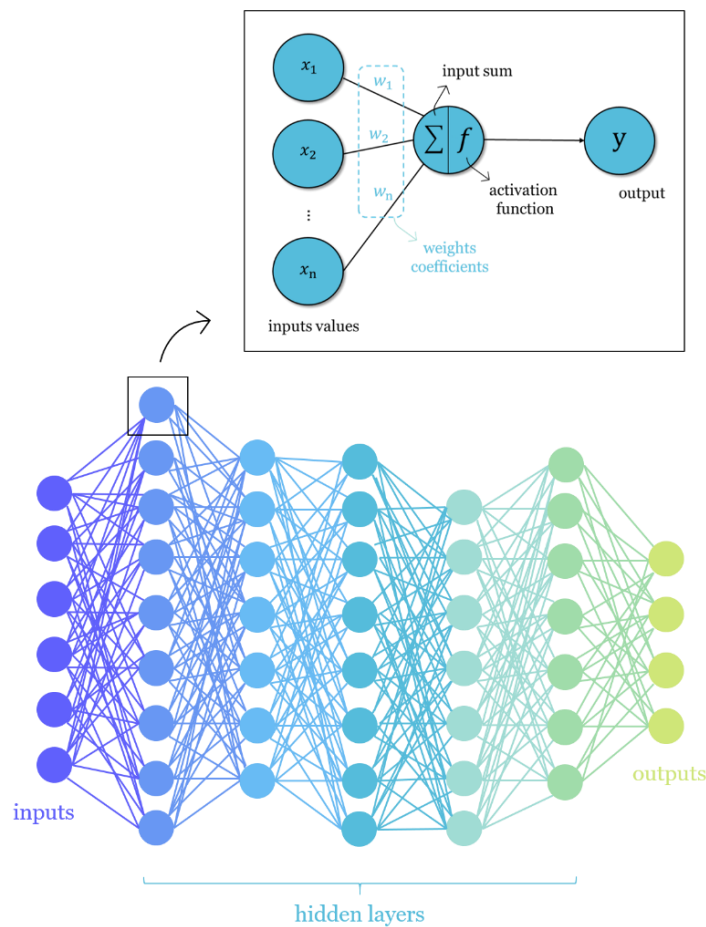


Figure 2.12 - Deep neural network and characteristics of a neuron.

An ANN is a model based on a collection of connected nodes, called "artificial neurons", organized in layers. Depending on the number of existing layers in the model, the ANNs are classified from shallow neural networks (single or few hidden layers) to deep neural networks (multiple hidden layers). In an ANN, each artificial neuron (Figure 2.12) has weighted inputs, an activation function (transfer function) and produces a single output which can be sent to multiple other neurons [31]. In autonomous vehicles the input neurons get activated from the AV sensors that perceive the environment, while the other neurons get activated from the previous neuron activations.

An ANN can also be defined by the direction of the signal flow in the network, it can have a feedforward architecture, where the signals travel in one way (from input to output layer), or have a feedback architecture, where the signals can travel in both directions [31]. To learn, a neural network needs to be trained. The learning process can be conducted through supervised learning, unsupervised learning and reinforcement learning (Table 2.3). In automotive industry, the neural networks are widely trained using supervised learning, however, and depending on the application, some automotive applications combine supervised training with reinforcement training [30].

Table 2.3 - ANN learning processes [32].

Supervised Learning	Unsupervised Learning	Reinforcement Learning
<p>The training dataset contains labelled inputs paired with desired outputs (known outputs), where the ANN use the training data to learn a link between the input and the outputs. Thus, training data can be generalized, and the neural network can be used on new data with some accuracy.</p>	<p>The training dataset is only based on a group of unlabeled input data (there is no fixed output variables). The model learns from the data, discovers patterns and features from it and returns the output.</p>	<p>This strategy is built on observation, where the training model uses an agent that learns from experience using a trial-and-error approach in a set environment. The ANN makes a decision based on feedback from the agent's own actions and if the feedback is negative, the network adjusts its weights to be able to make a different required decision the next time.</p>
<p><u>Classification algorithms:</u> these processes help categorizing the training dataset into classes based on different parameters, predicting the target class for each category of the data. Predict and classify discrete values.</p> <p><u>Regression algorithms:</u> these processes help finding the relationship between variables by predict the outcome of an event based on the relationship between the variables obtained from the dataset. Predict continuous values.</p>	<p><u>Clustering algorithms:</u> these processes help to classify unlabeled data sets, finding similarities in the training data and grouping similar data into a cluster group.</p>	

The software of autonomous vehicles is based on Deep Neural Networks (DNNs), which typically, are feedforward networks with many hidden layers and can be trained more in-depth to find patterns with high performance, even for complex nonlinear relationships [31]. Automotive applications can easily reach DNNs with 10 hidden layers and thousands of nodes, in which a large number of training datasets (images or other sensor data) are used to train the neural network of the vehicle [30].

B. Convolutional neural networks

Nowadays, Convolutional Neural Networks (CNNs) are considered as the most widely used DNN in autonomous vehicle software for object detection, since the architecture of a CNN is inspired by the organization of the visual cortex of the human brain. CNNs are one of the best learning algorithms for understanding image content, including image classification and segmentation, object detection and video processing.

When it comes to image processing, traditional DNNs (multilayer neural networks) have become unfeasible, because an image is a matrix of pixel values and each input in the DNN corresponds to a pixel in the image. In this way, the number of weights (input dependent) rapidly becomes unmanageable for large images and difficulties arise while training the network. CNNs can be used to solve the problem for larger inputs such as high-resolution images.

Convolutional neural networks are feedforward DNNs that manage data in the form of arrays that take into account spatial dependencies in an image (input) [30]. CNNs are also composed of layers, but those layers are not fully connected as in the traditional DNNs. Convolutional networks can be divided in two processes: feature extraction process and classification process, where fully connected layers are only used in the classification process (Figure 2.13). CNNs alternate between convolutional layers (with activation functions) and pooling layers, followed by one or more fully connected layers at the end of the network that results in the output layer [33].

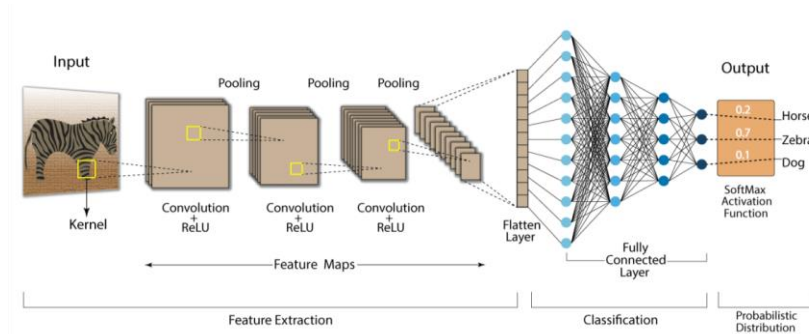


Figure 2.13 - CNNs processes⁹.

⁹ <https://developersbreach.com/convolution-neural-network-deep-learning/>

A convolutional layer (Figure 2.14) is composed of a set of convolutional kernels (also called filters) where each neuron acts as a kernel. Kernels are sets of weights (randomly generated vectors) that slide throughout the input image extracting different features from it and produce a feature map that is the output of the convolutional layer. Given the fewer number of parameters (due to weight sharing), convolution networks are more efficient to train [33]. Initial layers of convolution learn generic information and last layers learn more specific/complex features.

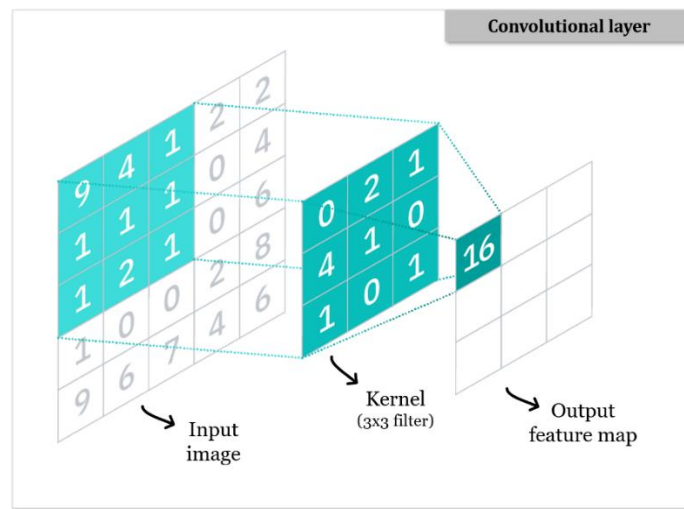


Figure 2.14 - Convolutional layer of a CNN.

After every convolutional layer, an activation function is deployed as a decision function that helps in learning patterns by adding non-linearity properties to the network. The most efficient function for CNNs is ReLU (Rectified Linear Unit) since it speeds up the network training [33].

A pooling layer (Figure 2.15) operates on each feature map to reduce the dimensionality of the network, reducing the number of parameters and computation in the network. This is possible because after features are extracted, their exact location becomes less important as long as its approximate position relative to others is preserved. The use of pooling operation also helps to extract a combination of features invariants to translational shifts and small distortions. The most commonly used pooling formulations in CNNs are the max pooling - which returns the maximum value from the portion of the image covered by the kernel - and the average pooling - which returns the average of all the values from the portion of the image covered by the kernel [33].

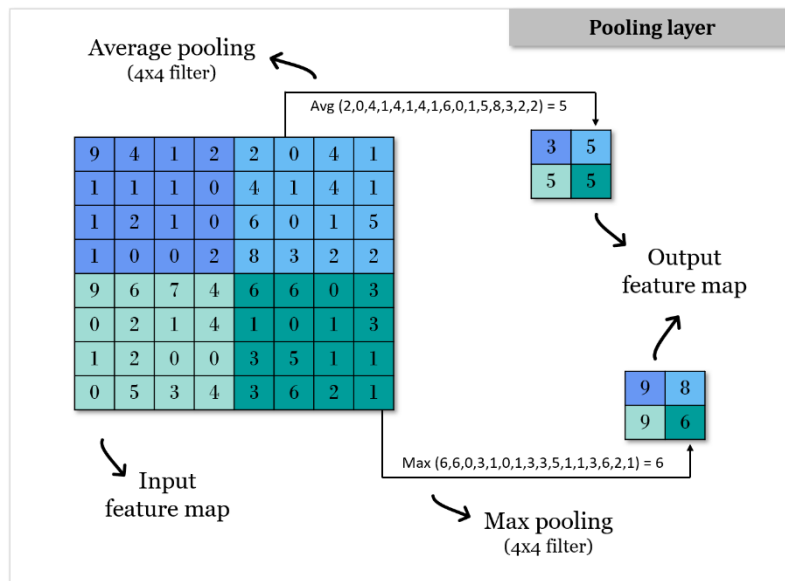


Figure 2.15 - Pooling layer of a CNN, through the two most used processes: max pooling and average pooling.

The last layer of the features extraction process is the flatten layer, where after the final convolution layer, ReLU, and pooling layer the output feature map (matrix) is converted into a vector (one dimensional array). The output from the flatten layer is fed into a fully-connected layer. The flattened matrix goes through a fully connected layer to learn features and classify data.

2.1.4 Architecture

Due to technological advances in the field of artificial intelligence, Internet-of-things (IoT) and cloud computing, AVs are able to learn and make intelligent decisions. The core competencies of an autonomous vehicle software are based on a multi-layer of Perception-Planning-Control algorithms, to identify objects, interpret situations, and make decisions when navigating on roads. To help the vehicle's software interact with the environment (collecting data from the environment and reacting to it), an autonomous vehicle makes use of hardware components such as sensors and actuators.

Autonomous vehicles use a combination of several on-board sensors that, by working together, provide a map of their surroundings and help to detect the speed and distance of nearby objects. These sensors activate the various actuators of the vehicle, which in turn, will generate the order to activate the physical movements in the vehicle, converting energy (often electrical, pneumatic, or hydraulic) into mechanical force.

Figure 2.16 presents the functional architecture of an autonomous vehicle, highlighting the interaction between the components of the vehicle's software and hardware.

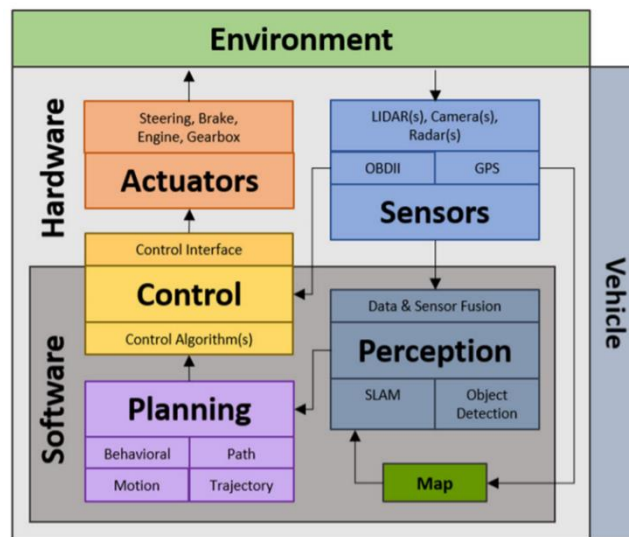


Figure 2.16 - General architecture of autonomous vehicles [34].

A. Perception system

The perception layer provides the environment model (through the vehicle's sensors) and the vehicle localization (through vehicle's navigation system). The perception system collects information and extract relevant knowledge from the environment [35].

The environment model is built through the perception of the driving environment, where data is acquired by gathering different information from through various on-board sensors and through the vehicular communications [34]. This environmental perception includes the identification of obstacles (e.g., other vehicles, pedestrians) and potential road hazards, the detection of traffic signs and road markings, and the categorization of all data through their semantic meaning [36]. In this layer, all the received outputs from the multiple sensors are combined (sensor fusion) to reduce the weaknesses of each sensor (Figure 2.17), where the limitations of any sensor (e.g., operating conditions, resolution, types of objects detectable) are potentially complemented by the strengths of another. By using a combination of sensors that work together, the system is able to create a complete and more reliable model of the surrounding environment that is far beyond the reach of individual sensors [37].














































	Camera 	LIDAR 	Radar 	Ultrasonic 	Fusion 
Field of View					
Range					
Velocity Resolution					
Angular Resolution					
Adverse weather					
Darkness/Ambient Light Disturbance					
Object classification/Semantic Information					
Complete detection of all object surfaces					

Figure 2.17 - Strengths and weaknesses of vehicle sensors.

This layer, in addition to providing the environment model, also provides the localization of the vehicle. Through the vehicle's navigation system, it is possible to calculate its global and local location, determining its position [35].

1) Vehicle's sensors

Intelligent observability is one of the prerequisites for autonomous driving. The sensors are able to sense the external environment (e.g., through obstacle detection) and provide data to the vehicle's software in order to allow the vehicle to take actions on its own. Each of these technologies uses a different part of the electromagnetic spectrum to collect information.

Ultrasonic sensor

Ultrasonic sensors consist of a transmitter and a receiver, as shown in Figure 2.18, and are used to measure the distance to an object by sending high frequency sound waves (range of 20 kHz to 40 kHz).

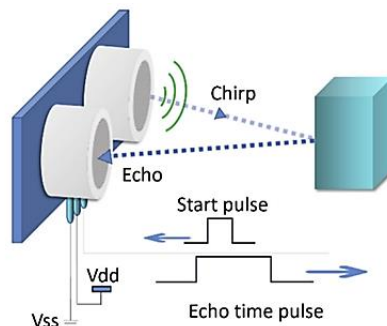


Figure 2.18 - Ultrasonic sensor scheme [38].

Ultrasounds are based on the time-of-flight principle, measuring the time difference between the emission of a signal (sound wave) and its return to the sensor, after this signal is reflected by the object [39]. The distance can be calculated as follows:

$$d = c * \frac{\Delta t}{2} \quad (2.1)$$

Where d is the distance to the object, c is the speed of sound (Table 2.4), which depends on relative humidity (RH) and temperature (θ), and the Δt is the time difference, which needs to be divided by 2 since the total time consist of the time of the emitted signal and the time of the received signal.

Table 2.4 - Speed of sound in dry and humid air (adapted from [40]).

θ (°C)	RH = 0 %	Exact speed of sound (CE)		Instantaneous speed of sound (CS)	
		RH = 30 %	RH = 100 %	RH = 30 %	RH = 100 %
20	343.210	343.592	344.489	343.800	344.696
30	349.015	349.721	351.392	350.180	351.824
40	354.725	355.977	358.970	356.754	359.596
50	360.344	362.481	367.677	363.581	368.213

Ultrasonic sensors are robust and provide reliable distance data, regardless of environmental factors (for example, at night, with fog) and regardless of the object's color and type [39]. However, the range of these vehicle sensors is limited to less than 10 meters, which means this sensors can only be used for nearby obstacle detection [41].

RADAR

RADAR (Radio Detection And Ranging) is a detection system that uses radio waves to measure the position and velocity of objects relative to the vehicle, determining the range (distance), the relative velocity, and the direction of other vehicles [35].

The measurement of the relative velocity is estimated by Doppler effect, calculating the difference in frequencies between the transmitted and the reflected radio waves. When a fixed frequency radio wave sent by the transmitter (from the radar sensor) continuously strikes an object that is moving towards or away from the transmitter, the frequency of the reflected radio wave will change, and it is possible to calculate the speed of that object [38]. The radar system uses a radio frequency generator to emit radio waves (Figure 2.19). The return waves are then picked up by the receiving antenna and are amplified and filtered in the radar system. After that, the signal pass through an A/D

converter to convert the analog signal to a digital signal to be processed by the interface of a computer [42].

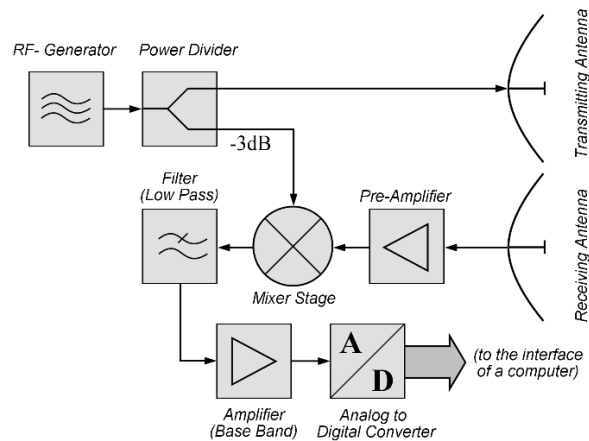


Figure 2.19 - Block diagram of a radar system [42].

There are two main kinds of automotive radars: the Short-Range Radar (SRR) and Long-Range Radar (LRR) and each of them can perform different functions. As shown in Figure 2.20, radars for intelligent vehicles operate at frequencies of 24/76/77 GHz and are a trade-off between range and field of view, where SSRs generally have a shorter detection range, but a bigger field of view. Today, autonomous vehicles have a set of both types of radars [38].

Application	Detection Range	Field of View	Technology
Adaptive cruise control (ACC)	150 ~ 200 m	$\pm 10 \sim 20$ degrees	Single beam, 24 GHz
Forward collision warning and precrash detection	40 ~ 90 m	$\pm 35 \sim 45$ degrees	Single beam, 76 GHz/24 GHz
Blind spot detection, lane change assist and cross-traffic detection	30 ~ 40 m	$\pm 40 \sim 50$ degrees	Single beam, 76 GHz/24 GHz
ACC with stop and go	Multiple ranges	Multiple ranges	Multimode electronically scan

Figure 2.20 - The different characteristics for radar in autonomous vehicles [38].

Radar sensors are robust and usually provide reliable data for all weather conditions - even in adverse weather conditions. Radar waves are generally immune to high luminosity, rain, fog, snow, and even dust [42]. However, due to the low resolution of the radar data, objects can be detected, but not classified - distance sensors have greater difficulty in identifying and differentiating between objects [39].

LIDAR

LIDAR (Light Detection And Ranging) is able to measure target distance and produce a three-dimensional map around the vehicle, using light in the form of a pulsed laser to measure ranges.

Lidar systems are also based on the time-of-flight principle, but instead of radio or ultrasonic waves, the lidar sensors emit infrared laser pulses that when reflected by an object are captured by the sensor receptor (photo-detector) [35]. These sensors measure the difference in time between the laser pulse emitted by an infrared laser diode (reflected off of a rotating mirror) and the returned laser pulse received by the photo-receiver (Figure 2.21). They emit up to one million laser pulses per second (3D Lidar systems integrate 4 to 128 lasers), and summarize the results on a high-resolution 3D map of the environment [39].

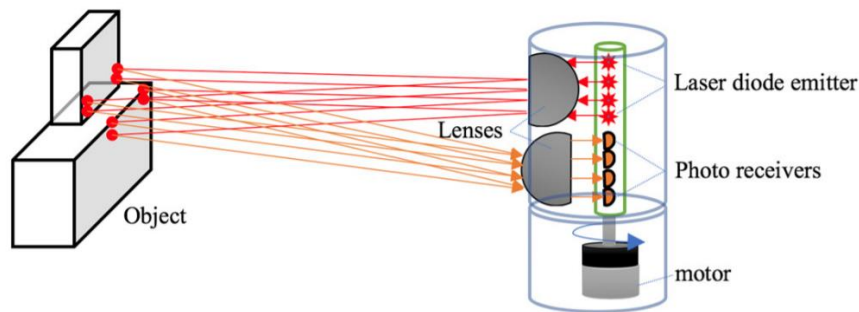


Figure 2.21 - Lidar sensor scheme [39].

Lidar systems are classified into rotating lidar systems and solid-state lidar systems. In rotating lidar systems, the rotation of the sensor makes a field of view of up to 360° possible, while solid state lidar systems have a field of view that oscillates between 20° and 45° [39]. Lidar sensors can measure obstacles in an 800m range of pulse [35].

Infrared emissions are in ranges of 905 nm or 1550 nm, where 905 nm emissions require less energy than those emitted at 1550 nm because from the emissions of 1400 nm, the water in the atmosphere begins to absorb energy [39].

Objects recognized by lidar sensors can be categorized, for example, a pedestrian can be distinguished from a cyclist [35]. Due to the diffraction of light in adverse weather conditions, such as rain, snow or fog, the operating range detection depends on the reflectivity of the objects that are reached by the laser beams [39].

Cameras

Cameras are capable of producing images of the surrounding world by capturing the electromagnetic waves emitted by objects [35] and converting them into electrical signals [38]. The light emitted by objects is captured by an electronic image sensor inside the cameras and they are classified into Charge-Coupled Device (CCD) sensors and Complementary Metal Oxide Semiconductor (CMOS) sensors. Typically, CCD sensors create higher quality images (with low noise [39]) and are the most used for vehicle applications [42].

Cameras capture wavelengths between 400 nm and 780 nm (visible light) and are divided into three bands: Red, Green and Blue (RGB) that are coded separately. A single camera has one camera lens and one image sensor and provides 2D images. The combination of two cameras results in a stereo camera, which consists of two camera lenses and two image sensors, and takes two images from different angles simultaneously. Stereo cameras (Figure 2.22) can have two or more lenses with an image sensor for each lens and provide depth information by creating 3D images, so stereo cameras with this feature are known as RGBD [39]. For detecting vehicles on the road, both single camera and stereo camera are used [35].

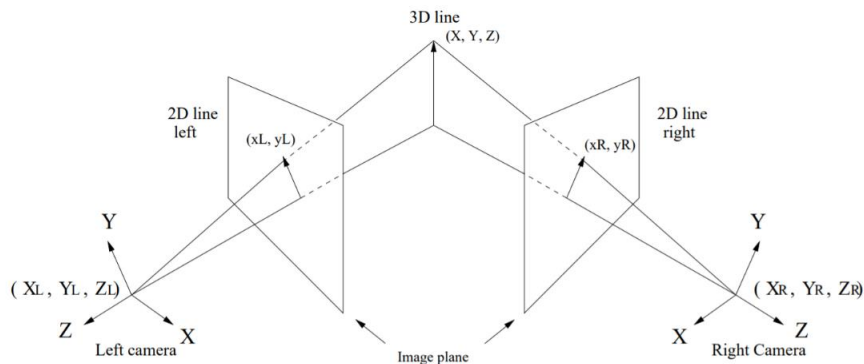


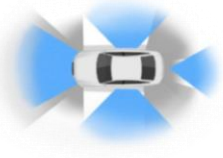
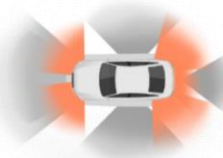

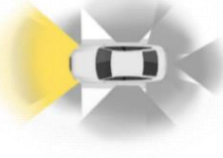
Figure 2.22 - Stereo cameras for creating 3D images [43].

In addition to the strong color processing capabilities, cameras can also provide texture and contrast data, making these sensors very efficient in classifying data [38]. However, cameras do not provide distance information and their reliability is limited in adverse environmental conditions, such as snow, ice or fog, and are highly affected by variations in lighting, such as in the dark [39]. Besides that, as autonomous vehicles use high-definition cameras, powerful processors are also required to process millions of pixels for each frame - some cameras shoot at 30 to 60 frames per second - and the price of these cameras becomes very expensive [38].

Autonomous vehicles also use infrared cameras that capture infrared light with wavelengths between 780 nm and 1 mm [39]. These cameras are mostly used for pedestrian detection at night [35].

Vehicles use these different sensors for different ADAS applications, summarized in Table 2.5.

Table 2.5 - Usage and applications of vehicle sensors.

Sensor type	Typical usage	ADAS Applications	Illustration
Cameras	Obstacle detection and classification, and 3D mapping.	Lane departure warning system[42]; Parking assistance.	
Radar	Obstacle detection and speed obstacle measurement.	Blind-spot detection [38]; Lane Change Assistant [39]; Forward Cross Traffic Alert [38]; Auto parking [35]. Adaptive cruise control [38]; Emergency brake assistant [35];	
Ultrasonic sensors	Near obstacle detection.	Parking assistance [39]; Blind-spot detection (for emergency brake assistants).	
Lidar	Obstacle detection and 3D mapping.	Adaptive Cruise Control [39];	

2) Vehicle's navigation systems

One of the most important subsystems of autonomous and connected vehicles is the navigation and guidance system, where the location of the vehicle is the key. Unlike the sensors described in the previous section, the vehicle's navigation systems provide the vehicle's absolute location.

Today, one of the most popular ways to locate a vehicle is through the fusion of satellite-based navigation systems and inertial navigation systems [36]. With their combination, each system can complement each other and overcome the shortcomings of the individual system [44].

Satellite-based navigation systems

The general term for satellite-based technology that provides global positioning, navigation and timing services for autonomous vehicles is the Global Navigation Satellite System (GNSS).

GNSS is a system that uses satellites to provide geo-spatial positioning, allowing electronic receivers to determine their location (longitude, latitude, and altitude) with high precision using time signals (radio signal) from a set of satellites that orbit approximately 20,000 km from the earth's surface [39]. From these signals it is possible to determine the exact location of the vehicle, which combined with the time information provided by the satellites can also accurately calculate the vehicle's speed [38].

Global Positioning System (GPS) is the most prevalent GNSS developed by the USA and uses information from a network of 24 satellites placed in orbit located on six fixed planes inclined 55° from the equator with a rotation period of 11 h 58 m (Figure 2.23). The configuration of the 24 satellites allows any receiver located on the earth's surface to receive signals from 6 to 12 satellites [39].

	GPS	GLONASS	GALILEO	BEIDOU
Satellites	24	24	30	30 + 5*
Precision	7.8 m, civil 5.9 m, military	7.4 m, civil 4.5 m, military	1.0 m, civil 0.01 m, advantage	10 m, civil 0.1 m, military
Coverage	Global	Global	Global	Chinese
Period	11 h 58 m	11 h 15m	14 h	12h 53m
height	26650 Km	19100 Km	23222 Km	21150 Km
Owner	EEUU	Russia	European Union	China

* Geostationary Satellite.

Figure 2.23 - Most commonly used GNSS [39].

GPS positioning is based on trilateration (Figure 2.24). If there is only one satellite, the receiver can be anywhere along the circle; if there are two satellites, the receiver can

be at any of the two points where the two circles intersect; if there are three satellites, the receiver can only be positioned where the three circles intersect. The vehicle's current position is calculated based on the analysis of the signals received from at least 4 satellites [38].

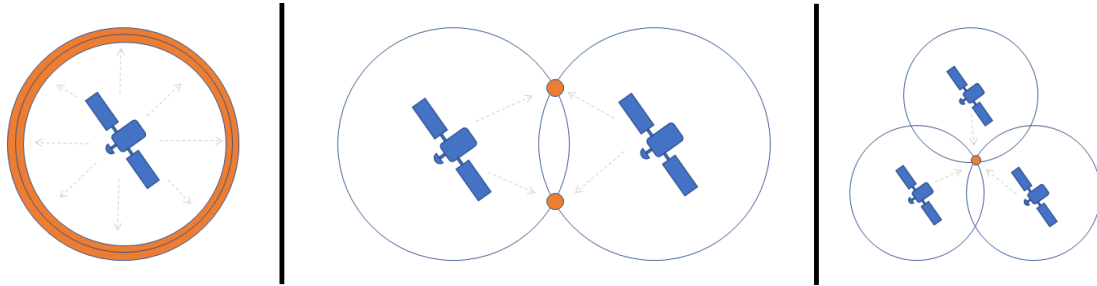


Figure 2.24 - Principle of trilateration¹⁰.

The GPS operating principle is based on measuring the time of flight of the signal emitted by the satellite and that received by the receiver to calculate the distance (or range) from the receiver to the satellite [39]. As the GPS receiver does not carry atomic clocks (like the GPS satellites), the measured distances between the receiver and the satellites introduce errors originating from the clock error. In this way, the distance between a receiver P and the i -th satellite (S_i) is called pseudo range (PR_i) and is determined as follows [44]:

$$PR_i = R_i + C * \Delta t_{Ai} + C * (\Delta t_u - \Delta t_{Si}) \quad (2.2)$$

Where $i = 1, 2, 3, 4$; R_i is the real distance between the i -th satellite to the receiver P ; C is the speed of light; Δt_{Ai} is the i -th satellite's transmission (radio wave propagation) delay and other errors that can occur, Δt_u is the receiver clock's errors relative to GPS system time and Δt_{Si} is the i -th satellite's error relative to GPS system time.

Assuming that the receiver position is $P(X, Y, Z)$ and the position of the i -th satellite is $S_i(X_i, Y_i, Z_i)$, which is known, the real distance between them is

$$R_i = \sqrt{(X_i - X)^2 + (Y_i - Y)^2 + (Z_i - Z)^2} \quad (2.3)$$

¹⁰ https://www.mathworks.com/help/fusion/gs/model-imu-gps-and-insgps.html?fbclid=IwARooTYj_HN3_LHmnZ-URfqT6sF1NfiWyjUb7h1Oifn_oHcMKKY9UnpSJmbE

Then,

$$PR_i = \sqrt{(X_i - X)^2 + (Y_i - Y)^2 + (Z_i - Z)^2} + C * \Delta t_{Ai} + C * (\Delta t_u - \Delta t_{Si}) \quad (2.4)$$

The pseudo range (PR_i) is measured by the receiver and by linearizing equation (2.4) it is possible to get the receiver's position $P(X, Y, Z)$ – the vehicle's position.

Inertial navigation systems

The basis of the inertial navigation systems (INS) is the inertial measurement unit (IMU), which is an electronic device (Figure 2.2.5) mounted on a platform fixed to the vehicle [45]. The basic concept behind an inertial navigation systems is the measurement of changes in relative motion, which requires two components: a combination of individual inertial sensors and a starting position [46].

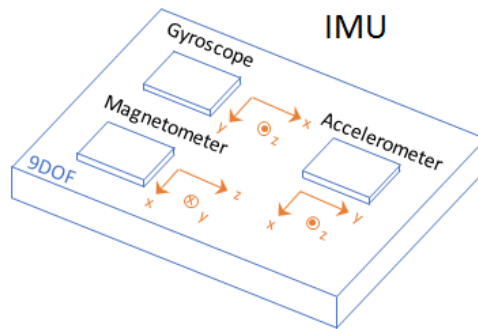


Figure 2.2.5 - Inertial Measurement Unit¹⁰.

Inertial sensors include accelerometers, gyroscopes and magnetometers (each oriented towards the orthogonal , X, Y and Z axes) which measures vehicle's linear acceleration, angular velocity and magnetic field intensity data, respectively. The data reported by the IMU is fed into a processor which calculates vehicle's relative velocity and position [47].

As inertial sensors can only make measurements when detecting changes between states, a starting position (initial location of the vehicle) is required. Therefore, a state S_0 is defined in the starting position as the inertial reference and all changes are determined between the starting position and the current state [46]. The starting position is usually determined by GPS. Besides that, an IMU allows a GPS receiver to

work when GPS-signals are unavailable (e.g., in tunnels, inside buildings, with electronic interference).

Unlike the other vehicle sensors, an IMU does not require a connection or knowledge of the external world to provide data to the vehicle's software for perception and localization.

B. Planning system

All perception data with the location of obstacles are fed into the vehicle's system for motion planning. The planning system of an autonomous vehicle usually consists of three software steps: the route planning, the behavioral planning and trajectory planning, as illustrated in Figure 2.26.



Figure 2.26 - Planning system.

Route planning

The first step is the searching for a feasible path. A route planner (or algorithm) is responsible for choosing an optimal global route - taking into account high-level parameters such as energy efficiency, traffic flows and travel time - from the vehicle's current position to the requested destination. This task needs to access the external map to obtain the road network and road information, and uses global optimization algorithms to find the optimal paths for the vehicle [35]. The classic algorithms for route planning are graph search algorithms, which can find the shortest path from a starting node to a destination node in the search space [48]. The route planners define the optimal route on the road network map, by discretizing it into waypoints (sets of coordinates) [49]. Once the global route plan has been found, the vehicle's architecture goes to the decision-making task [35].

Behavioral planning

The behavioral planner is responsible for decision making and is based on motion prediction techniques. The goals of these planner are:

- to determine a collision-free trajectory that the vehicle should follow based on the outputs from the environment perception [50];
- to ensure that the vehicle follows any stipulated road rules while making incremental progress along the planned route [36].

Decision making consists of the vehicle driving behavior that maximizes the driving function within a dynamic environment, where the motion prediction is an essential element for this task. The existing motion prediction models are categorized into physics-based models, maneuver-based models and interaction-aware models (Table 2.6) [35].

The motion prediction task stores the current and historic dynamics data to predict the dynamics of all the elements surrounding the vehicle (Deep Reinforcement Learning [49]). This process allows to perform risk estimation and dynamic re-planning [51].

The output of the behavioral planner is usually a high-level characterization of motion (e.g., "going straight", "turning"), which is fed into the next task - the trajectory planning. The trajectory planning system translates the behavioral output into a dynamic feasible trajectory, where the path is parameterized by time [35].

Table 2.6 - Motion prediction models [52].

Physics-based motion models	Maneuver-based motion models	Interaction-aware motion models
These models consider that the motion of vehicles only depends on the laws of physics, which rely on dynamic and kinematic properties.	These models, in addition to considering the laws of physics, also consider that the future movement of a vehicle depends on the maneuver that the driver intends to perform.	These models consider that the motion of the vehicles is influenced by the motion of the other vehicles in the scene, taking into account the interdependencies between vehicle maneuvers.
Provide only short-term (less than a second) motion and risk estimation.	Provide long-term motion and risk estimation, but are not always reliable since they ignore the dependencies between the vehicles in the scene.	Provide reliable long-term motion and risk estimation, however, due to their computational complexity, they are not always compatible with real-time risk assessment.

Low level of abstraction

High level of abstraction

Trajectory planning

The final step in the planning process is the trajectory generation, which is the real-time planning of a vehicle's driving function, where trajectory planning algorithms

take the given path defined by the previous layers (reference path) and endow it with the time information, as well as velocity or acceleration and other position derivatives [35].

Including time as a dimension in the configuration space is crucial for an accurate determination of the vehicle's trajectory and obstacle trajectories [36]. These planners process the data from the waypoints on the collision-free geometric path (created by the route planners) in order to associate a velocity for each waypoint while preserving the geometric properties of the continuous path [53].

In the literature, the most relevant path planning algorithms implemented are classified into three groups: graph search based planners, sampling-based planners and interpolating curve planners [54].

Graph search based planners are algorithms that find the shortest path in a graph. This state space is often represented as a discrete occupancy grid or lattice that depicts where objects are in the environment. The most used in motion planning are the Dijkstra's algorithm, shown in Figure 2.27, and the A* Algorithm, which is an extension of the Dijkstra's algorithm with heuristics, which enables a fast node search [54].

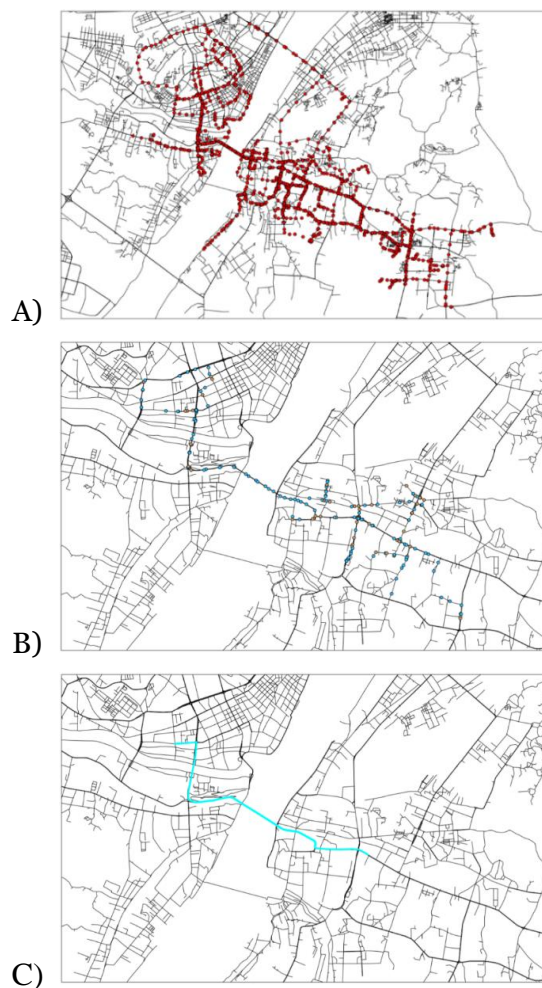


Figure 2.27 - Dijkstra's algorithm: A) the red points are the way-points planned by the route planner, B) the blue are matched points and the brown are interpolated points, C) generated trajectory [55].

The sampling-based planners consists of connecting points sampled randomly in the configuration space in order to build a graph (roadmap) of feasible trajectories, collision-free trajectories. These algorithms find the shortest path that connects the initial state with a final state through the roadmap. The most commonly used algorithm is the Rapidly-exploring Random Tree (RRT) [54] - illustrated in Figure 2.28.

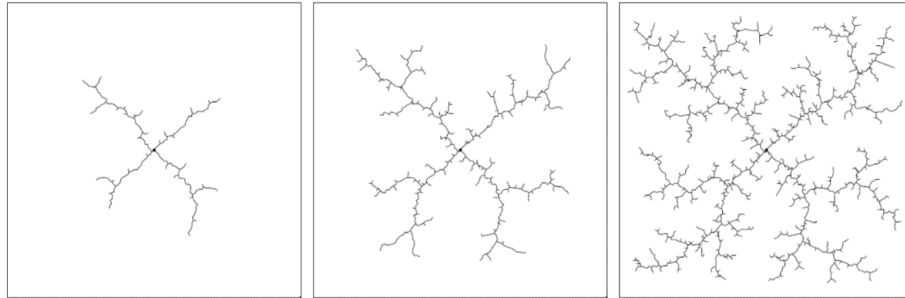


Figure 2.28 - RRT search with increasing iterations [53].

Unlike graph search planners and sampling-based planners that are global planners that provide a rough approximation of the solution, the interpolating curve planners interpolate the waypoint list using geometric functions. The main goal of these planners is to smooth the path from a given set of waypoints. Interpolating curve planners are represented by a specific geometric function such as the clothoids curves, the spline curves, the polynomial curves and the Bézier curves (Figure 2.29) [53].

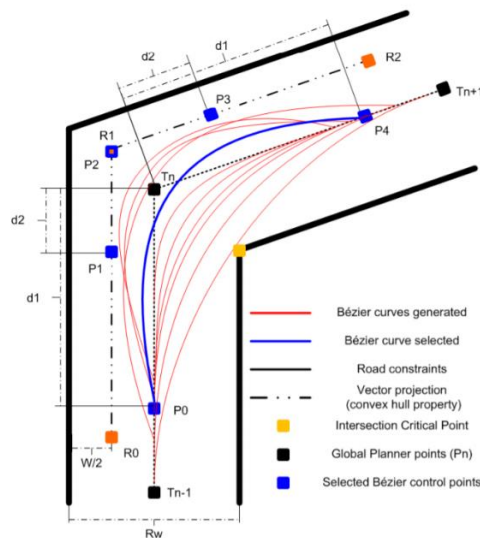


Figure 2.29 - Bézier curve for the trajectory generation - the blue curve is the optimal trajectory for the vehicle to follow [56].

Once the path with the waypoints is generated, this information is passed to the vehicle's control system so the planned trajectory can be executed by the actuators.

C. Control system

Motion control is the process of converting planned intentions into actions. The control system physically performs the actions outlined by the planning system by tracking the list of waypoints (path tracking) and target velocities (speed tracking) and by controlling the vehicle at the hardware level to follow the planned trajectory, based on sensor readings [35].

The control system passes the waypoints and velocities to an algorithm, called controller, which calculates how much steering, acceleration or braking is required to ensure the vehicle longitudinal and lateral motion [35].

In literature, control strategies for trajectory tracking can be classified into traditional control and learning-based control (Figure 2.30). Along with that, some control strategies can be achieved trajectory tracking directly, while others achieve trajectory tracking through path tracking and speed tracking, separately.

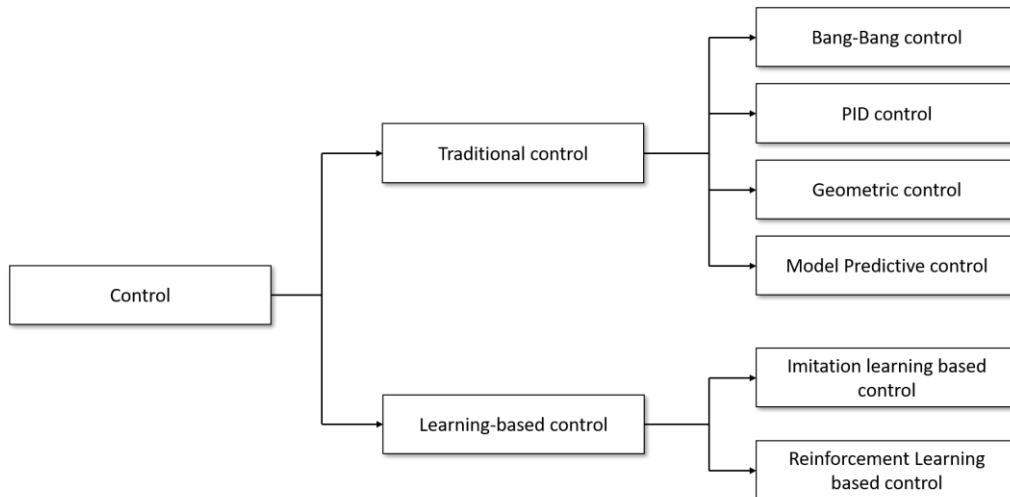


Figure 2.30 - Control Strategies for Autonomous Vehicles.

Traditional control is one of the most widely used and reliable control strategy for safety-critical systems, such as autonomous vehicles, while learning-based control strategies have only just begun to emerge. Learning-based control schemes are a promising alternative to traditional ones, however, these schemes still have a long way to achieve the same goals as the traditional control strategies, since learning-based controllers are not entirely reliable - when presented with a new scenario (not stored in database), they can generate wrong control actions based on the resources learned. Hybrid control strategies are also possible. Aspects such as gains or systems models of traditional controllers can be learned through recursive training, thereby improving the performance of the controller [57].

The Bang-Bang control switches abruptly between two states (simple binary controller) and is recommended to be applied only to Variable Structure Systems (VSS) that allow Sliding Motion Control (SMC) [57]. Sliding motion control is a nonlinear control method for nonlinear dynamical systems, such as AVs, and is widely applied to trajectory tracking to deal with the disturbances, like the crosswind, varying vehicle parameter and changing road friction [35].

The PID control is used for trajectory tracking and takes advantage of the three primary controllers - the proportional controller, the integral controller and derivative controller – to rise to a much more efficient controller [57] for both vehicle's lateral and longitudinal control [35].

Geometric control is used for path tracking and the most popular controllers are the Pure Pursuit controller and the Stanley controller [57]. As these controllers ignore the vehicle's velocity and acceleration (speed tracking), they are unable to achieve good high-speed tracking performance [35].

Model Predictive Control (MPC) is an optimization control strategy for trajectory tracking [57] and has the capability to systematically include system constraints and future predictions in the controller [35]. MPC controllers predict the future states of the vehicle, considering control constraints, in order to select the optimal solution – the optimal set of waypoints by minimizing a cost function (minimal cost trajectory) [57]. Control restrictions are used to increase the vehicle's fuel economy and the safety performance [36]. In this way, MPC has become the most attractive approach in the control of autonomous vehicles [35].

Imitation learning and Reinforcement learning are also applied to trajectory tracking control. Imitation learning is a type of supervised learning that uses a set of labeled data to train the system in order to directly predict the control actions required to drive the vehicle, minimizing the error in its predictions. However, achieving perfection by using this technique requires especially a very large dataset and an extended training duration. In contrast, reinforcement learning learns through trial-and-error situations, which allows the system to explore the environment and discover new strategies on its own - where some of them may be even better than those performed by humans. When the system performs a set of well-done actions, it is rewarded and, therefore, the purpose of this learning is to maximize the reward function. However, reward functions are difficult to define and as the system seeks to maximize the reward function, it can take any possible action, including cheating, which can compromise road safety [57].

2.2 Connected vehicles

Connected Vehicles (CVs) use various communication technologies to exchange information. Their connectivity refers to an Intelligent Transport System (ITS) where all vehicles and infrastructure systems are interconnected with each other. The communication mechanisms are collectively known as Vehicle-To-Everything (V2X) communication, which includes the following subcategories: Vehicle-To-Vehicle (V2V) communication, Vehicle-To-Infrastructure (V2I) communication, Vehicle-To-Pedestrian (V2P) communication and Vehicle-To-Network (V2N) communications, as well as Vehicle-To-Grid (V2G) communication and Vehicle-To-Home (V2H) communication.

All of this interconnectivity has become possible with the introduction of Vehicular Ad hoc Networks (VANETs). VANETs are a new type of Mobile Ad hoc Networks (MANETs), where all the nodes in the network are moving or stationary vehicles connected by wireless using dedicated short-range communication between them. This intelligent transport system enables a wide range of road applications, such as prevention of collisions, safety, blind crossing, dynamic route scheduling, real-time traffic condition monitoring, etc. [58], as well as providing Internet connectivity to the vehicular nodes [58].

According to the National Highway Traffic Safety Administration (NHSTA) predictions, by effectively applying V2V and V2I communications it is possible to reduce and/or eliminate up to 80% crashes of any type from non-impairment [59]. In addition to reducing the number of accidents, V2X connectivity and its constituents allow other benefits on the roads, such as reducing congestion, by optimizing traffic flows, and minimizing vehicle emissions.

2.2.1 Communication networks

A. Mobile Ad Hoc Networks

Mobile ad hoc networks are networks composed of a set of mobile devices (or nodes) spontaneously interconnected with wireless links.

By default, a node in an ad hoc network is only able to communicate with its neighbors (single-hop), restricting communications to the node's radio range. To solve this problem, MANETs use specially designed routing protocols that provide the network with multi-hop communication. Therefore, a path between two nodes (source node and destination node) is a sequence of an arbitrary number of hops of intermediate nodes, as shown in Figure 2.31. Nodes are autonomous and have the ability of organizing

themselves randomly, and the success of interaction extremely relies on other nodes collaboration. Each node has to execute routing by forwarding messages for other nodes. Besides that, nodes can be router and host at the same time [60].

MANETs are dynamic networks where nodes move freely, that is, the nodes do not have fixed positions, they are free to move inside and outside the network. In addition, MANETs are completely self-organized networks that have the ability to work anywhere without any pre-existing infrastructure, such as routers in wired networks, so they can be established anywhere without any geographical restrictions [60].

Another feature of MANETs is devices heterogeneity, where the network works regardless of the types of devices that constitute it.

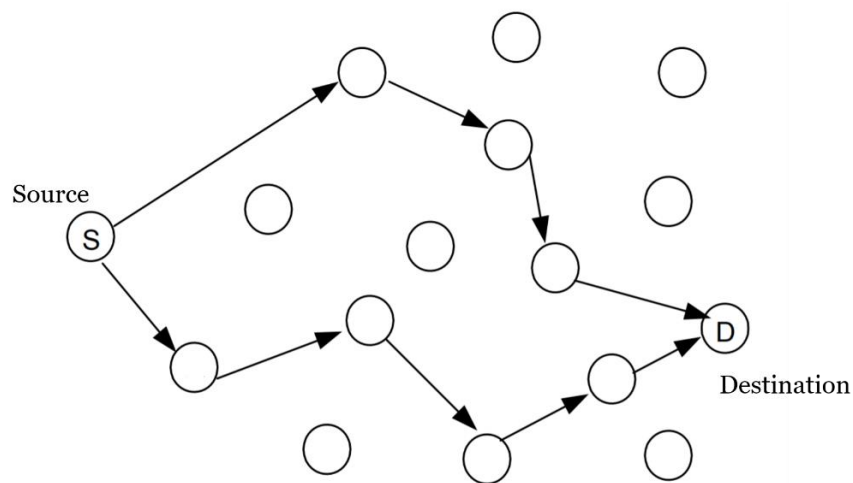


Figure 2.31 - Multi-hop routing, using dual paths, between the source node and the destination node (adapted from [61]).

B. Vehicular Ad Hoc Networks

Vehicle ad hoc networks are MANETs where the nodes in the network are vehicles. Vehicles can communicate with each other, or with Roadside Units (RSU), by transmitting a Basic Safety Message (BSM). A BSM is a package of data that contains information about vehicle position, speed and other information relating to a vehicle's state and predicted path to its destination [62]. A BSM is updated and broadcasted up to 10 times per second to surrounding vehicles [63]. A roadside unit is a static access point (or a communication node) installed within infrastructures located alongside the roads. RSUs have the ability to enable communications between vehicles and an infrastructure [64].

1. Data dissemination in VANETs

Dissemination of data is a scalable process because the number of broadcasted messages is limited and, therefore, the network is not flooded [65]. In literature, data dissemination techniques for VANETs can be classified into three models: push model, pull model, and hybrid model. Push models disseminate data proactively using periodic broadcast, while pull models disseminate data on demand. Hybrid models combine both models for data dissemination in order to support different applications.

Push model

Push models are preferred for safety applications where an immediate response is required. The purpose here is to regularly exchange information between vehicles in motion, in order to allow each individual vehicle to view and assess traffic conditions ahead of it [65]. Thus, these models are used for safety messaging systems, such as collision warning systems, emergency message dissemination systems and information systems specified for dangerous road conditions (e.g., ice, water, snow) [66].

Vehicles generate data with their information, such as their position, which are updated in every broadcast period, and they store other vehicles data whenever they receive a broadcasted message. Each vehicle broadcasts the information about itself and the information it knows about other vehicles (relayed data) in a single package. Every time a vehicle receives information broadcasted by another vehicle, it updates its stored information - in this process, the received message is postponed to the next transmission period, where the information broadcasted by the vehicle is already the updated information [65].

Vehicular information dissemination can be broadcasted in all directions, that means that the communication could be broadcasted using vehicles traveling in the same direction, vehicles traveling in the opposite direction, or vehicles traveling in both directions (bidirectional mobility). Figure 2.32 illustrates the model for data dissemination for vehicles that travel in the same direction and in opposite direction, separately.

In the same-direction model, when a vehicle broadcasts a package, only vehicles that move in the same direction, that are in the transmission range and that are behind the vehicle are responsible for the propagation of this package (propagated backwards). In the opposite-direction model, generated and relayed data are not broadcasted together. Vehicles traveling in the same direction broadcast only their own generated data. This data is then aggregated and propagated backwards by vehicles moving in the opposite direction [65].

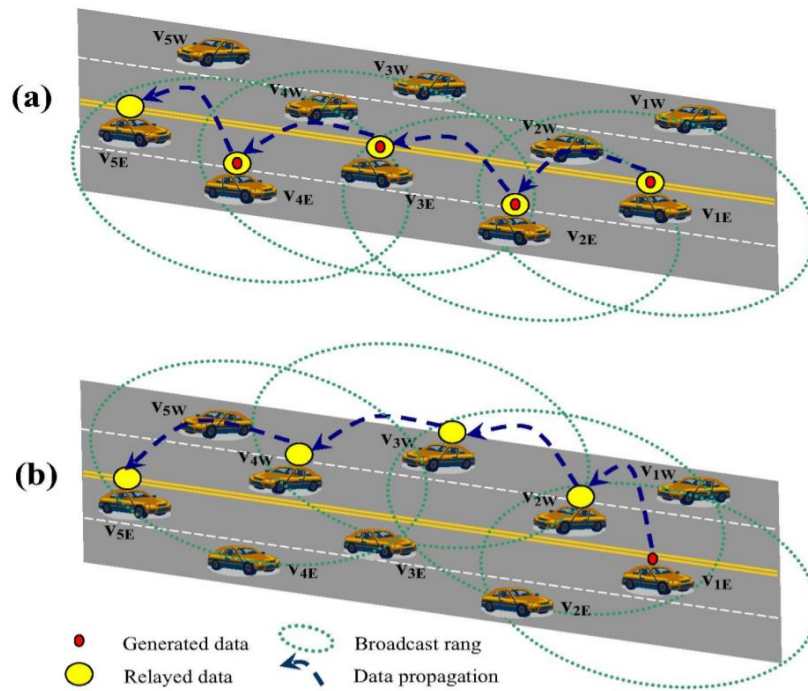


Figure 2.32 - Dissemination models: (a) the same-direction model, and (b) the opposite-direction model [65].

The bidirectional model combines both the same-direction model and the opposite-direction model. In this model, vehicles moving in the same direction propagate backwards vehicles' generated and relayed data, while vehicles moving in opposite direction only propagate relayed data [65]. This model, however, affects the performance of the data dissemination model because all vehicles traveling in the desired direction (both directions) participate in broadcasting, although it may be sufficient to transmit data for only a subset of the vehicles [66].

Pull model

Pull models, on the other hand, follow the request-response paradigm for data dissemination, therefore, they are used for delay-tolerant applications with the aim of improving traffic efficiency and travel comfort [66].

Usually, when a vehicle needs a service, it creates a data package with the service request containing its specifications, such as the type of service and the desired service area, and transmits it to the nearest RSU. According to what it proactively learned, the roadside unit does two things [66]:

- If the RSU knows the IP address of the provider of that service, it forwards the service request to the target service provider,
- If the RSU does not know the IP address, it broadcasts the service request over the backbone network to search for the target service provider. The RSU can transmit the service request to the target service provider over the vehicular network or the backbone network, however, if the RSU transmitted it over the vehicular network it would rapidly flood the network with data packages.

After receiving the request, the service provider creates a data package carrying the requested content (service response) and transmits it to the original vehicle [66].

Comparing to push models, pull models requires less overhead, since the number of data requests are smaller, tolerate more delays, as long as a response eventually returns [66], and have a better control in terms of bandwidth [67].

Hybrid model

There are some schemes that combine both models in order to support different types of applications, where for dangerous traffic conditions and emergency messages, push models are used and for location-sensitive queries issued by vehicles on demand, pull models are used [66].

2. VANETs characteristics

The main goal of VANETs is to improve road safety, convenience and comfort of the passengers in the vehicles [68]. The deploying of VANETs leads to enhance traffic safety and efficiency by reducing the traffic jams and accidents. This is possible to VANETs unique features, represented in Figure 2.33.

These vehicular networks have high mobility and therefore, have a high dynamic topology. The nodes of VANETs (vehicles) are more dynamic than the nodes of the typical MANETs because they are usually moving at a very high speed and changing their position constantly, making hard to predict a vehicle's position [68].

As the position of the nodes changes frequently, the network topology in VANETs also tends to change frequently, this is because the topology of the networks relies on the radio range between vehicles [68]. Due to the high movement of the nodes and the frequent change in the environment, VANETs also have frequent disconnections in the link connection between the vehicles, mostly in low traffic environments [69].

In this way, these networks have a variable network density based on the traffic density – in a case of traffic jam, the network density is very high and, in a case of a

suburban traffic, the network density can be very low. This means that the scale of the network in VANETs can reach unlimited geographic levels.

In VANETs, position measurements and energy consumption are not an issue. The vehicles' current position, speed and direction can be easily accurate. Given the number of on-board sensors in the vehicles that make continuous measurements and the number of computational resources (e.g., processors, large memory capacity) inside the vehicles, it is possible to obtain routing information. Besides that, vehicles have rich resources of power since they have the ability to provide continuous power via long-life batteries.

Another characteristic of VANETs is the predictable mobility patterns. VANETs are more capable of predicting mobility patterns than regular MANETs, since all vehicles move on pre-defined roads and highways [69] constrained by roads, streets and highway's structure, traffic lights, road signs, speed limit, traffic conditions, and drivers' driving behaviors [70].

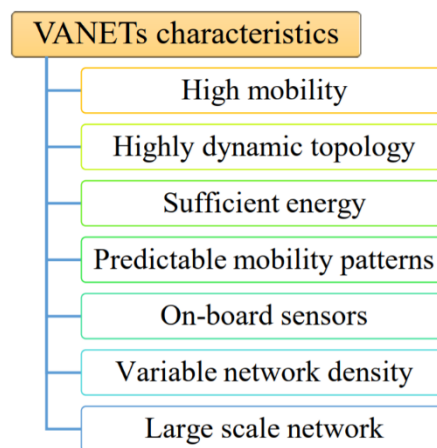


Figure 2.33 - General characteristics of VANETs [68].

3. Communication standards of VANETs

There are two types of radio technologies that are competing to become the standard for V2X connectivity: the Dedicated Short Range Communication (DSRC) and the Cellular V2X (C-V2X). These technologies are the pathway that allows vehicles to communicate with each other and with infrastructures.

Dedicated short range communication is a type of wireless technology that uses radio frequencies in the 5.9 GHz band, based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11p. The IEEE 802.11p is the IEEE standard for DSRC and is an amendment to the IEEE 802.11 standard to add wireless access in vehicular environments [71]. DSRC was introduced to add intelligence to transportation systems for interchanging wireless broadcast messages in a vehicular environment. It is a mature

technology with proven road-tested experience and is widely used in most places around the world. DSRC is a wireless communication method with a limited range that covers a maximum of ≈ 150 m (in all weather conditions) [59] and an end-to-end latency of 1–10 ms. This implies that the speed of the vehicle has to be relatively low, in order to allow enough time for the exchange of messages between vehicles and any roadside device. DSRC also have some complications during intensive traffic. As in these situations the number of devices connected using the same radio channels increases, the signal interference and the transmission delay also increase. As a result, the data transmission rate decreases and device authentication may not be handled properly or not be handled at all [72].

A promising alternative to DSRC is the cellular based V2X communication. C-V2X is defined by 3rd Generation Partnership Project (3GPP) and was first specified as part of the 3GPP, release 14, in 2017 using fourth generation wireless technology/Long-Term Evolution (4G/LTE) [71]. However, 4G C-V2X did not provide higher performance and lower latency than DSRC, where simulation results showed that when 50 vehicles are present, 4G C-V2X is actually worse than that of DSRC. However, 4G was improved and the fifth-Generation New Radio (5G NR), release 15 and 16, is a new promising possibility. 5G NR C-V2X offers higher performance, higher reliability, longer range, and reduced latency (operates at higher vehicular speeds). As 5G NR C-V2X is in development, it is important to test, verify, and improve the performance when they become available [59].

2.2.2 Types of V2X connectivity

There are several forms to exchange information in VANETs and each subcategory of V2X has specific concerns, strengths, weaknesses and different applications in the automotive field. Bringing together all of these subcategories takes advantage of the synergy between them. Figure 2.34 represents all subcategories of V2X connectivity.

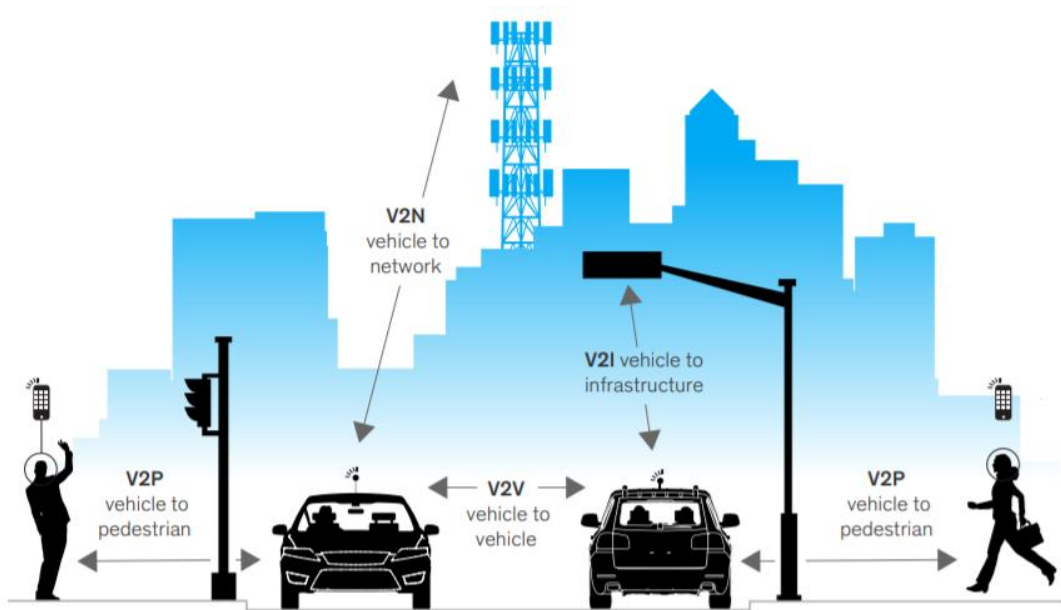


Figure 2.34 - V2X connectivity [71].

A. Vehicle-to-Vehicle

V2V technologies allows the direct vehicular communication, exchanging data between vehicles without relying on a fixed infrastructure [70]. Vehicles can share their speed and location, as well as any other relevant information between them, giving the system a 360-degree representation of its surroundings. This means that vehicles can issue warnings, avoid collisions or share immediate road and traffic conditions [71].

B. Vehicle-to-Infrastructure

V2I technologies allows a vehicle to communicate with the roadside infrastructures, exchanging data between vehicles and road infrastructures, such as traffic lights, road signs, and other transport infrastructure to strengthen safety measures [70]. For instance, dynamic traffic signaling can alert vehicles so that they can adjust its speed [71].

C. Vehicle-to-Pedestrian

V2P technologies allows data exchanging between vehicles and the electronic devices (e.g., smartphones) carried by pedestrians to ensure their safety. This enable both vehicles and pedestrians to receive information about activities taking place nearby [70]. For example, such devices can alert vehicles that a pedestrian is walking on the pedestrian walkway ahead, close to a crosswalk [71].

D. Vehicle-to-Network

V2N technologies allows data exchanging between vehicles and the Traffic Control Centre (TCC), receiving real-time information on traffic and weather conditions, as well as real-time custom navigation, and other cloud services [73].

E. Vehicle-to-Grid and Vehicle-to-Home communication

With the introduction of electric vehicles, V2G communication is beginning to flourish in the automotive industry. V2G technologies consist of bidirectional energy flow: from vehicle to grid, if the energy stored in the battery is high, and from grid to vehicle when the energy stored in the battery is low, as illustrated by Figure 2.35. However, if the energy of the electric vehicle battery is supplied to individual houses instead of the grid, it is called as V2H technology. The structure of V2H technology is similar to the V2G structure, in which the energy stored in batteries of the electric vehicles can be used as an energy source for houses. In this way, during the night, when power consumption is low on the grid, the vehicle's battery can be charged. When energy consumption is high in the grid, the vehicle's stored energy can be sold to the grid [74].

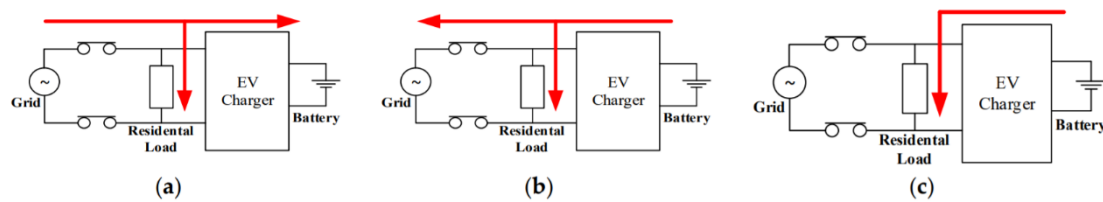


Figure 2.35 - Bidirectional electric vehicle charger: a) grid to vehicle mode; b) vehicle to grid mode; and c) Vehicle-to-Home [74].

Chapter 3

Verification, validation and testing of connected and autonomous vehicles

Connected and autonomous vehicles require large-scale development and testing in all possible scenarios before they can be deployed. Validating their systems through traditional methods requires driving hundreds of millions of miles, and in some cases hundreds of billions of miles, in the real world over the course of several decades to demonstrate their reliability, especially in terms of their safety on the road. However, these testing approaches are time- and capital-intensive.

Simulations, on the other hand, play an essential role in the testing and validation of CAVs, where it is possible to test drive billions of miles carried out in a virtual environment quickly and economically. Simulations are very close to real road testing and are based on the V-model, which is specifically designed to fulfil the automotive hardware and software development lifecycle. This approach follows the concept of the X-in-the-loop simulation for vehicle validation before going to public roads with mixed traffic, which represents a safer and more efficient way than live testing [75]. Field tests, like test drives, will contribute with further validation insights, which derive from unexpected driving situations and retroactive effects under real driving conditions [76]. Figure 3.1 shows the different models that can be configured to simulate a driving scenario.

Testing and validation (through simulation) alone are insufficient to ensure the correctness of the system, this is where software verification plays an important role. Through the use of formal methods, verification proves or disproves the correctness of the system [77].

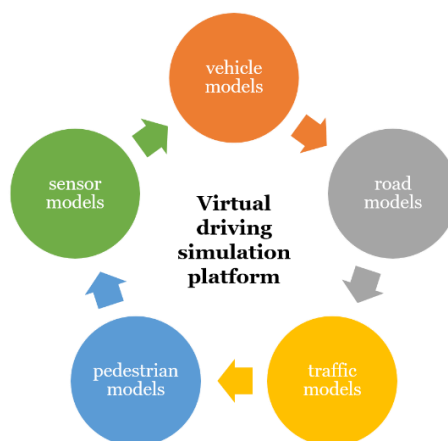


Figure 3.1 - Possible models to configure for testing and validation of CAVs.

3.1 Software development models

The Verification and Validation model (V-model) is the most commonly used model by the automotive industry for software development because CAVs are projects with well-defined and clear requirements and are built step by step, where the system parts are developed consecutively. However, some parts can be broken into smaller parts and be developed in phases in which agile models are used [78]. To decide which process model to use when developing a system, the following conditions must be considered: How stable are the requirements?, Who are the end users for the system?, What is the size of the project? and Where are the project teams located? [79].

3.1.1 V-model vs agile model

The V-model (Figure 3.2) is a sequential way of developing a system, in which each phase must be completed before the next phase begins. All phases have a corresponding verification and validation test - the components are tested and corrected immediately at each stage of the project and the test objectives are specific for each test level [79]. The entire development phase is planned in parallel with product testing, so software testers are involved from the beginning [78]. In this way, the fault-finding occurs at the early stage of the development process, which provides the cheapest alternative to fix it.

Due to its rigid nature, all requirements are written at the beginning of the process, so this model is used for projects where requirements rarely change during the system's development lifecycle. If one of the requirements changes, all the requirements and test documentation need to be updated [79]. At the end of the development process, the system is presented to stakeholders.

In contrast, the agile model (Figure 3.2) is a continuous process developed gradually in iterations [78]. In each iteration, requirements analysis, design, implementation, and testing are performed [79]. In this way, the agile model is more suitable for projects that must be delivered in a short time, since the development of the system is faster. The V-model requires more resources, especially more time, and is therefore more expensive.

Agile model has a flexible nature and is open to changes, which means that changing requirements can happen at any stage of the SDLC, even at the last stage of the iteration [78]. This is possible because at the end of each iteration, a product version is released to enable the constant feedback from the stakeholders. This allows the customer

to adaptively refine the requirements for the next released based on the observation of the evolving product [79]. The differences between the V-model and the agile model are summarized as shown in Table 3.1.

Table 3.1 - Characteristics between V-model and agile model.

V-model	Agile model
Sequential process	Continuous process
Rigid nature	Flexible nature
Requirements rarely or occasionally change	Requirements regularly change
Long or complex projects	Brief or simple projects

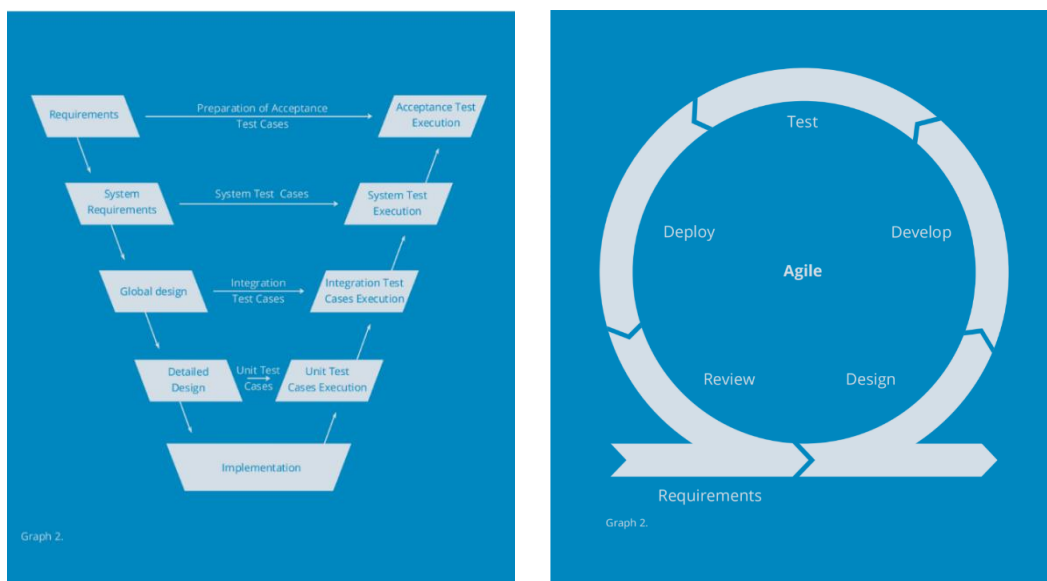


Figure 3.2 - V-model (left) and agile model (right) diagrams [78].

3.1.2 V-model

The traditional V-model is an extension of the waterfall model, but with an emphasis on verification and validation activities in the process of developing a system [80].

The waterfall model is a sequential development model that divides the SDLC into pre-defined phases (Figure 3.3). In this model, each phase must be completed (in a specified period of time) before the next phase can begin with no overlap between the phases. In this way, the system requirements must be specified in the start of the process because further changes in it will not be considered. Unlike the V-model, defects in the waterfall model are found very late in the development lifecycle, as the test phase only

happens later in the process and the test team has not been involved since the beginning of the project - the tester role is only involved in the testing phase [81]. Besides that, all errors that were not detected during the testing phase are only corrected in the maintenance phase. In this phase, functional improvements and software corrections are also implemented in order to allow its integration with other software.

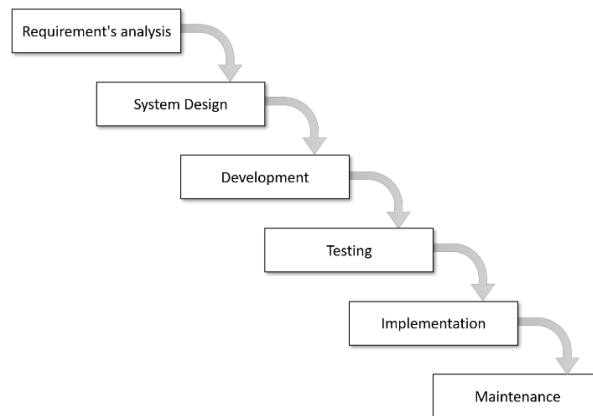


Figure 3.3 - Waterfall model (adapted from [81]).

The V-model starts in high-level architecture and requirements of the system to be designed and it decomposes in lower-level architectures where the development and implementation of the software and hardware takes place [82]. The V-model uses a ‘top-down’ approach to design and a ‘bottom-up’ approach to test and validation of safety-critical systems [83].

As illustrated in Figure 3.4, the SDLC of the V-model consists of those four phases, however, this work is focused on the right side of the “V” diagram – the testing phase (VVT) – which is divided into four test levels: Unit testing, Integration testing, System testing and Acceptance testing. The main objective of this phase is to test the system under development and its different levels of maturity, to ensure that the system is designed correctly, and all requirements are satisfied [80].

Unit tests (UT)

The main characteristic of this test level is that each module, unit, and component are tested in isolation and without interfacing with other components. Each software component, each sensor, the bus system and the ECUs are described in detail. The objective in this level is to check if the individual components meet the defined requirements and are working properly [84]. As no other components are involved it is much easier to find defects, reducing the number of errors in the basic functions, such as calculations of distances, time intervals, time for collision [85].

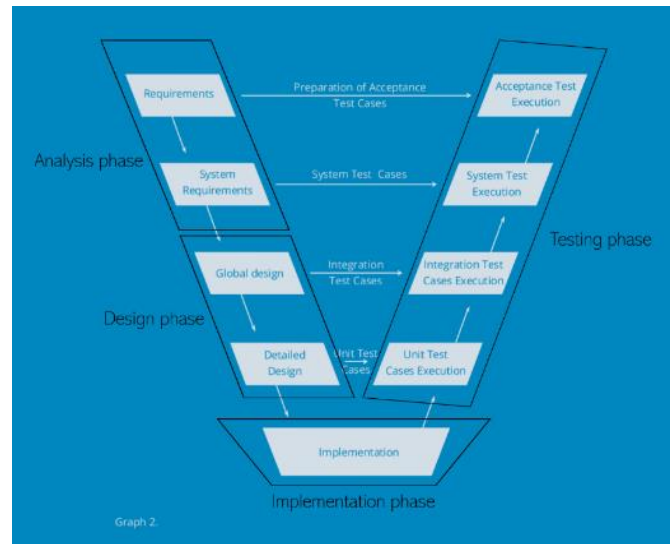


Figure 3.4 - V-model phases (adapted from [78]).

Integration tests (IT)

This test level is performed after the unit test is completed. Integration tests check whether the components, which have been developed and tested independently (in the unit tests), can be integrated with each other and function as a group – these are considered as sub-systems [86]. The objective in this level is to verify the interaction and functionality of the individual sub-systems and to verify if the subsystem meets the specifications. The sub-systems are modeled to represent different functionalities of the system under development: perception, processing and interpretation of data, and planning of path, maneuver and trajectory [84].

System tests (ST)

Although the components and the integration of the components (sub-systems) have already been tested at this stage, system tests are necessary because some features cannot be tested without running the complete software system [86]. The objective in this level is to check the behavior and the capacity of the entire system under development according to the defined functional and non-functional requirements through the test cases. Each test case contains all necessary information in order to be executable [84].

Acceptance tests (AccT)

The last test level of the V-model is performed to determine whether the system has met the requirement specifications and is ready to be used in the real world. The objective in this level is to test the acceptability of the system under development [86].

Test cases are designed to demonstrate the system behavior and evaluate the system from the user’s perspective by focusing on the external behavior of the system through field tests. Acceptance drives are set up as show cases, matching concrete scenarios carried out in real vehicles in the real world (on road testing) [84].

The main characteristics of the four test levels of the V-model can be summarized as shown in Table 3.2.

Table 3.2 - Test levels of the V-model.

Unit testing	Integration testing	System testing	Acceptance testing
First level	Second level	Third level	Fourth level
Test individual components	Test integrated components	Test the entire system	Test the entire final system
Finding errors is easy	Finding errors is difficult		
Done by developers	Done by the developers or testers	Done by testers	Done by End Users or developers

3.2 Verification, validation and testing approaches

Training and testing the machine learning algorithms of CAVs is crucial for them to be able to respond to all driving scenarios - SAE levels 1 and 2 tests were based on defined scenarios with specific maneuvers (depending on the function under test), while in SAE level 3+ test, the scenario space is infinite and requires many hours of virtual tests.

During the testing phase, the implemented system is examined through the software verification phase, to establish if the systems behavior is what is specified in the requirements set, and through the software validation phase, to establish if the system actually works as required by the customer. Thus, the approaches for VVT are divided in: verification approaches and test and validation approaches, as illustrated in Figure 3.4. Verification approaches are Formal-Based (FB) techniques for verifying the modeled system, where the main interests are the security properties. Test and validation approaches, on the other hand, are performed through two ways: (1) Model-Based (MB) and Ontology-Based (OB) methods to build vehicle models, and (2) SCenario-Based (ScB) and Search-Based (SB) methods to build the environment model. Ontology-based (OB) and search-based (SB) techniques are sub-categories of the MB and ScB testing, respectively [85].

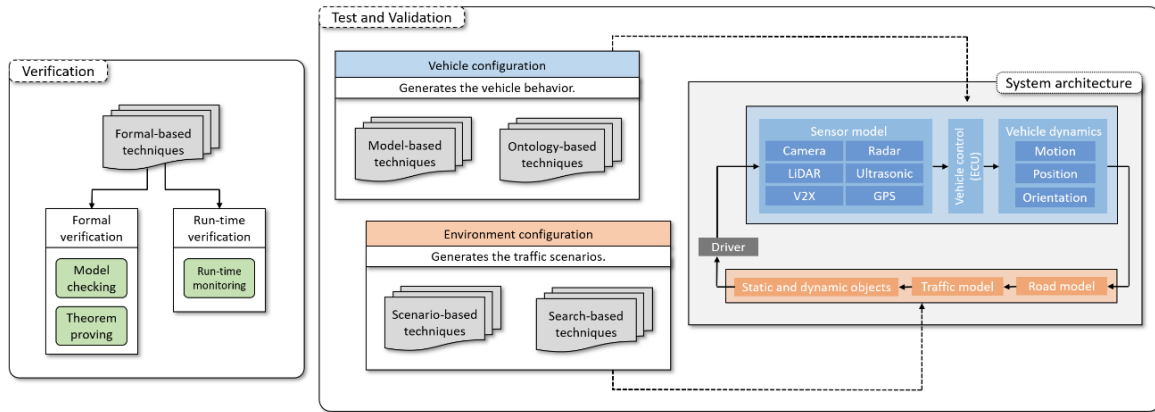


Figure 3.5 - Approaches for VVT and architecture of CAVs.

3.2.1 Formal-based techniques

Formal methods can be categorized into two ways according to their execution proceedings: static formal methods (Formal verification) and dynamic formal methods (Run-time verification).

Formal Verification (FV) is generally found in the literature through two techniques: Model Checking or Theorem Proving. Model checking is the primary technique used by FV tools to analyze the behavior of a sequential system over a period of time, it verifies whether a finite-state model of a system meets a given specification. This method is divided into three phases: the modeling of the system, the execution of the model checking algorithm, and the analysis of the results. Theorem-proving, on the other hand, requires expert knowledge in formal methods, and hence, is difficult and it is not so popular. The verification is viewed as a theorem to prove based on rules and axioms to find design and specification errors early, where the system properties are transformed into mathematical objects [26].

Run-time Verification (RV) is an area of formal methods and is usually performed using Run-time Monitoring. Run-time monitoring is based on observing executions of a system. Typically, the two main activities in run-time monitoring are the generation of a monitor from a specification and then the use of the monitor to analyze the dynamics of the system under study [86].

In addition, since system verification is carried out early in the development cycle, it helps to identify many critical bugs, making it cheaper to fix them at this level.

3.2.2 Model-based techniques

The adoption of model-based techniques for the development of ECU software has shown great advantages in productivity in the automotive industry. Thus, MB techniques are used to build vehicle models (vehicle software and hardware) and are found in the literature based on two main steps: the system modeling process and the verification and validation (V&V) process. The system modeling process consists of creating models to specify the expected behavior of the system under test (functions and their states, inputs and structure), and the verification and validation process follows the four test levels of the V-model to detect failures in the system as early as possible.

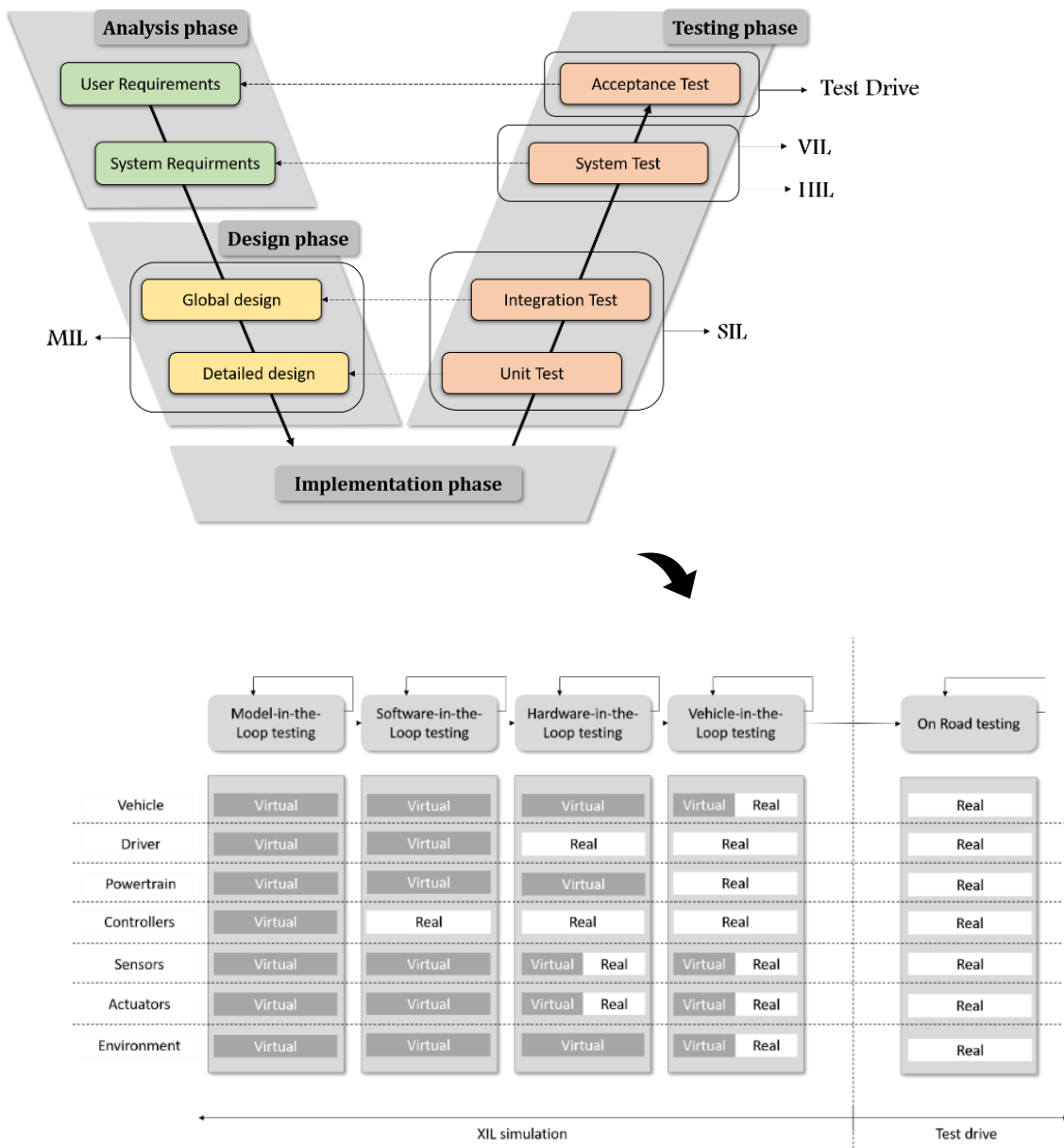


Figure 3.6 - XIL simulation and Test drive and the relationship between the virtual/real components.

In CAVs, these two processes are performed using X-In-the-Loop (XIL) simulations and Test drive (last test stage), as illustrated in Figure 3.5. These simulation loops are a key element to reduce the development time and cost of the verification and validation process (the testing phase) [87]. The XIL simulations are a model-based approach, where the “X” refers to any type of test included in the development process, such as the abstract model, software, or hardware. In other words, XIL simulation is the set of tests with different configurations, generally performed sequentially: Model-In-the-Loop (MIL), Software-In-the-Loop (SIL), Hardware-In-the-Loop (HIL) and the Vehicle-In-the-Loop (VIL) simulations.

Model-based testing is considered as "black box", since only its inputs and outputs are considered controllable and observable, respectively [88].

XIL Simulation

The role of simulation in CAVs saves years of testing that would be required on the road since those same miles are driven in a fraction of the time in a virtual simulation. Simulation provides an opportunity to adjust vehicle safety properties in a virtual environment before field trials.

MIL simulation

In the design phase, a model-in-the-loop is developed with the different modules, considering the different functionalities of a vehicle system [80]. MIL is an offline simulation and its goal is to abstract the behavior of the system in a way that the function model is tested to validate the system concept [4], [89]. From each of the different modules, the code is generated and tested through the UT and the IT using a SIL simulation.

SIL simulation

Software-in-the-loop simulation allows to test some driving software components in real time simulations [90]. This method involves linking all the AV algorithms that correspond to the hardware of a vehicle, that is, it allows developers to check the performance of the code in a simulated environment without real hardware parts. If the system passes the UT and the IT, the system test can be performed using first a HIL simulation, where the software is integrated into a real ECU, and then a VIL simulation, where the complete vehicle is tested. Otherwise, it returns horizontally to the design phase to modify its features and rebuild the unit test and the integration test [80].

HIL simulation

The hardware-in-the-loop simulation is a hybrid system, where real and virtual elements form a closed-loop virtual reality environment that allows the test (ST) of an automotive component or a complete autonomous driving system [90]. In this way, is one of the most important steps in the real-time simulation environments before any road testing. After the full integration of all electronic and mechanical components, the complete vehicle is tested using VIL simulation.

VIL simulation

With the vehicle-in-the-loop, real-time simulations are used to study human behavior inside a real car as it drives in virtual traffic either by itself or controlled by the driver when needed. On this level the entire vehicle is the system under test and this requires an integration of a real vehicle with the hardware-in-the-loop system [91]. This technique is recently used to simulate real vehicles in virtual environments, to ensure the safety of critical test scenarios in real-world tests.

If the vehicle system passes the ST, through HIL and VIL simulations, the next test level - the acceptance test - can be performed. Otherwise, at this level, it returns horizontally to system requirements for its modification. The testing phase ends with the AT, where the complete vehicle is tested in the real world through Test drives for validation.

Test drive

Road tests with prototype vehicles are the last stage of vehicle testing (carried out in the real world) and certify the results obtained in the XIL simulation (virtual world), where field monitoring in the simulated models is used for further verify and refine the system of CAVs with aggregated data collected from the field. Field trials are decisive tests to ensure these vehicles exhibit the intended behavior and to capture any potential violations of the safety requirements before an event of significant loss occurs. [92].

Field experience has shown that VVT applied to a given operational design domains structures the whole process. Standardized processes provide generic VVT processes and CAVs testing grounds, however, with field testing, datasets are created to satisfy ODD constraints instead of working generically, refining the test and validation of these vehicles [93]. Ensuring that training and testing are completed requires at least ensuring that all aspects of the ODD have been addressed [94].

Typical descriptions of an ODD tend to be somewhat simplistic as the relevant factors are road types, geographic features, speed ranges, weather and “other domain restrictions” (listed by NHTSA in 2017). The list of “other” considerations can be extensive and difficult to enumerate due to the variety of automated vehicle projects. [94]

3.2.3 Ontology-based techniques

The ontology-based approach is a model-based testing approach that instead of modeling the vehicle's system behavior, relies on existing vehicle system models to model the behavior of the system. In another words, ontology-based testing relies on a description of the environmental concepts and their relationships to extract inputs for a system under test. OB testing is a recent concept and was originally developed to identify critical scenarios - to test autonomous vehicles where interactions of environmental concepts (e.g., street junctions, weather conditions, pedestrians crossing the street) could cause a wrong behavior of the AV being tested [95], [96].

3.2.4 Scenario-based techniques

After the vehicle configuration (MB or OB approaches), the vehicle needs to be tested in driving scenarios, hence, scenario-based approaches are used. Scenario-based simulations specify an entire scenario (scenes, events, goals and values) in a test case and the scenario information is the input for the driving function. This information is usually reconstructed from crash data analysis and naturalistic driving data analysis for the construction of scenarios [92]. For the Scenario-based testing, two greatest difficulties are pointed out in the literature:

- the input data are not discrete, it occur in large quantities, change very quickly and depend on the environment [92],
- it is difficult to repeat the tests with the same input data, unless by simulation, since an infinite number of different scenarios can theoretically occur with the same input data [92].

3.2.5 Search-based techniques

Search-based approaches are used to find particularly dangerous situations - generally applied to collision avoidance systems and are typically vision-based systems using thousands of simulated scenarios [97]. Most of these techniques in the field of autonomous vehicles use evolutionary algorithms. These algorithms rely on machine learning or a combination of machine learning and Darwinian genetic operators to automatically generate new solutions (program inputs - test scenarios) [98], [99]. These evolutionary algorithms work through the iterative sampling of the input space, select the fittest scenarios (critical test scenarios) and evolve the fittest ones using genetic search operators to generate new (critical) scenarios. They are able to effectively generate the most critical test scenarios and provide useful results, regardless of specific time constraints and the size of the input space [100].

3.3 Verification, validation and testing tools

Due to the different processes in the verification, validation and testing of CAVs, the tools are categorized according to the software process they perform, as illustrated in Figure 3.7.

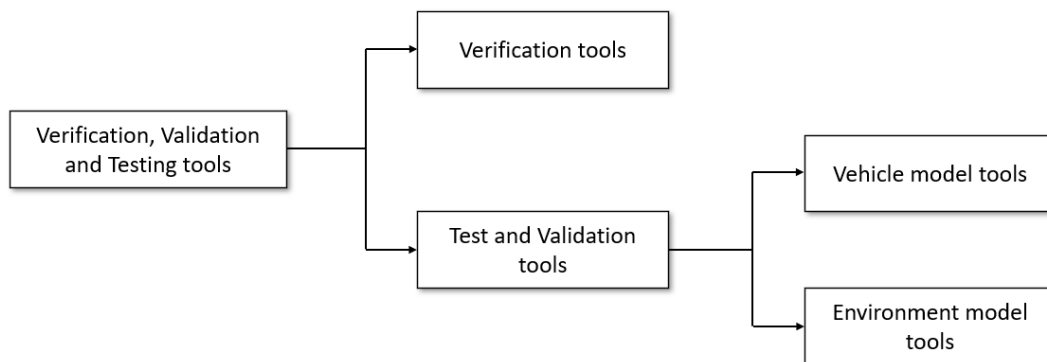


Figure 3.7 - Classification of tools for VVT.

In the verification tools, both the theorem proving and the run-time monitoring do not use automatic tools to check the correctness of the system properties. Model checking is the only formal method that uses a wide range of tools available on the market. These tools are chosen according to the most suitable purpose [101].

Test and validation tools use driving simulation platforms to conduct simulated road testing in a virtual environment. In this way, companies can quickly and economically complete the design explorations needed to develop a wide range of technologies that are required for the connected and autonomous vehicles. These platforms are very popular because they are developed and tested with a high level of accuracy and, therefore, can be trusted for high fidelity testing [102].

The section reviews some of the most popular software tools for VVT of CAVs.

3.3.1 Verification tools

Model checking allows verification tools to be largely automatized, where the most used model checkers are shown in Table 3.3.

Table 3.3 - Model checkers.

Model checkers	Description
UPPAAL	integrated probabilistic tools for the modeling and the verification of real-time systems and its modeling formalism is an extension of timed automata.
CADP	
PRISM	
BTC Embedded Validator	verification tools that use formal methods to identify hidden design errors in Simulink/Stateflow models by performing automated searches for violations.
Reactis Validator	
Simulink Design Verifier	

Several verification tools, like UPPAAL, CADP and PRISM model checkers, use timed automata (TA) as modeling formalism to specify the behavior of CAVs with clock variables and simple constraints over clocks and states [103]. The TA model is the most well-established model for the specification and verification of real-time systems and allows to create clear and concise abstract models of CAV systems and allows to apply algorithms designed for timed properties [104].

Other tools, such as BTC Embedded Validator, Reactis Validator and Simulink Design Verifier, allow the formulation of a property that the model's behavior should have as an assertion, and checks if the model (compatible with the verification tool) satisfies that property under all possible scenarios. The result of the search for violations can either be “True” or “False”, where in the case of being “False” the tool produces a simulated scenario to see how the model violates the property [105].

3.3.2 Test and validation tools

Test and validation tools are driving simulation tools that place the driver in an artificial environment believed to be a valid substitute for one or more aspects of the actual driving experience, exposing drivers to various challenging and risky scenarios in a controlled environment, since it is difficult to reproduce in reality some road situations. These tools provide a safe environment for real-time testing and help predicting equivalent measurements in the real world that lead to a better understanding of the complex driver–vehicle–road interaction in critical driving situations.

In the literature, the validation of closed-loop perception, planning and control algorithms is performed by testing the configuration of the vehicle and a wide variety of traffic scenarios [37]. In this way, the driving simulation tools can be classified in two ways: vehicle configuration tools and environment configuration tools. Table 3.4 shows the different models that can be configured with each tool.

Table 3.4 - Difference between tools.

Vehicle model tools	Environment model tools
vehicle settings	vehicle settings
sensor models	sensor models
-	road models
-	traffic models
-	pedestrian models

Vehicle model tools

These tools are used for prototyping the characteristics of road vehicles and to test their dynamics (physics-based), where is possible to create vehicle models: motor models, sensor models (vision-based, infrared and ultrasonic sensors, radar, LiDAR, etc.), actuator models (brake and steering systems) and vehicular communication (V2X). It involves executing high-fidelity mathematical models capturing continuous dynamic behaviors of vehicles and their environment [106]. Vehicle dynamics require dynamic driving tasks, which includes operational (e.g., steering, braking, accelerating) and tactical (e.g., responding to events, determining when to change lanes, turn) aspects of the driving task [9]. Some of the most used tools to configure the system of an autonomous vehicle are: Matlab/Simulink, LabVIEW, CarMaker/TruckMaker, ROS, Webots, Simcenter Amesim and Unity 3D.

Matlab/Simulink, created by MathWorks, allows the design of vehicle models to simulate their behavior - enables the simulation of their perception, planning, and control systems, that is, it allows the design and simulation of the sensors and the dynamics of the vehicle (e.g., how a vehicle's forward movement changes in response to driver inputs) [107]. The laboratory virtual instrument engineering workbench (LabVIEW), created by National Instruments, also allows the design and the simulation of vehicle models [108]. Both Simulink and LabVIEW have difficulty interacting with the hardware. CarMaker/TruckMaker, created by the IPG Automotive, is an extremely fast vehicle simulation platform with sophisticated vehicle models that allows full HIL simulations [102]. The robot operating system (ROS), created by Open Robotics, and Webots, created by Cyberbotics [109], are open-source robotic simulators that help building robot applications. As CAVs are a type of robot, the same types of programs can be used to control them [34]. Simcenter Amesim, created by SIEMENS, provides an integrated simulation platform to accurately predict the performance of intelligent systems, specifically the vehicle dynamics [37]. Unity 3D is a game engine platform originally design for the creation of video games. However, in recent years it has served to generate several simulators to implement path planning, control and vision systems of vehicles [34].

Environment model tools

These tools are perception-based simulators (AVs are decision-making systems that receive feedback from the surrounding environment through sensors), which simulate a dynamic world, allow not only the modeling of vehicle models, but also allow the development and manipulation of road models and traffic models. Simulations can model a real system with all its static (e.g., buildings, traffic signs) and dynamic (other vehicles or the traffic flow, pedestrians, etc.) components, as well as modeling the weather (e.g., rain, fog, day or night). Some of environment configuration tools are: CARLA, Dynacar, DYNA4, CarSim/TruckSim, Pro-SiVIC, Simcenter Prescan, VTD, NVIDIA DRIVE, SUMO, Synchro and Gazebo, where most of them are compatible with Matlab/Simulink.

The CAR Learning to Act (CARLA), created by the Computer Vision Center, is an open-source simulator that has been used to study the performance of autonomous driving according to three different approaches: a modular pipeline, an end-to-end model trained via imitation learning and an end-to-end model trained via reinforcement learning [110]. Dynacar, developed by Tecnia Research and Innovation, focuses on the validation of control and route planning [34] – it allows a good definition of trajectories and cooperative maneuvers. In addition, this platform allows HIL simulation with

different types of vehicles and scenarios [111]. DYNA4, created by Vector Informatik, also allows the execution of closed-loops simulations with the integration of ECUs. Driving simulation tools like CarSim/TruckSim, created by Mechanical Simulation Corporation, the simulator vehicle infrastructure sensors (Pro-SiVIC), created by Civitec, the Simcenter Prescan, created by SIEMENS, [111], the Virtual Test Drive (VTD), created by VIRES [85] and the NVIDIA DRIVE are commercial simulators (most of them are expensive to buy and maintain [102]) that allow developers to design and implement detailed simulations for vehicle testing and validation [34]. They support HIL simulation. The Simulation of Urban Mobility (SUMO), created by DLR [24], and Synchro, created by Trafficware [112], are open source microscopic traffic flow simulators for analysis, optimization and visualization of road networks. Each vehicle is modeled with its own route and moves individually through the network. Gazebo, created by Open Robotics, is a 3D environment simulator that provides realistic rendering of environments, including high-quality lighting, shadows and textures, and supports HIL simulations [24].

3.4 Challenges of SAE level 3+

With the increase in the level of automation in the automotive industry, vehicles are becoming increasingly complex and new and different challenges are emerging. In this way, the main challenges related to verification, validation and testing of CAVs can be classified into five major categories: sensors, communication, machine learning, testing and regulation, where the first four of these challenges must be solved to help overcome the last one.

3.4.1 Sensors

Sensors feed the vehicle's control system with data acquired from the perception of the surrounding environment and this is highly influenced by external factors like adverse weather conditions, intense traffic or poor road signaling. This means that the decisions made by the CAVs depend directly on the accuracy of the data acquired by the sensors and, therefore, high precision sensors are needed to achieve autonomous driving.

In addition, another difficulty of CAVs is the integration of all sensors in the vehicle system, and their fusion to form a single model - the sensor fusion unit [34].

3.4.2 Communication

Like sensors, CAVs connectivity need to capture and transmit data in real time over the surrounding environment in a wide range of conditions. In complex scenarios, such as intersections, especially the V2I communication reveals problems of scalability, bandwidth, versatility and universality during its simulation [112].

Another challenge in vehicle communications is the data security and privacy. Because of the information exchanged between vehicles and with infrastructures, these system are vulnerable to cyber-attacks. More than ever, security software needs to be developed to make cyber-attacks more difficult [113].

3.4.3 Machine learning

Machine learning algorithms are difficult to test and analyze, since it is difficult to characterize all the behaviors of these components under all circumstances, mainly due to the high number of parameters and the difficulty in characterizing the training data [109] – it is difficult to validate perception, planning and control algorithms with the simulation of vehicle dynamics in several traffic scenarios [37].

Besides that, integrating machine learning components with traditional software is a very challenging task.

3.4.4 Testing

The challenges in testing CAVs start with the requirements definition, where this is difficult due the multiple areas involved – it is necessary to define functional and safety requirements and make them all consistent [26]. However, the biggest problem in testing comes from the high complexity of these systems, where given the complexity of CAV systems (based on events) it is difficult to test such systems and consequently it is difficult to evaluate the entire implementation with all the time constraints [21]. Other difficulties for the verification, validation and testing of CAVs are summarized as follows:

- Difficulty in the specification of input data for the test case generation [15];
- Difficulty in characterizing scenarios: the need to consider the complexity and variety of scenarios and the constant possibility of interaction between multiple systems [114], the need to consider the expression of uncertainty in the driving

environment, as well as the absence of approaches to assess the safety of driving decisions [115];

- Difficulties in generating scenarios: generating environmental elements and assembling them in simulations that implement relevant and realistic test cases that challenge the autonomous vehicle software [98] and that closely resemble the situation on the road [27], respectively;
- Difficulty in integrating the vehicle in an infrastructure with human drivers - other traffic participants can cause a critical scenario for CAVs [87].
- Difficulties in simulating the vehicle dynamics: the configuration must correspond to the interface of the control algorithm and its behavior must correspond as closely as possible to the actual behavior of the vehicle [116].
- Difficulties presented by the simulation platforms: they do not provide guidance for which test scenarios should be selected for simulation and simulations of vehicle dynamics are computationally expensive [97].
- Difficulty to perform tests that produce quantitative, repeatable and comparable results, since we do not have a detailed and testable definition of the intelligence of autonomous vehicles [75].
- Difficulty to facilitate efficient interaction at different stages of testing and validation in the V-model, have feedback from real-world test and the development of the test cases and test scenarios [83].

3.4.5 Regulation

Sufficient standards and regulations for a complete system do not exist in any industry. However, the safety of connected and autonomous vehicles is vital, especially since a malfunction of the system can result in loss of life. Therefore, new legislation is needed to verify, validate and test CAVs - there are some standards and guidelines in the automotive industry, but they do not solve the problems of sensor, communication, testing and machine learning concepts [91]. New standards (e.g., ISO/SAE 21434) are starting to become more mature to improve the safety and security of CAVs and can be expected in the near future [82].

Chapter 4

Trajectory planning based on adaptive model predictive control: Study of the performance of an autonomous vehicle in critical highway scenarios

This chapter shows a detailed description of the modeling process used to test and validate an autonomous vehicle model through several pre-defined scenarios using a software-in-the-loop simulation in Simulink. The main objective of the project is to avoid any collision in order to ensure the vehicle's safety, while driving autonomously. As the surrounding conditions and disturbances could not be controlled, the only option to avoid collision is the intelligent functioning of the main vehicle (called ego vehicle) and controlling its velocity, acceleration and steering angle by using an adaptive controller. For this purpose, a model predictive control (MPC) controller was used to accomplish this task. This model was implemented in a test bench by merging and configuring three ADAS functions in Matlab & Simulink: cruise control (CC); following lead vehicle (FLV); and lane change (LC). Furthermore, this vehicle model does not include a sensing system (sensor fusion).

As mentioned in Chapter 2, the planning system of an autonomous vehicle consists of route planning, behavioral planning and trajectory planning. However, for this model only trajectory planning is addressed, as shown in Figure 4.1. Trajectory planning takes the global path previously defined by route and behavioral planners (called reference path) and provides it with time, velocity, acceleration and jerk information to generate inputs to the vehicle's control system - ensuring that the planned trajectory is executed.

The model has been simulated in some pre-defined scenarios - with an ego vehicle reference path, designed in the Driving Scenario designer app from Matlab - to verify if the main vehicle uses all ADAS functions properly.

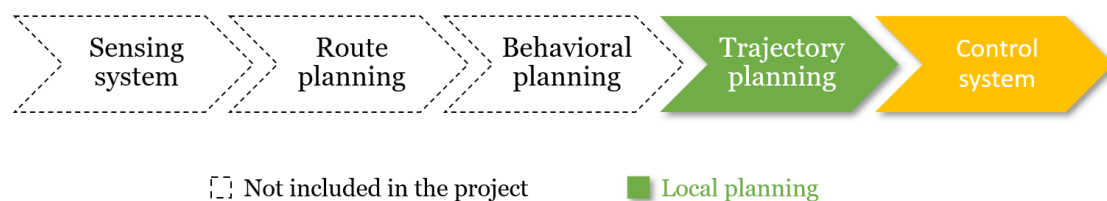


Figure 4.1 - Planning system and control system used in the model.

Thus, the vehicle model was built in four steps: (1) designing the ego vehicle dynamics, (2) designing the pre-defined scenarios, with an ego vehicle reference path, (3) designing the planning system for the ego vehicle and (4) designing the ego vehicle control system.

This project uses object-oriented programming (OOPs) to allow the combination of data and its associated actions (functions) into objects where they can evolve and change over time without introducing incompatibilities. Compared to other conventional programming languages (e.g., Java, Python), the OOPs in MATLAB allow to solve complex computing problems as it facilitates code modularization making programming faster.

To model an object to have certain characteristics and behavior, classes are used. A Matlab class contains a blueprint used to build a specific type of object and is mainly composed of a set of properties and methods, and their attributes. The class's properties store data for each of the class's objects, while the class's methods contain a set of functions that define the operations that can be performed on each object of the class. Some specialized kinds of methods like the constructor are optional. Constructor methods are specialized methods that create objects of the class. A constructor method have the same name as the class and typically initializes property values with data obtained from input arguments. When defining a class, attributes can be specified to control the behavior of the class's properties and methods and to control how they are accessed from outside the object. For example, properties and methods can be public, private, or protected.

Inheritance is one of the key points of object-oriented programming, as it allows the reuse of existing code, facilitating the creation of the project. Inheritance allows to create new classes built from existing ones (superclasses), to specify a new implementation while maintaining the same behaviors. An inherited class is called a subclass and all methods and attributes of its respective superclass will be inherited by it. To the subclass, new methods and new attributes can be added in a process of successive specialization.

In addition to defining properties and methods, it is also possible to define enumerations in classes. Enumeration classes are specialized classes that define a fixed set of names representing a single type of value (integer types). This class type was used to define the driving modes supported in this model.

Furthermore, as this project also requires implementation and simulation of dynamic systems with inputs that change over time and the values of the output signals depend on the instantaneous values of the input signals and on the past behavior of the

system, system objects were used. A System Object¹¹ is a specialized kind of Matlab class with specific methods and properties, designed specifically for these dynamic systems. System objects use internal states to store their past behavior, which is used in the next computational step. As a result, system objects are optimized for iterative computations that process large streams of data in segments. This ability to process streaming data provides the advantage of not having to hold large amounts of data in memory and simplifies the model by using loops efficiently.

4.1 Coordinate systems

This model uses three cartesian coordinate systems defined in the SAE J670 and ISO 8855 standards. Vehicle dynamics modeling (Chapter 4.2) follows the SAE J670 convention with two coordinate systems: the earth-fixed coordinate system (inertial) and the vehicle coordinate system. Path planning, localization, mapping and driving scenario simulation (Chapters 4.3, 4.4 and 4.5) follow the ISO8855 convention, which defines the world coordinate system.

Earth-fixed coordinate system: Newtonian physics considers that the earth is an inertial reference, therefore, in the earth-fixed coordinate system, the axes (X_E , Y_E , Z_E) are fixed in an inertial reference frame where the angular velocity and linear and angular acceleration are zero. The X_E and Y_E axes are parallel to the ground plane and the Z_E axis is aligned with the gravitational vector. [117]

Vehicle coordinate system: This system coincides with the earth-fixed coordinate system and begins in the point where motion is initialized. The vehicle system (X_V , Y_V , Z_V) is anchored to the main vehicle placed on the ground right below the midpoint of the rear axle (Figure 4.2). The X_V axis points forward from the vehicle, the Y_V axis points to the right and the Z_V axis points downward in the Z-Down orientation. [117]

World coordinate system: All vehicles are placed in a fixed universal coordinate system that follows the ISO 8855 convention for rotation. The X axis points forward, the Y axis points to the left and the Z axis points up from the ground (Z-Up orientation), as illustrated in Figure 4.2. In this study, the world coordinate system is used in global-frame and frenet-frame.

¹¹ https://www.mathworks.com/help/matlab/matlab_prog/what-are-system-objects.html

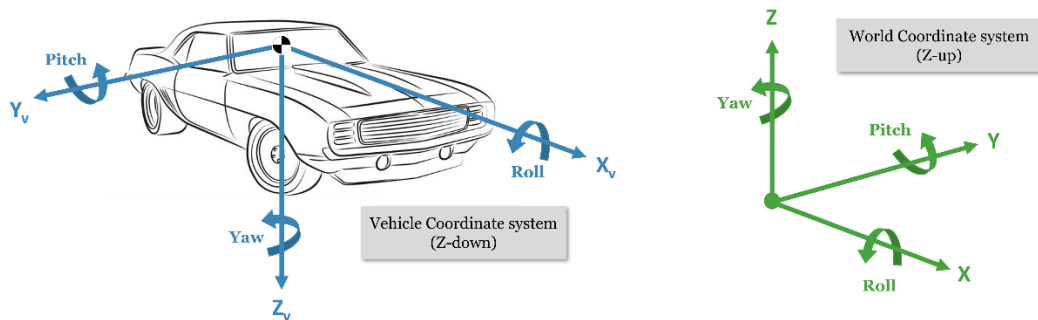


Figure 4.2 - Axis systems for the vehicle coordinate system and the world coordinate system.

4.2 Vehicle dynamics

To effectively design a control system for the ego vehicle, a simple dynamic model of the system under study is required. Here, Vehicle Dynamics Blockset¹² of Matlab was used (vehicle coordinate system) and the dynamic aspects are explained in detail.

Assumption 1: The ego vehicle only travels in a forward motion with a constant velocity and has the following initial conditions: (1) the longitudinal velocity is equal to the initial longitudinal velocity parameter (V_o) set in the driving scenario (pre-built scenarios in Driving scenario app), (2) the lateral velocity is zero, as well as the longitudinal acceleration, the angular velocity and the steering angle.

Assumption 2: The ego vehicle has 4 wheels in which the left and right axle are lumped into a single wheel each (bicycle model). The tractive force of the vehicle comes only from the front wheels (front-wheel drive). The vehicle mass is lumped in the Center of Gravity (CG), lying in the segment that connects the two wheels.

Assumption 3: The vehicle has 3 Degrees Of Freedom (DOF): two displacements on the plane (longitudinal and lateral) and the rotation around an axis normal to that plane (yaw rotation).

Assumption 4: Suspension movement, road inclination and aerodynamic influences are neglected. Whenever to simulate the model it was necessary to use parameters related to these characteristics, default values were used.

¹² <https://www.mathworks.com/products/vehicle-dynamics.html>

Assumption 5: The other vehicles that travel around the main vehicle are called target vehicles or Most Important Objects (MIOs). All these vehicles travel at a constant speed on a fixed reference path (fixed waypoints).

Vehicle dynamics models are distinguished with regards to degrees of freedom and the model of this project uses 3DOF. These 3DOF models are widely used for simulation purposes in which several behaviors of a vehicle such as velocity, acceleration, braking, and steering are being studied. To design and model the ego vehicle, the Vehicle Body 3DOF¹³ block from the Vehicle Dynamics Blockset™ of Matlab was used. This block implements a rigid two-axle vehicle body model to calculate longitudinal, lateral and yaw motion; and is used when vehicle pitch, roll, and vertical motion are not significant. By controlling these 3DOF over time, the vehicle's trajectory will be known, so the path described by the vehicle can be studied.

As additional simplifications are made, such as considering that the vehicle travels at constant speed, the model can be represented by a two-wheeled vehicle model: single-track (bicycle model). The bicycle model represents the vehicle's lateral dynamics in good detail, since in this project, the lane change trajectories are more important than tire forces or vehicle stability. The diagram of the model is showed in figure 4.3.

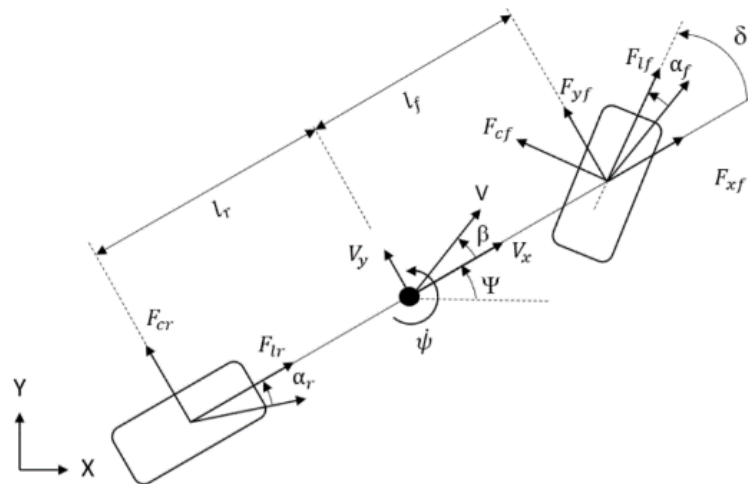


Figure 4.3 - Bicycle model.

Since the vehicle is assumed to have planar motion, three coordinates are necessary to describe the vehicle motion: X, Y and Ψ , where (X, Y) represent the inertial coordinates of the location of the center of gravity of the vehicle, while Ψ (yaw angle) indicates the orientation of the vehicle-fixed frame about the earth-fixed z-axis. The

¹³ <https://www.mathworks.com/help/vdynblks/ref/vehiclebody3dof.html>

vector V is the velocity at the CG of the vehicle and makes a slip angle (β) with the longitudinal axis of the vehicle. In an absolute inertial frame, the vehicle's equations of motion are represented as follows

$$\dot{X} = V \cos (\Psi + \beta) \quad (4.1)$$

$$\dot{Y} = V \sin (\Psi + \beta) \quad (4.2)$$

$$\dot{\Psi} = \frac{V \cos (\beta)}{l_f + l_r} \tan (\delta) \quad (4.3)$$

where \dot{X} and \dot{Y} are the longitudinal and lateral velocities, $\dot{\Psi}$ (yaw rate) is the vehicle angular velocity about the vehicle z-axis, δ is the steering angle, l_f and l_r are the longitudinal distance from the center of gravity to front and rear wheel, respectively.

The bicycle model is derived by adding forces (F_x, F_y, F_z) and moments (M_x, M_y, M_z) around the vertical axis at the center of gravity and it shows the vehicle lateral motion dynamics as well as the rotational dynamics. In this simple case, to obtain longitudinal and lateral accelerations, we can derive the equations of longitudinal and lateral motion by using Newton's equations for translational motion in X and Y , while to obtain angular motion we use Euler's equations in z-axis. Additionally, as we want to take into account the changes in the longitudinal velocity on the lateral and yaw motion, the vehicle is configured to use external longitudinal forces to accelerate or brake the vehicle.

$$m\ddot{X} = m\dot{Y}\dot{\Psi} + F_{xf} + F_{xr} + F_{x\ ext} \quad (4.4)$$

$$m\ddot{Y} = -m\dot{X}\dot{\Psi} + F_{yf} + F_{yr} + F_{y\ ext} \quad (4.5)$$

$$I_{zz}\ddot{\Psi} = l_f F_{yf} - l_r F_{yr} + M_{z\ ext} \quad (4.6)$$

where \ddot{X} and \ddot{Y} are the longitudinal and lateral accelerations, $\ddot{\Psi}$ is the vehicle angular acceleration, m is the mass of the vehicle, F_{xf} and F_{xr} are the longitudinal forces applied to front and rear wheels, F_{yf} and F_{yr} are the lateral forces applied to front and rear wheel, $F_{x\ ext}$ and $F_{y\ ext}$ are the external longitudinal and lateral forces applied to vehicle CG, $M_{z\ ext}$ is the external moment of the vehicle CG about the vehicle z-axis and I_{zz} is the yaw polar inertia.

Longitudinal and lateral forces applied to front and rear wheels are calculated with respect to the longitudinal and lateral tire forces, and the steering angle

$$F_{xf} = F_{lf} \cos(\delta) - F_{cf} \sin(\delta) \quad (4.7)$$

$$F_{yf} = F_{lf} \sin(\delta) - F_{cf} \cos(\delta) \quad (4.8)$$

$$F_{xr} = F_{lr} \quad (4.9)$$

$$F_{yr} = F_{cr} \quad (4.10)$$

where F_l is the traction force of tires and F_c is the cornering force of tires.

Tires generate longitudinal force during accelerating and decelerating (traction) and generate lateral force during cornering. In this way, the longitudinal tire forces are calculated using the input forces by the equations (4.11) and (4.12), and the lateral tire forces are calculated by the equations (4.13) and (4.14) using tire slip angles and linear cornering stiffness, and depend on the friction characteristics between the road and the tires

$$F_{lf} = F_{xf} \text{ input} \quad (4.11)$$

$$F_{lr} = F_{xr} \text{ input} \quad (4.12)$$

$$F_{cf} = -C_{yf} \alpha_f \mu_f \frac{F_{zf}}{F_{znom}} \quad (4.13)$$

$$F_{cr} = -C_{yr} \alpha_r \mu_r \frac{F_{zr}}{F_{znom}} \quad (4.14)$$

where C_{yf} , C_{yr} are the front and rear wheel cornering stiffness, α_f and α_r are the front and rear wheel slip angles, μ_f and μ_r are the front and rear wheel friction coefficient (dimensionless), F_{zf} and F_{zr} are the normal force applied to front and rear wheels along vehicle z-axis and F_{znom} is the nominal normal force applied to axles, also along in the vehicle z-axis.

To maintain pitch and roll equilibrium, the normal forces, obtained by equations (4.15) and (4.16), are divided by the nominal normal load to vary the effective friction parameters during weight and load transfer

$$(l_f + l_r) F_{zf} = l_r mg - (\ddot{X} - \dot{Y}\dot{\Psi})mh + hF_{x\ ext} + l_r F_{z\ ext} - M_{y\ ext} \quad (4.15)$$

$$(l_f + l_r) F_{zr} = l_f mg - (\ddot{X} - \dot{Y}\dot{\Psi})mh - hF_{x\ ext} + l_f F_{z\ ext} - M_{y\ ext} \quad (4.16)$$

where h is the height of vehicle CG above the axle plane, $F_{z\ ext}$ is the external force applied to vehicle CG along the vehicle z-axis, $M_{y\ ext}$ is the external moment of the vehicle CG about the vehicle y-axis.

The slip angle of the tires, given by equations (4.17) and (4.18), represent the angle between the wheel velocity and the direction of the wheel itself

$$\alpha_f = \text{atan}\left(\frac{\dot{Y} + l_f \dot{\Psi}}{\dot{X}}\right) - \delta \quad (4.17)$$

$$\alpha_r = \text{atan}\left(\frac{\dot{Y} - l_r \dot{\Psi}}{\dot{X}}\right) \quad (4.18)$$

The Vehicle Body 3DOF block used to design the ego vehicle in Simulink implements all these equations and the required fixed values for the simulation to work used in this block are shown in Table 4.1.

Table 4.1 - Fixed ego vehicle parameters.

Parameter	Value	Units
I_{zz}	2875	$Kg\ m^2$
V_0	0	$\frac{m}{s}$
C_{yf}	19000	$\frac{N}{rad}$
C_{yr}	33000	$\frac{N}{rad}$
ψ_0	0	$\frac{rad}{s}$
F_{znom}	5000	N
m	1575	Kg
l_f	1,2	m
l_r	1,6	m
h	0,35	m
α_f	0,1	rad
α_r	0,1	rad

In addition, Figure 4.4 shows all the values used in the Vehicle Body 3DOF block: the fixed parameters and the time-varying parameters required for the simulation.

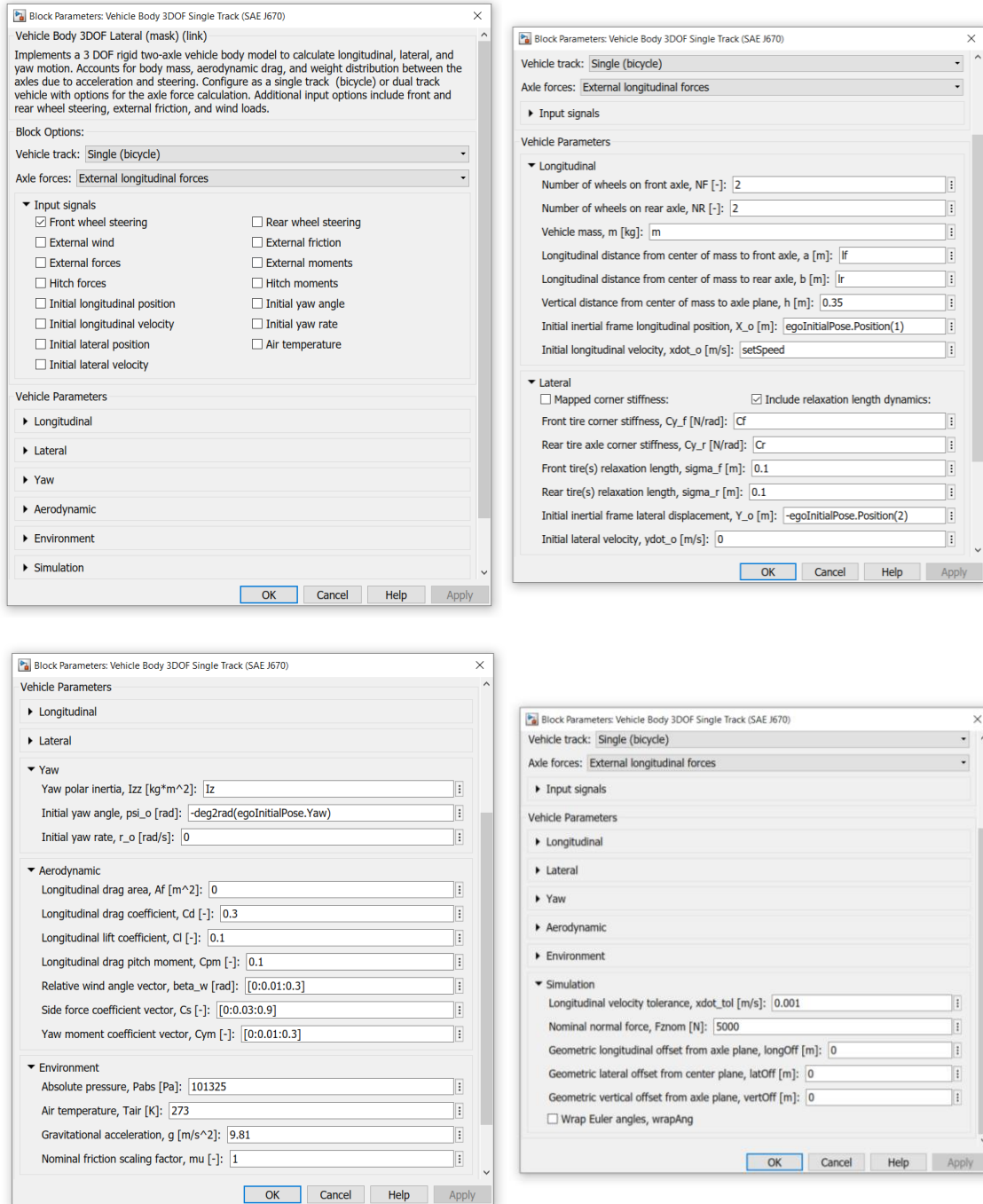


Figure 4.4 - Parameters defined for the ego vehicle dynamics.

4.3 Scenario setup

The simulation environment was built using the Driving Scenario Designer¹⁴ app and the Automated Driving Toolbox¹⁵ from Matlab. Both use the world coordinate system for testing the autonomous driving model. All the scenarios were built in this app (see Figures 4.6, 4.7 and 4.8) and were exported as a Matlab function, where the Matlab code generated is equivalent to the scenarios created. The Driving Scenario Designer app generates a drivingScenario¹⁶ object which contains information about roads, vehicles (actors), pedestrians and barriers of a scenario. By modifying the code in the exported Matlab function, it was possible to generate multiple variations of each original scenario, such as changing the waypoints (reference path) or the lateral offset (lane) of the ego vehicle and the target vehicles, varying the speeds of all vehicles, etc.. These exported Matlab function were called in the main Matlab script to generate the scenario variables needed to run the closed-loop Simulink model in real time. In Simulink, these scenario variables are read by the Scenario Reader¹⁷ block. The Scenario reader block needs two conversions to work properly (Figure 4.5):

- (1) It requires a discrete signal to update the vehicle state in the driving scenario, for which Rate transition¹⁸ blocks were used, where the rate of transition is equal to the controller sample time.
- (2) It requires the conversion of the physical quantities of the ego vehicle (coming from the Vehicle body 3DOF block) expressed in the SAE J670 convention to the ISO8855 convention (used by the Scenario reader block).

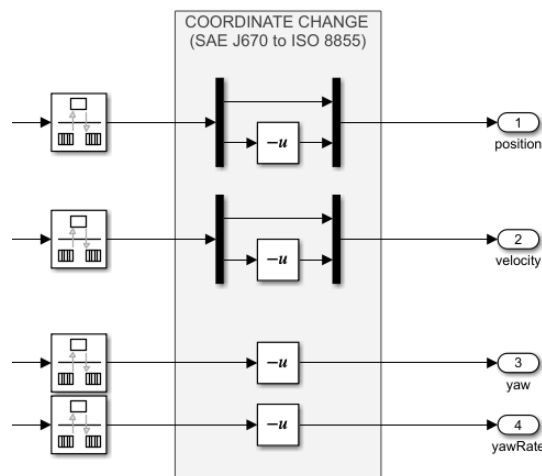


Figure 4.5 - Signal conversions.

¹⁴ <https://www.mathworks.com/help/driving/ref/drivingsceniodesigner-app.html>

¹⁵ <https://www.mathworks.com/products/automated-driving.html>

¹⁶ <https://www.mathworks.com/help/driving/ref/drivingscenario.html>

¹⁷ <https://www.mathworks.com/help/driving/ref/scenarioreader.html>

¹⁸ <https://www.mathworks.com/help/simulink/slref/ratettransition.html>

All scenarios were set with an ego vehicle with a reference path that has two waypoints: the initial waypoint (positioned on the vehicle's CG) and the final waypoint, which corresponds to the end of the road. All roads have a longitudinal extension of 800 meters and each lane has a lateral length of 3.6 meters (default value of the Driving Scenario Designer app). Simulation time was set to 30 seconds for all scenarios.

In relation to the ego vehicle, the middle of the lateral length of the road corresponds to the zero reference, where the right lanes support negative values (negative lane centers) and the left lanes support positive values (positive lane centers).

Driving scenario 1: This scenario has two lanes and contains three target vehicles.

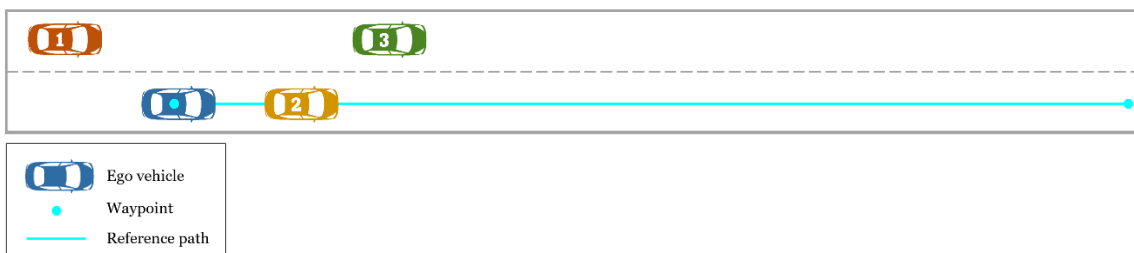


Figure 4.6 - Driving scenario 1.

Driving scenario 2: This scenario has three lanes and contains six target vehicles.

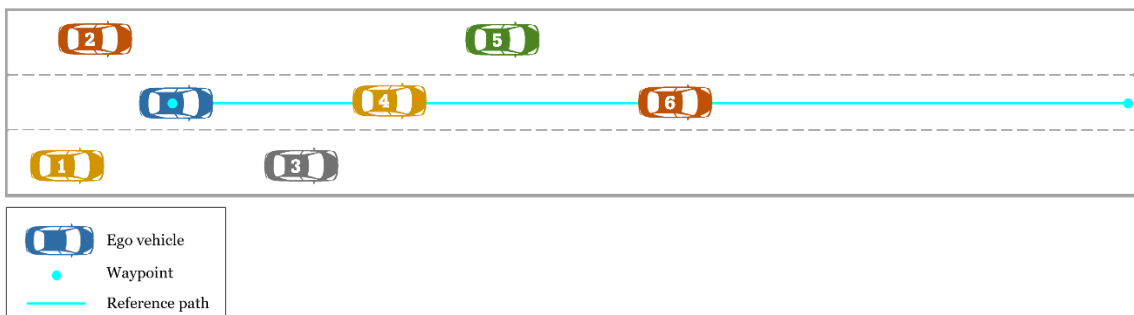


Figure 4.7 - Driving scenario 2.

Driving scenario 3: This scenario has four lanes and contains nine target vehicles.

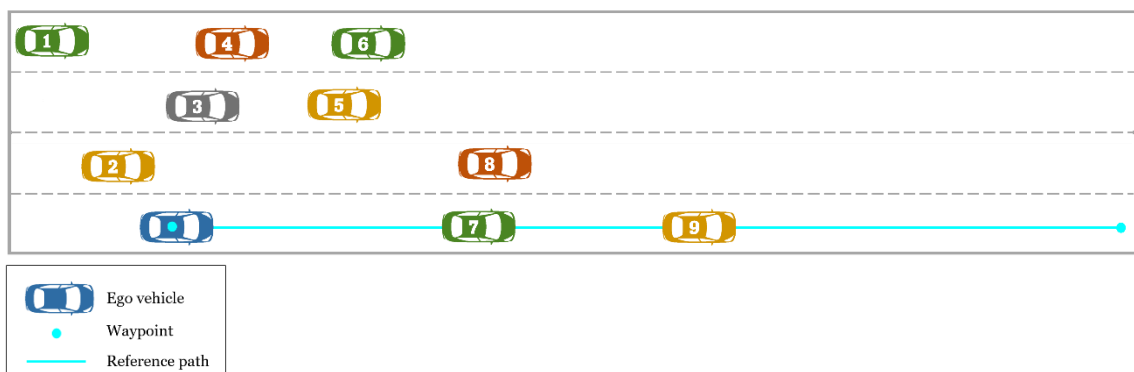


Figure 4.8 - Driving scenario 3.

The physical quantities of the ego vehicle converted before are then packed in a Matlab structure through the function “Ego” (see Figure 4.10), since the Scenario Reader block demands an input signal of the ego vehicle pose containing a Matlab structure with the fields: ID, position, velocity, yaw angle and yaw rate (angular velocity) of the ego vehicle. In the existence of an ego vehicle (which is the case), the Scenario reader block will convert the physical quantities of all other actors into the vehicle coordinate system. This block can also read the road boundary data (in vehicle coordinate system). The parameters of the Scenario reader block can be seen in Figure 4.9.

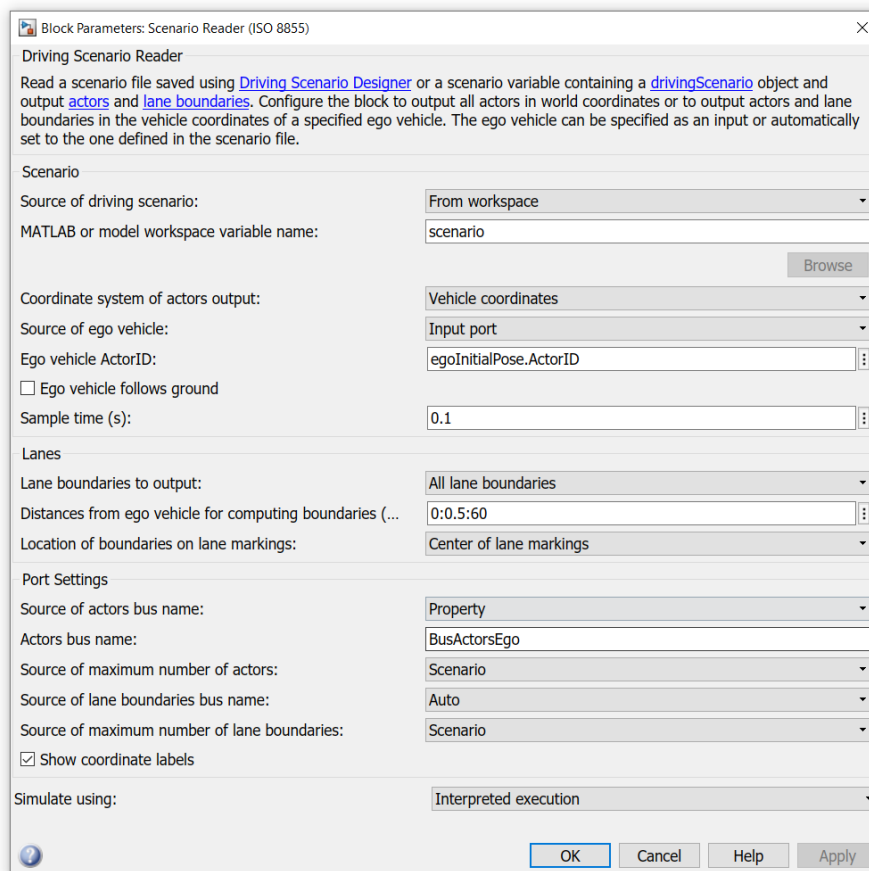


Figure 4.9 - Scenario reader block parameters.

The trajectory planning system of the Simulink model requires target vehicles positions in frenet and global coordinates. In this way, the Vehicle To World¹⁹ block was used to convert physical quantities of non-ego actors outputted by the Scenario Reader block into world coordinates.

¹⁹ <https://www.mathworks.com/help/driving/ref/vehicletoworld.html#:~:text=%C3%97-Description,with%20the%203D%20simulation%20environment.>

Both the outputs of Scenario Reader block (in vehicle coordinates) and Vehicle To World block (in world coordinates) are a Simulink bus containing a Matlab structure with the fields: number of actors, current simulation time and actor poses. The actor poses include the positions, velocities, and orientations of actors in the driving scenario, where each actor pose is a Simulink bus containing the Matlab structure with the fields of Table 4.2.

Table 4.2 - Actor pose structure^{17,19}.

Field	Description	Values	Units
Actor ID	Scenario-defined actor identifier	Positive integer	-
Position	Position of the ego actor	Real-valued vector of the form [x y z]	Meter
Velocity	Velocity of the ego actor in the x-, y-, and z-directions	Real-valued vector of the form [vx vy vz]	Meter per second
Roll	Roll angle of the ego actor	Real-valued scalar	Degree
Pitch	Pitch angle of the ego actor	Real-valued scalar	Degree
Yaw	Yaw angle of the ego actor	Real-valued scalar	Degree
Angular Velocity	Angular velocity of the ego actor in the x-, y-, and z-directions	Real-valued vector of the form [ω_x ω_y ω_z]	Degree per second

The Scenario reader block automatically creates four Simulink buses: “BusActorsEgoActors” and “BusActorsEgo” for actors, and “BusLaneBoundaries1” and “BusLaneBoundaries1LaneBoundaries” for lanes. In this vehicle model, as the lane information was not used, only the actor bus objects were exported from the Bus Editor²⁰ and saved into Matlab files, stored in the Simulink.Bus object format. The “BusActorsEgoActors” (output of the Scenario Reader block) was change²¹ to “BusActorsEgo” and saved in Matlab file called “CreateBusActorsEgo.m”, where the number of actors was replaced for a variable, since this is a variable parameter for each scenario. The “BusActorsEgo” (actor pose structure) was changed²¹ to “BusActor” and saved in the Matlab file called “CreateBusObjects.m” to initialize this bus object in the Matlab base workspace.

In addition, a function was also created to interrupt the simulation when a collision between the ego vehicle and a target vehicle occurs - "collisionDetect" function (Figure 4.10). Collision detection occurs when the ego vehicle and target vehicle polyshapes overlap. The calculation of polyshapes is achieved through the width, length, and rear overhang of the actors.

²⁰ <https://www.mathworks.com/help/simulink/slref/buseditor.html>

²¹ The bus name was changed to a more intuitive name only by authors' choice.

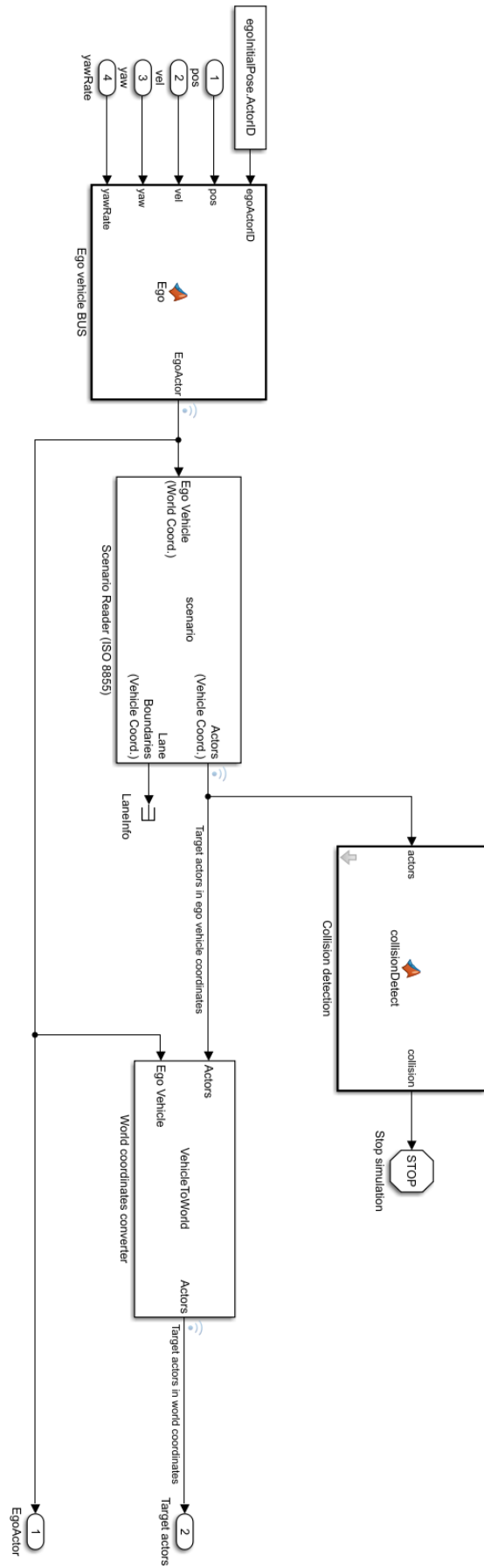


Figure 4.10 - Scheme of the scenario setup of the Simulink model.

4.4 Vehicle path planning

Trajectory planning uses local planners to adjust the global path based on obstacles in the environment, generating alternative trajectories for the ego vehicle through the traffic. The local planning strategy samples a set of local trajectories based on the current and foreseen state of the environment before choosing the most optimal trajectory. The generator of local trajectories²² for planning adaptive routes in Matlab is done in Frenet-frame.

4.4.1 States conversion

The traveling path is defined by parametric curves, particularly on curved roads and lane changes. To achieve optimization and time efficiency, many trajectory planning algorithms are developed in the frenet-frame to reduce the planning dimension when generating trajectories along the shape of the reference path, that is, the frenet coordinate system is a way of representing position on a road in a more intuitive way than the traditional cartesian coordinates (mathematically simpler representation) [118]. In this way, the cartesian states (global-frame) of the waypoints in the reference path need to be converted to the frenet-frame, as illustrated in Figure 4.11.

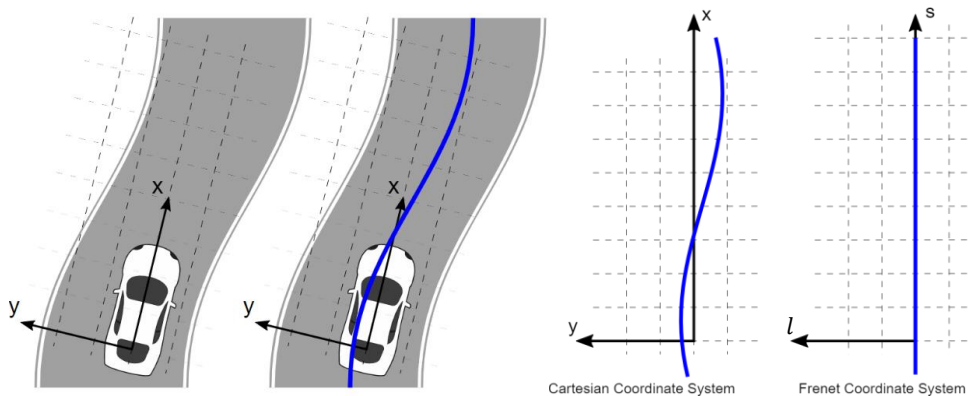


Figure 4.11 - Comparison of a planned trajectory in Cartesian and Frenet coordinates.

The global states of a vehicle are represented by a six-element row vector

$$globalState = [X, Y, \theta, K, V, A]$$

²² <https://www.mathworks.com/help/nav/ug/choose-path-planning-algorithms-for-navigation.html>

where (X, Y) is the vehicle position in meters, θ is the orientation angle in radians, K is the curvature in m^{-1} , V is the vehicle velocity in m/s and a is the vehicle acceleration in m/s^2 .

With frenet coordinates, the variables s and l are used to describe a vehicle's position, where the s coordinate represents distance along the road (arc length), also known as longitudinal displacement, and the l coordinate represents the side-to-side position on the road relative to a reference path, also known as lateral displacement. Thus, the frenet states are a six-element row vector represented by position, velocity and acceleration relative to a reference path

$$frenetState = [s, \frac{ds}{dt}, \frac{d^2s}{dt^2}, l, \frac{dl}{ds}, \frac{d^2l}{ds^2}]$$

The global states of the ego vehicle and target vehicles obtained before, are saved and also converted to frenet states in the function "statesConverter", requiring the creation of a Simulink bus for the closed-loop model to work. In the Bus Editor, this bus named "BusEgoAndTargetStates" was constructed with the following fields: number of target actors, ego global state, ego frenet state, target global states and target frenet states, as shown in Figure 4.12. The created bus was then exported and saved in the Matlab file "CreateBusObjects.m" to initialize this bus object in the Matlab base workspace.

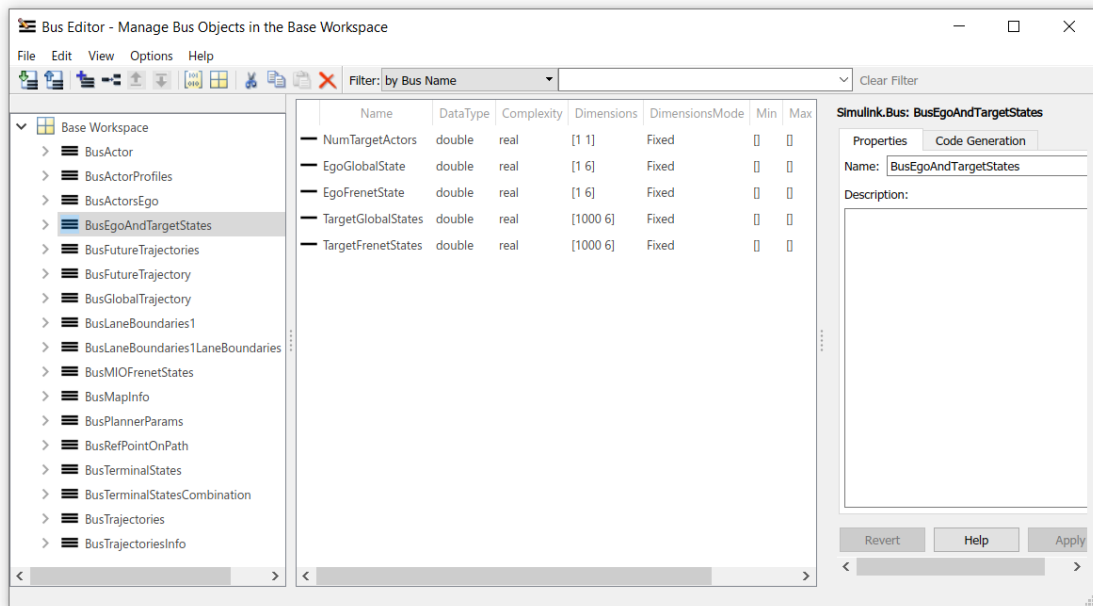


Figure 4.12 - Bus Editor showing the fields of the Simulink bus "BusEgoAndTargetStates".

4.4.2 Polynomials trajectories

The vehicle trajectory is a combination of the lateral movement (offset pattern) $l(t)$ and the longitudinal movement $s(t)$, as illustrated in Figure 4.13. Generally, both patterns are formulated as a 4th/5th order polynomial function, where the fourth-order function enables to generate the minimum acceleration trajectory and the fifth-order order function enables to generate the minimum jerk (acceleration change) trajectory [118]. Since we seek to minimize the jerk to smooth the motion and the safety for high speeds, a fifth-order polynomial is used

$$l(t) = a_6t^5 + a_5t^4 + a_4t^3 + a_3t^2 + a_2t + a_1 \quad (4.19)$$

$$s(t) = b_6t^5 + b_5t^4 + b_4t^3 + b_3t^2 + b_2t + b_1 \quad (4.20)$$

where variable a_i and b_i are computed according to the initial and terminal conditions over time, in frenet coordinates.

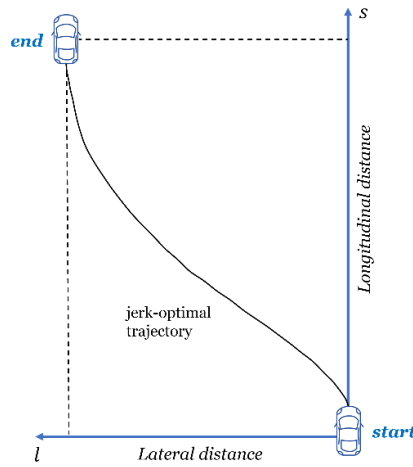


Figure 4.13 - Vehicle trajectory.

As both patterns can be written following the same logic, the adopted quintic polynomial is defined as

$$s(t) = a_6t^5 + a_5t^4 + a_4t^3 + a_3t^2 + a_2t + a_1 \quad (4.21)$$

where s here is considered the longitudinal or lateral distance. In this way, the velocity $\dot{s}(t)$, the acceleration $\ddot{s}(t)$ and the jerk $\dddot{s}(t)$ are given by

$$\dot{s}(t) = 5a_6t^4 + 4a_5t^3 + 3a_4t^2 + 2a_3t + a_2 \quad (4.22)$$

$$\ddot{s}(t) = 20a_6t^3 + 12a_5t^2 + 6a_4t + 2a_3 \quad (4.23)$$

$$\dddot{s}(t) = 60a_6t^2 + 24a_5t + 6a_4 \quad (4.24)$$

The start boundary conditions ($t = 0$) are $s_{start} = a_1$, $\dot{s}_{start} = a_2$, $\ddot{s}_{start} = 2a_3$ and $\dddot{s}_{start} = 6a_4$. In this way, substituting in equation (4.21), the formulation of the longitudinal or lateral movement for a jerk-optimal trajectory generation is defined as

$$s(t) = \frac{1}{6}\ddot{s}_{start}t^3 + \frac{1}{2}\dot{s}_{start}t^2 + \dot{s}_{start}t + s_{start} \quad (4.25)$$

The same happens for velocity, acceleration and jerk

$$\dot{s}(t) = \frac{1}{2}\ddot{s}_{start}t^2 + \dot{s}_{start}t + \dot{s}_{start} \quad (4.26)$$

$$\ddot{s}(t) = \ddot{s}_{start}t + \ddot{s}_{start} \quad (4.27)$$

$$\dddot{s}(t) = \dddot{s}_{start} \quad (4.28)$$

4.4.3 Formulation of local trajectory planning

The Local trajectory planning for the ego vehicle consists of a large and complex system, therefore, it was designed in a separate Simulink file labeled “planner.slx”. This model works as a separate model and is used in the main Simulink file through a Model²³ block. Model blocks reuse models as blocks in other models, creating a model hierarchy where the main Simulink file is the parent model.

The planning model needs some parameters, which have been condensed into a subsystem. This subsystem requires a Simulink bus for the parameters to work within the closed loop model. This bus was named “BusPlannerParams” and holds the planning

²³ <https://www.mathworks.com/help/simulink/slref/model.html>

parameters, as shown in Figure 4.14. The model also needs information from the map which also requires a Simulink bus. The bus that contains this type of information was named “BusMapInfo” and has the following fields: number of lanes, lane width, lane centers, number of global planning points (waypoints) and the global planning points (Figure 4.15). These buses were then saved in the Matlab file “CreateBusObjects.m”.

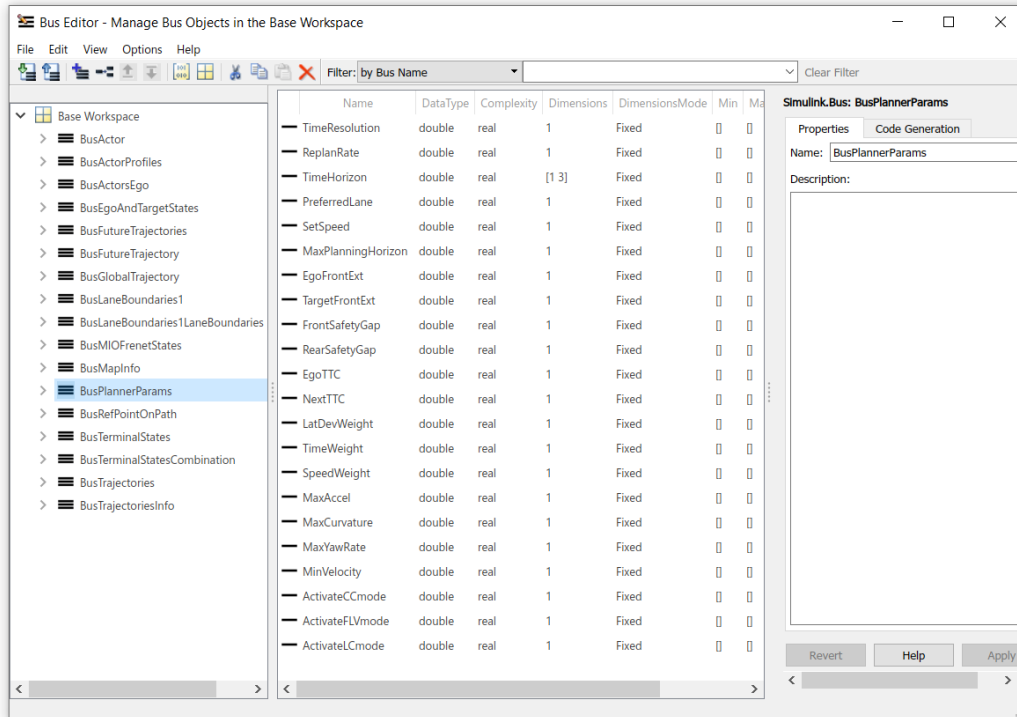


Figure 4.14 - Bus Editor showing the fields of the Simulink bus "BusPlannerParams".

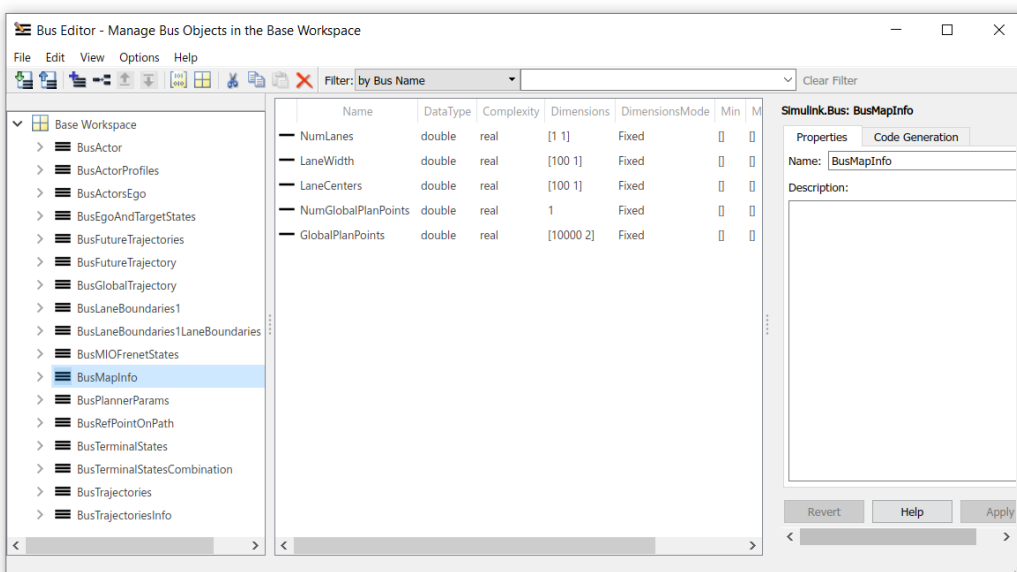


Figure 4.15 - Bus Editor showing the fields of the Simulink bus "BusMapInfo".

The trajectory planning model was structured as follows, where Figure 4.16 shows a synthesis of the process:

- I. Generate terminal states for the ego vehicle,
- II. Evaluate cost of the terminal states,
- III. Generate trajectories for the ego vehicle,
- IV. Check trajectories (kinematic) feasibility of the ego vehicle,
- V. Check Trajectories for Collision and Select the Optimal Trajectory.

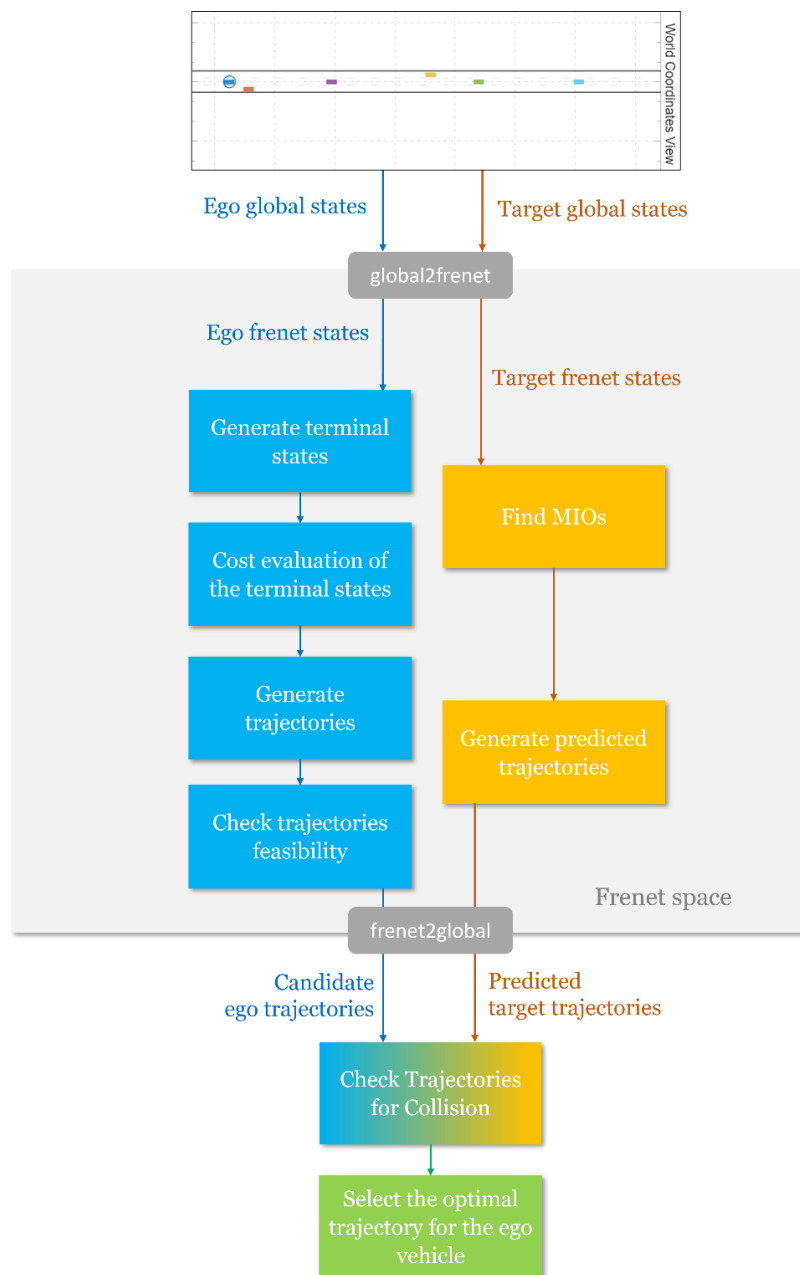


Figure 4.16 - Scheme of the local planning system for the ego vehicle to select the optimal trajectory at every sampled time.

I. Generate terminal states for the ego vehicle

The generation of the terminal states for the ego vehicle requires the knowledge of its position and the position of the target vehicles, in frenet states. Target vehicles are tracked in the scenario by the function “trackMIOs”, while the position of the ego vehicle is constantly updated by the function “updateEgoInfo”, to compute the distance between the main vehicle and the target vehicles that are in its current lane and adjacent right and left lanes.

The “trackMIOs” function outputs the frenet states for each target vehicle, which also requires a Simulink bus. The bus was named “BusMIOFrenetStates” and contains the following fields: number of target vehicles (or MIOs), their IDs and their states (Figure 4.17). This bus was exported and saved in the Matlab file “CreateBusObjects.m” as well.

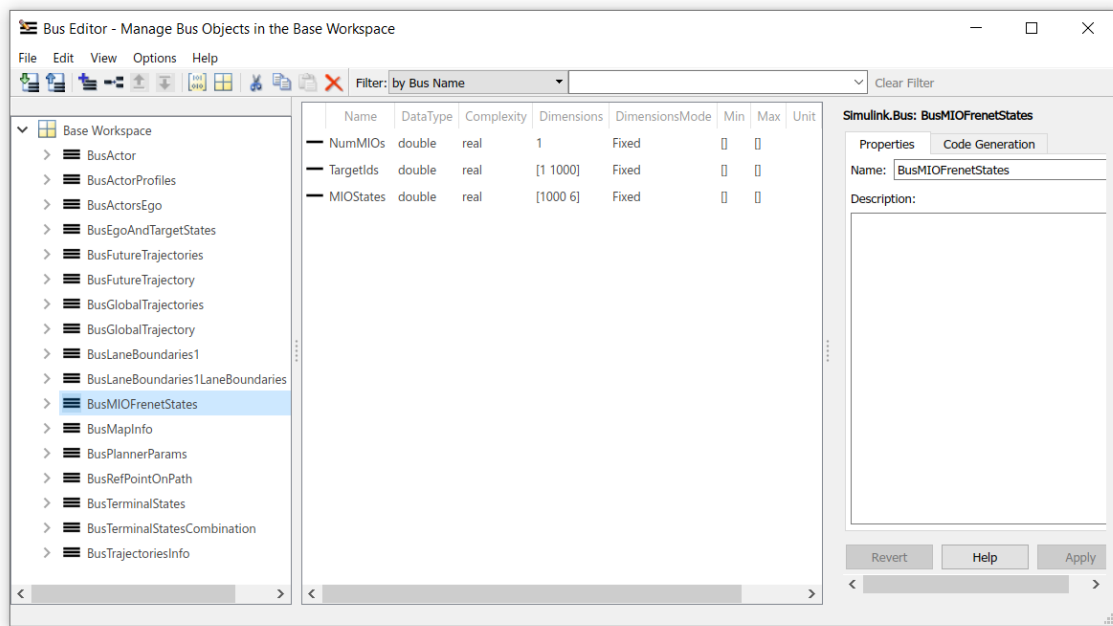


Figure 4.17-Bus Editor showing the fields of the Simulink bus “BusMIOFrenetStates”.

The function “updateEgoInfo” requires knowing the preferred lane for the vehicle, that is, the lane where the ego vehicle can travel with safety (collision-free). In this way, the preferred lane is calculated by the “FindPreferredLane” function which estimates this lane based on the Time-To-Contact (TTC) between MIOs and the ego vehicle

$$TTC = \frac{\text{Relative Distance}}{\text{Relative Velocity}} \quad (4.29)$$

Time-To-Contact is defined as the time for two vehicles to collide if both vehicles continue to drive at their current velocity and with the same heading angle. For lane change maneuvers, the calculation of the TTC is not trivial, since projections of future interactions between the ego vehicle and the target vehicles are required [119].

To estimate the magnitude of the distance and velocity vectors (relative distance and relative velocity) it was used the `cart2pol` function from Matlab, which uses polar coordinates generating the magnitude vectors and their respective angles. The angle of the relative velocity, calculated in equations (4.30) and (4.31), is the relative yaw angle, which is the angle from the lane centerline in relation to the longitudinal velocity, as illustrated in Figure 4.18.

$$v_x = V * \cos(\theta_{yaw}) \quad (4.30)$$

$$\theta_{yaw} = \theta_{vel} - \theta_{dist} \quad (4.31)$$

where v_x is the longitudinal velocity, θ_{yaw} is the relative yaw angle, θ_{vel} is the velocity angle and θ_{dist} is the relative distance angle.

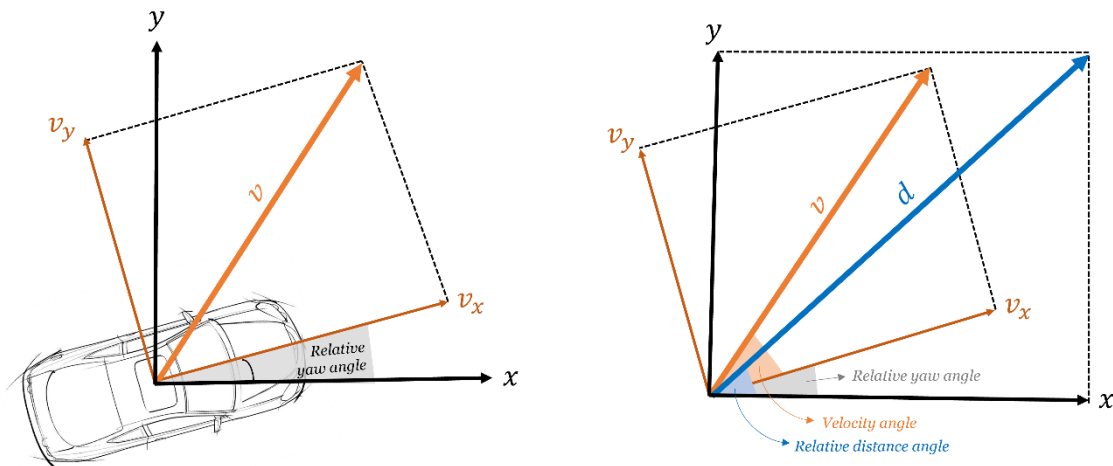


Figure 4.18 - Illustration of the relative velocity angle (relative yaw angle) and the relative distance angle (adapted from [120]).

When the planning model has both ego vehicle position and target vehicles position, it initializes the generation of the terminal states according to three driving modes supported by the planner.

(1) Cruise control: when this function is activated, cruise control terminal states are generated. Regarding longitudinal conditions, the longitudinal position is unrestricted,

and is, therefore, specified as not a number (NaN). Longitudinal velocity is set to the speed that the ego vehicle is traveling in the driving scenario (\dot{s}_{des}) and the terminal acceleration is set to zero. This function uses the ego vehicle's lateral velocity and the time horizon (t_{hor}) to predict the expected lateral offset (ego vehicle's lane) N-seconds in the future. With this predicted lateral offset, the lane center is calculated and becomes the terminal state's lateral deviation $l_{expLane}$. The lateral velocity and acceleration are set to zero.

$$cruiseControlState = [NaN, \dot{s}_{des}, 0, l_{expLane}, 0, 0, t_{hor}]$$

(2) Following Lead Vehicle: when this function is activated, terminal states that track a vehicle in front of the ego vehicle are generated. First, this function determines the ego vehicle's current lane through the estimation of its lane centers from the driving scenario. The function predicts the future state of each target vehicle, for each time horizon, to find the actors that occupies the same lane as the ego vehicle. In case there is a leading vehicle, the function searches for the vehicle that is closest to the main vehicle

$$closestLeadVehicleState = [s_{Lead}, \dot{s}_{Lead}, 0, l_{Lead}, \dot{l}_{Lead}, 0]$$

The terminal state of the ego vehicle is calculated by taking the position and speed of the lead vehicle and reducing the longitudinal position by the safety distance.

$$followLeadVehicleState = [(s_{Lead} - d_{safety}), \dot{s}_{Lead}, 0, l_{Lead}, \dot{l}_{Lead}, 0, t_{hor}]$$

(3) Lane change: when this function is activated, terminal states that transition the vehicle from the current lane to either adjacent lane are generated. First, this function determines the ego vehicle's current lane and then checks if adjacent lanes exist and each future ones are available. For each valid adjacent lane, the terminal state is defined in the same manner as the cruise control behavior, with the terminal velocity is set to the current speed that the ego vehicle is travelling in the driving scenario (\dot{s}_{cur}).

$$laneChangeState = [NaN, \dot{s}_{cur}, 0, l_{desLane}, 0, 0, t_{hor}]$$

Each driving mode outputs the number of terminal states and their own values. Therefore, to hold these two information in the same output, it was necessary to create a

Simulink bus and exported to the Matlab file “CreateBusObjects.m”. This bus was named “BusTerminalStates”.

When the planner has all terminal states, it concatenates them to ensure that the planned continuous trajectory is feasible and also natural. This operation is made in the function “TerminalStatesConcatenation” in which the output variable is composed by the following fields: the number of resulting combinations, the actual combinations (with all the states) and the driving mode (CC, LFV and LC). Both combinations and driving mode hold a maximum of 60 terminal state values. For this output, the Simulink bus “BusTerminalStatesCombination” was created (Figure 4.19) and exported to the Matlab file “CreateBusObjects.m”.

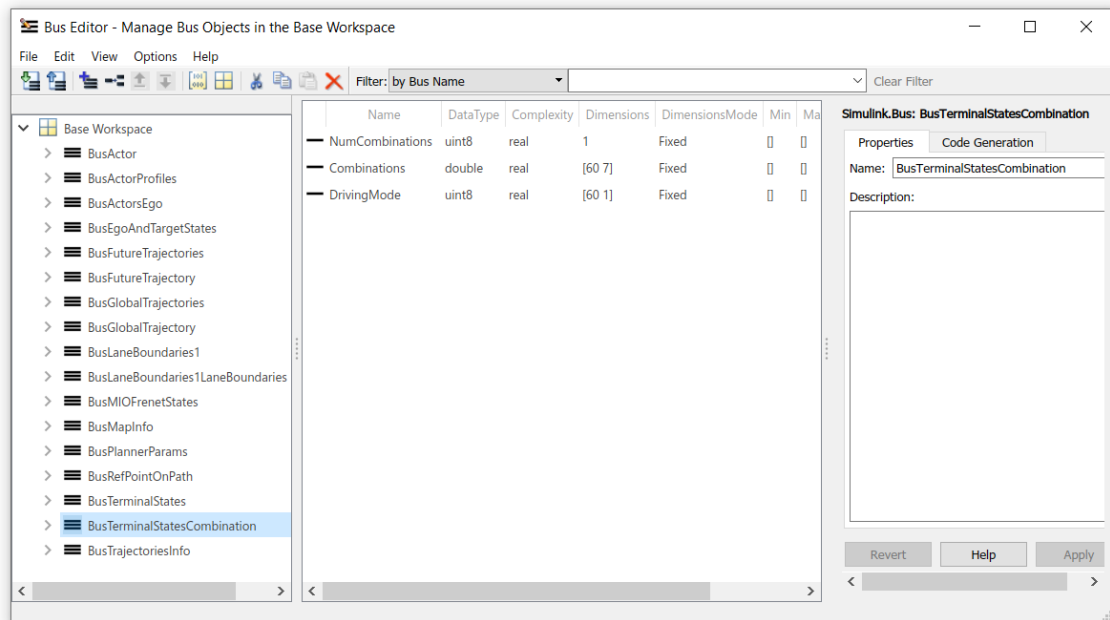


Figure 4.19-Bus Editor showing the fields of the Simulink bus “BusTerminalStatesCombination”.

II. Evaluate cost of the terminal states

Trajectory evaluation is done after the generation of the terminal states by evaluating their costs and finding the trajectory with the minimum cost (the optimal one). The function “EvaluateTSCost” defines the cost as the combination of three weighted sums

$$C_{TOTAL} = C_{latDev} + C_{time} + C_{speed} \quad (4.32)$$

The lateral deviation cost presented in the equation (4.33) is achieved by the lateral deviation difference penalized with a positive weight for all the states that deviate from the center of a lane. The lateral deviation difference, given by the equation (4.34), uses the minimum argument of the lateral deviation vector. This vector is calculated by the difference between the lateral deviation of the ego current lane and each terminal states lateral deviation.

$$C_{latDev} = W_{\Delta L} * \Delta L \quad (4.33)$$

$$\Delta L = \text{argmin} (|L_{egoLane} - L_{terminalState}|) \quad (4.34)$$

where C_{latDev} is the lateral deviation cost, $W_{\Delta L}$ is the lateral deviation weight, ΔL is the lateral deviation difference, $L_{egoLane}$ is the lateral deviation of the lane (lane center) in which the ego vehicle is travelling and $L_{terminalState}$ is the lateral deviation of all terminal states.

The time cost, given by the equation (4.35), is computed by prioritizing motions that occur over a longer interval, resulting in smoother trajectories. In this way, the time of the terminal states is multiplied with a negative weight.

$$C_{time} = W_{\Delta t} * \Delta t \quad (4.35)$$

where C_{time} is the time cost, $W_{\Delta t}$ is the time weight and Δt is the time interval of the terminal states (time horizon).

The velocity cost, expressed by the equation (4.36), is determined by the multiplication of the velocity difference for each terminal states, multiplied by a positive weight in order to prioritize motions that maintain the ego vehicle velocity from the driving scenario - resulting in less dynamic maneuvers. The velocity difference is achieved by the contrast between the ego vehicle velocity (from the scenario) and its terminal states velocity – equation (4.37).

$$C_{velocity} = W_{\Delta v} * \Delta v \quad (4.36)$$

$$\Delta v = v_{terminalStates} - v_{ego} \quad (4.37)$$

where $C_{velocity}$ is the velocity cost, $W_{\Delta v}$ is the velocity weight, Δv is the velocity difference, $v_{terminalStates}$ is the velocity of all terminal states and v_{ego} is the ego vehicle speed.

The three weighted sums are calculated and sorted in ascending order, placing terminal states with the lowest costs in the first positions of choice for the ego vehicle. Once the terminal states are sorted, it is possible to generate trajectories for the main vehicle. For that, a system object named “EgoTrajectoryGenerator” was created, where a trajectoryGeneratorFrenet²⁴ object is used. This object generates trajectories using fifth-order polynomials relative to the reference path, which is specified as a referencePathFrenet²⁵ object. This last object fits a smooth, piecewise and continuous curve for the specified set of waypoints from the reference path, resulting in a trajectory that smoothly matches the ego vehicle's velocity over the time horizon.

The trajectoryGeneratorFrenet object generates global trajectories for the ego vehicle, calculated according to the minimal cost previously estimated. All global trajectories are updated at every replan cycle based on the information received from the upstream blocks. The default replan cycle is one second and is created by a pulse generator. Thus, the “EgoTrajectoryGenerator” system object output has the following fields: number of the global trajectories generated, if each global trajectory is new, the index of each global trajectory and the characteristics of a global trajectory for each trajectory. To support this output, the Simulink bus “BusGlobalTrajectories” (Figure 4.20) was created.

Each global trajectory sustains the trajectory waypoints and respective times, the number of these waypoints, checks if the trajectory is valid for the ego vehicle, verify if the trajectory is evaluated and if it collides, and the driving mode used. In this way, the Simulink bus named “BusGlobalTrajectory” was created (Figure 4.21). The trajectory waypoints has a maximum number given by the following equation

$$maxTrajPoints = \frac{t_{hor}}{t_{res}} + 1 \quad (4.38)$$

where t_{hor} is the time horizon and t_{res} is the time resolution. A waypoint must be added to the equation (4.38) since it is necessary to consider the ego vehicle's waypoint (located in its CG).

²⁴ <https://www.mathworks.com/help/nav/ref/trajectorygeneratorfrenet.html>

²⁵ <https://www.mathworks.com/help/nav/ref/referencepathfrenet.html>

Both buses were exported and saved in the Matlab file “CreateBusObjects.m”.

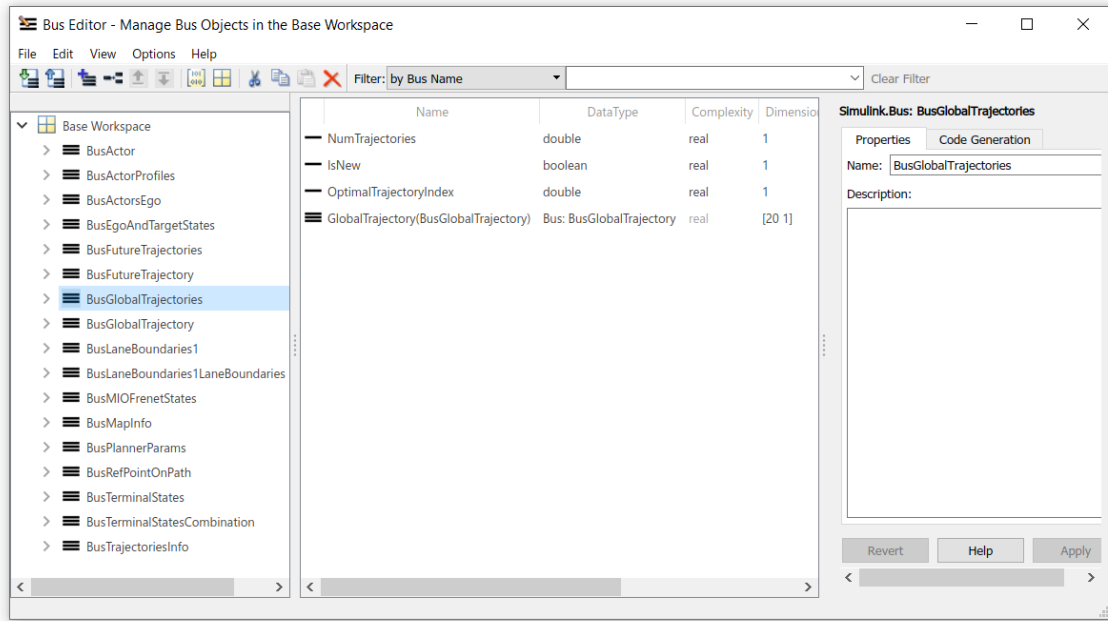


Figure 4.20 - Bus Editor showing the fields of the Simulink bus “BusTrajectories”.

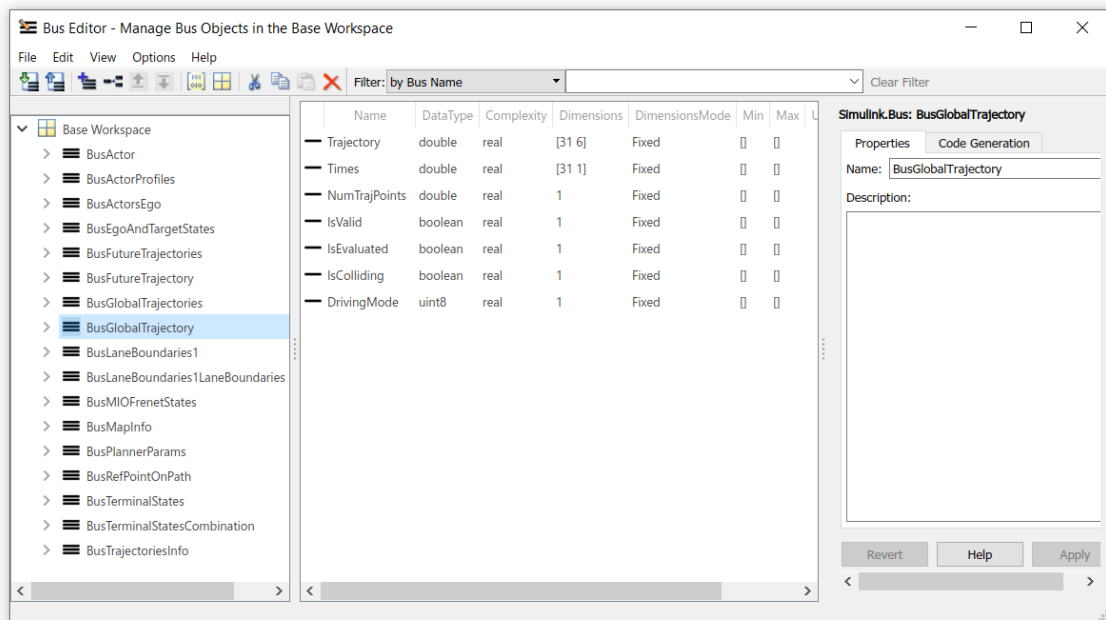


Figure 4.21 - Bus Editor showing the fields of the Simulink bus “BusGlobalTrajectory”.

III. Check trajectories kinematic feasibility of the ego vehicle

In addition of having terminal states with minimal cost, an optimal trajectory also needs to take into account the kinematic feasibility for a good ride quality. For this purpose, the "checkTrajectoryFeasibility" function was created to guarantee that all generated global trajectories eliminate infeasible maneuvers, taking into account some important kinematic constraints. This function defines bounds as follows: (1) maximum values of curvature, acceleration, yaw rate, and (2) minimum value of velocity.

Constraining curvature, acceleration, yaw rate to minimum values results in a smoother driving experience. Regarding the velocity, as the ego vehicle only travels in forward motion, it is necessary to assign it a minimum speed. By restricting these features, it is possible to eliminate all kinematically infeasible trajectories that would otherwise be performed by the ego vehicle.

IV. Check Trajectories for Collision and Select the Optimal Trajectory

The final step in the planning process is choosing the best trajectory: the collision-free trajectory. Collision checking is an expensive operation, so for the simulation optimization it is saved for after the cost evaluation and the analysis of constraints. After these two tasks the remaining trajectories can then be checked for collision until a collision-free path has been found or all trajectories have been evaluated.

To accomplish this task, it is necessary to know all the future trajectories that the ego vehicle can perform in order to avoid possible collisions with the target vehicles. For that, a system object named "EgoTrajectoryPredictor" was created. This system object uses the referencePathFrenet object to smooth the path along the waypoint set. The system object uses the current MIO position to predict the future trajectories of MIO by assuming them to be moving with a constant velocity during the simulation time and computing ego states accordingly. It returns the predicted trajectories of the MIOs with the following fields: number of trajectories, all the target IDs and the future trajectories of all target vehicles (have their own features). In the Bus Editor, a Simulink bus containing these field was created and named "BusFutureTrajectories". After that it was exported and saved in the Matlab file "CreateBusPredictedTrajectories.m", where the number of target vehicles was replaced for a variable, since this is a variable parameter for each scenario.

Each single predicted trajectory holds the ID of the target actor travelling on it, its number of waypoints and the frenet states for each point. The maximum number of frenet states for each waypoint on that trajectory was set to 255 (rows), since the target vehicle ID and the number of waypoints are uint8²⁶ data types. In this way, the Simulink bus named “BusFutureTrajectory” was created and saved in the Matlab file “CreateBusPredictedTrajectories.m” as well (Figure 4.22).

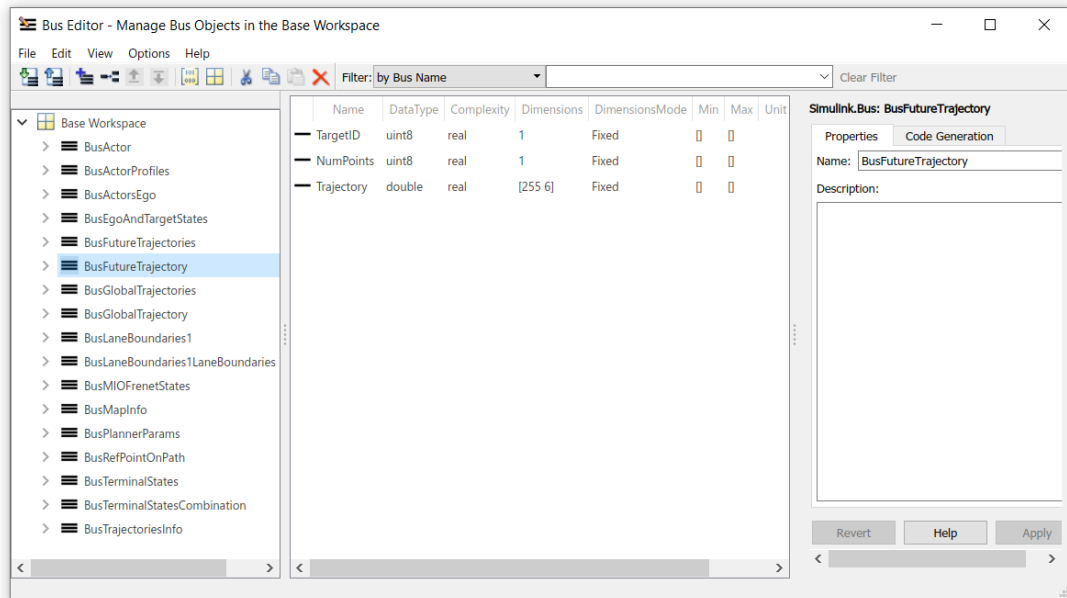


Figure 4.22 - Bus Editor showing the fields of the Simulink bus “BusFutureTrajectory”.

When the planning system has the global valid trajectories for the ego vehicle and the predicted trajectories of the target vehicles, the system checks for collision. For that, is created a system object named “CollisionChecker” that uses the dynamicCapsuleList²⁷ object, This object manages two separated lists of capsule-based collision objects, ego bodies and obstacles, in 2-D space. Each collision object in the two lists has three key elements: ID for each object (EgoIDs for ego vehicles and ObstacleIDs for target vehicles), their states which is a three-element row vector in the form of $[X Y \theta]$ and their geometry (Figure 4.23).

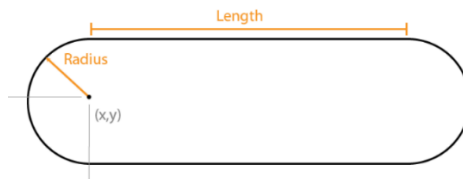


Figure 4.23 - Geometry of a capsule-based collision object.

²⁶ https://www.mathworks.com/help/matlab/matlab_prog/integers.html

²⁷ <https://www.mathworks.com/help/nav/ref/dynamiccapsulelist.html>

The Driving Scenario Designer allow the configuration of different geometries for the actors, in which for collision checking it represents an important and indispensable feature to take into account. In “CollisionChecker” system object, the main vehicle is defined with default vehicle dimensions of a sedan, shown in Figure 4.24, since these values are the same as the ego vehicle of the driving scenario.

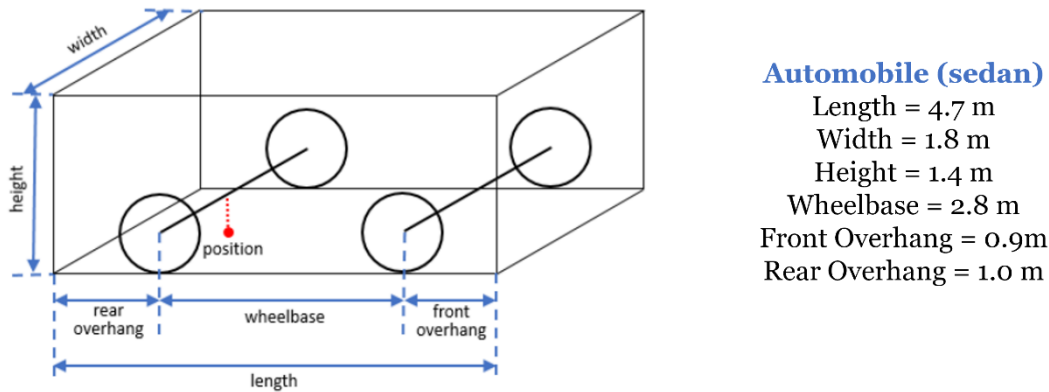


Figure 4.24 - Vehicle dimensions of a sedan vehicle.

Regarding the target vehicles, it was used the actorProfiles²⁸ function from Matlab which is a structure with the fields presented in Table 4.3. To be able to access these vehicle characteristics as variables, it was necessary to create a Simulink bus with these parameters which was named "BusActorProfiles " and saved in the Matlab file “CreateBusPredictedTrajectories.m”.

The “CollisionChecker” system object updates constantly the two capsule lists according to the predicted trajectories of MIOs and the sampled ego trajectories. It checks for collisions between ego and target vehicles during replan cycles and in between replan cycles. The first valid collision-free trajectory obtained is chosen as the optimal trajectory. This means that all the trajectory information outputted by this system block needs to be a Simulink bus with:

- valid trajectory information for the ego vehicle (bus “BusTrajectories”),
- predicted trajectories for target vehicles (bus “BusFutureTrajectories”).

²⁸ <https://www.mathworks.com/help/driving/ref/drivingscenario.actorprofiles.html>

Table 4.3 - Vehicle profile structure.

Field	Description	Value	Units
ActorID	Scenario-defined actor identifier	Positive integer	-
ClassID	Classification identifier	Nonnegative integer *	-
Length	Length of actor	Positive real-valued scalar	Meter
Width	Width of actor	Positive real-valued scalar	Meter
Height	Height of actor	Positive real-valued scalar	Meter
OriginOffset	Offset of the actor's rotational center (origin) from its geometric center	Real-valued vector of the form [x, y, z]	Meter
MeshVertices	Mesh vertices of actor	N-by-3 real-valued matrix of vertices. Each row in the matrix defines a point in 3-D space	-
MeshFaces	Mesh faces of actor	M-by-3 matrix of integers. Each row of MeshFaces represents a triangle defined by the vertex IDs, which are the row numbers of vertices	-
RCSPattern	Radar cross-section pattern of actor	numel(RCSElevationAngles)-by-numel(RCSAzimuthAngles) real-valued matrix	Decibel per square meter
RCSAzimuthAngles	Elevation angles corresponding to rows of RCSPattern	Vector of values in the range [-180, 180]	Degree
RCSElevationAngles	Elevation angles corresponding to rows of RCSPattern	Vector of values in the range [-90, 90]	Degree

* The default ClassID of zero is reserved for an object of an unknown or unassigned class, the ClassID of actors set to 1 is a car, 2 is for trucks, 3 is for bicycles, 4 is for pedestrians, 5 is for jersey barrier and 6 is for guardrail.

In this way, the Simulink bus named “BusTrajectoriesInfo” was created containing the information of the ego trajectory and the information of the future trajectories of the target vehicles, as shown in Figure 4.25. It has exported and saved in the Matlab file “CreateBusPredictedTrajectories.m”.

The last step of the local trajectory planning is to compute the next state of the ego vehicle, required by the control system. A MPC controller includes a nominal operating point at which its model applies to obtain the linear-time-invariant approximation. The function “EgoCurrentState” estimates the next state (waypoint on path) of the ego vehicle based only on its current pose and outputs that waypoint

information - updated at each time step as the ego's operating waypoint keeps changing. According to that information, the control system adjusts the longitudinal and lateral characteristics of the ego vehicle. To support all function output fields, the Simulink bus "BusRefPointOnPath" was created (Figure 4.26). This is the last bus necessary to create and was also saved in the Matlab file "CreateBusPredictedTrajectories.m".

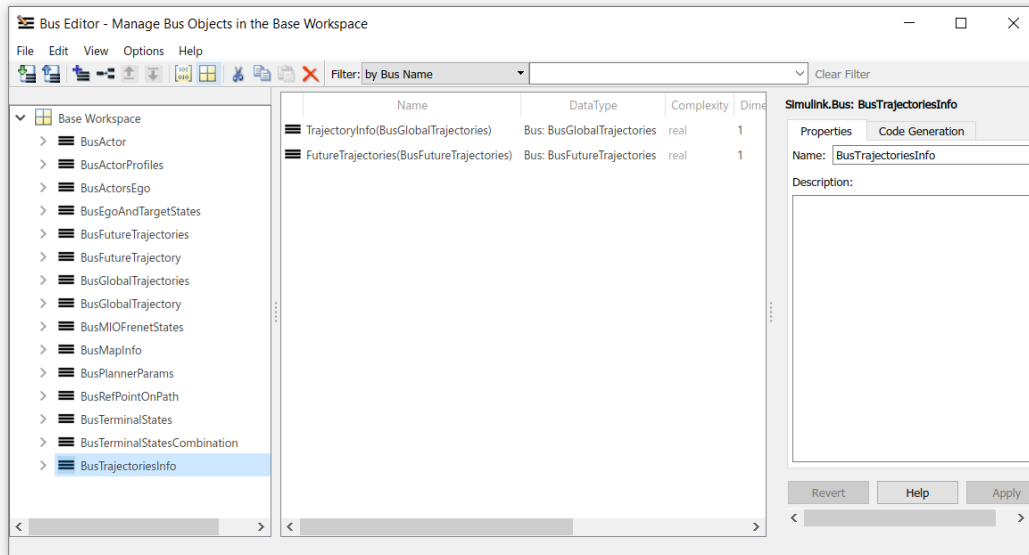


Figure 4.25 - Bus Editor showing the fields of the Simulink bus "BusTrajectoriesInfo".

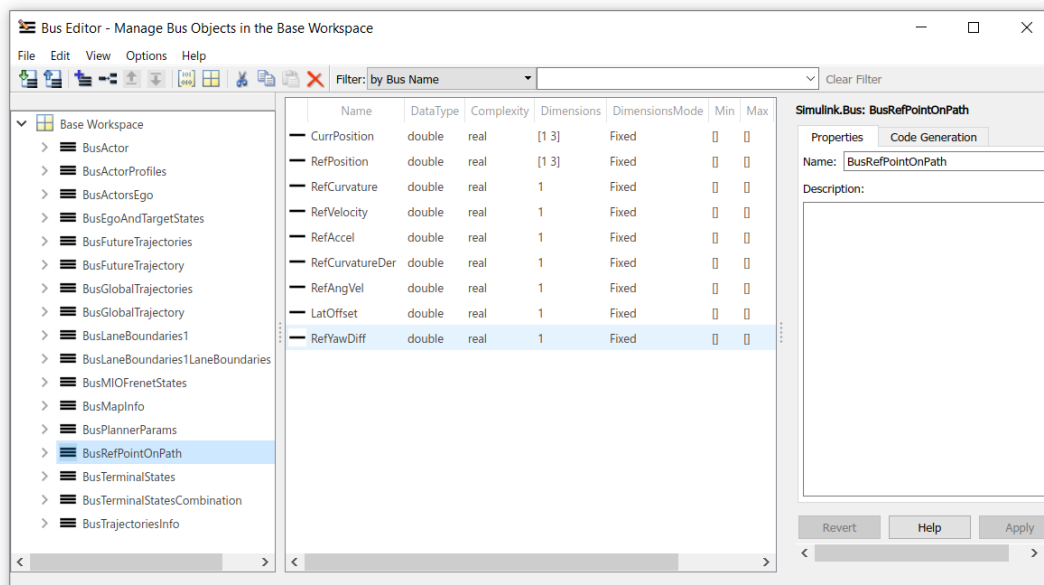
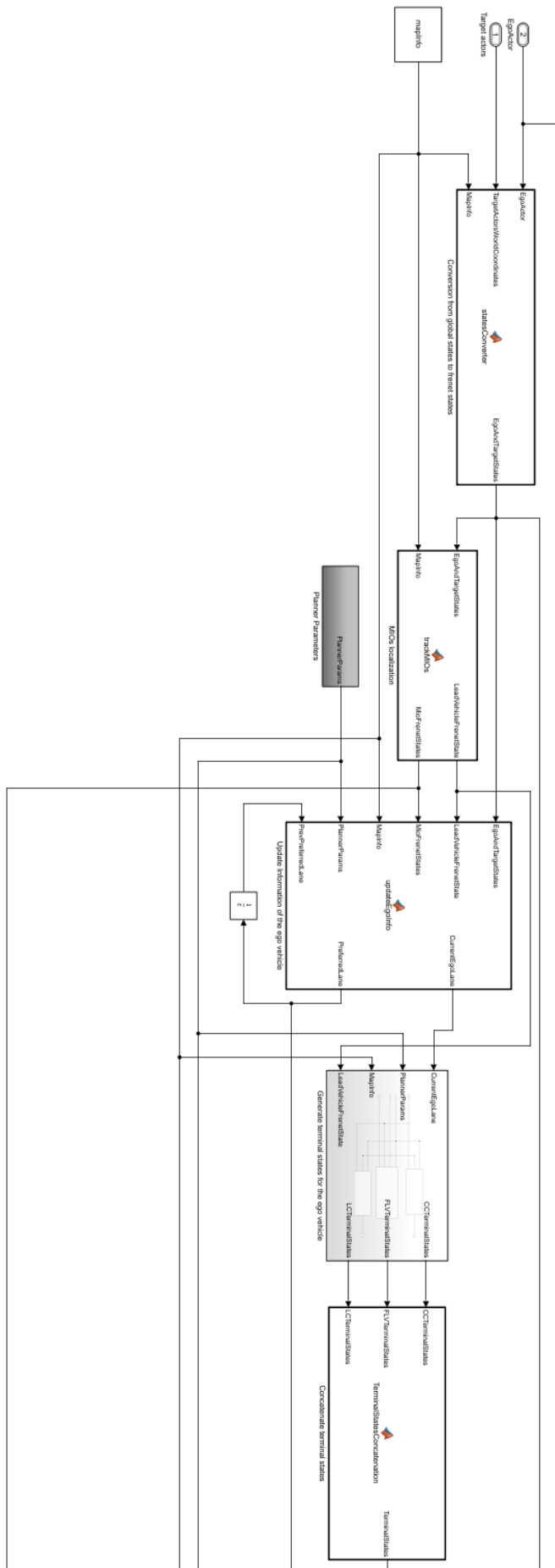


Figure 4.26 - Bus Editor showing the fields of the Simulink bus "BusRefPointOnPath".

The vehicle model planning system designed in the Simulink file can be found in Figures 4.27, 4.28, 4.29 and 4.30.



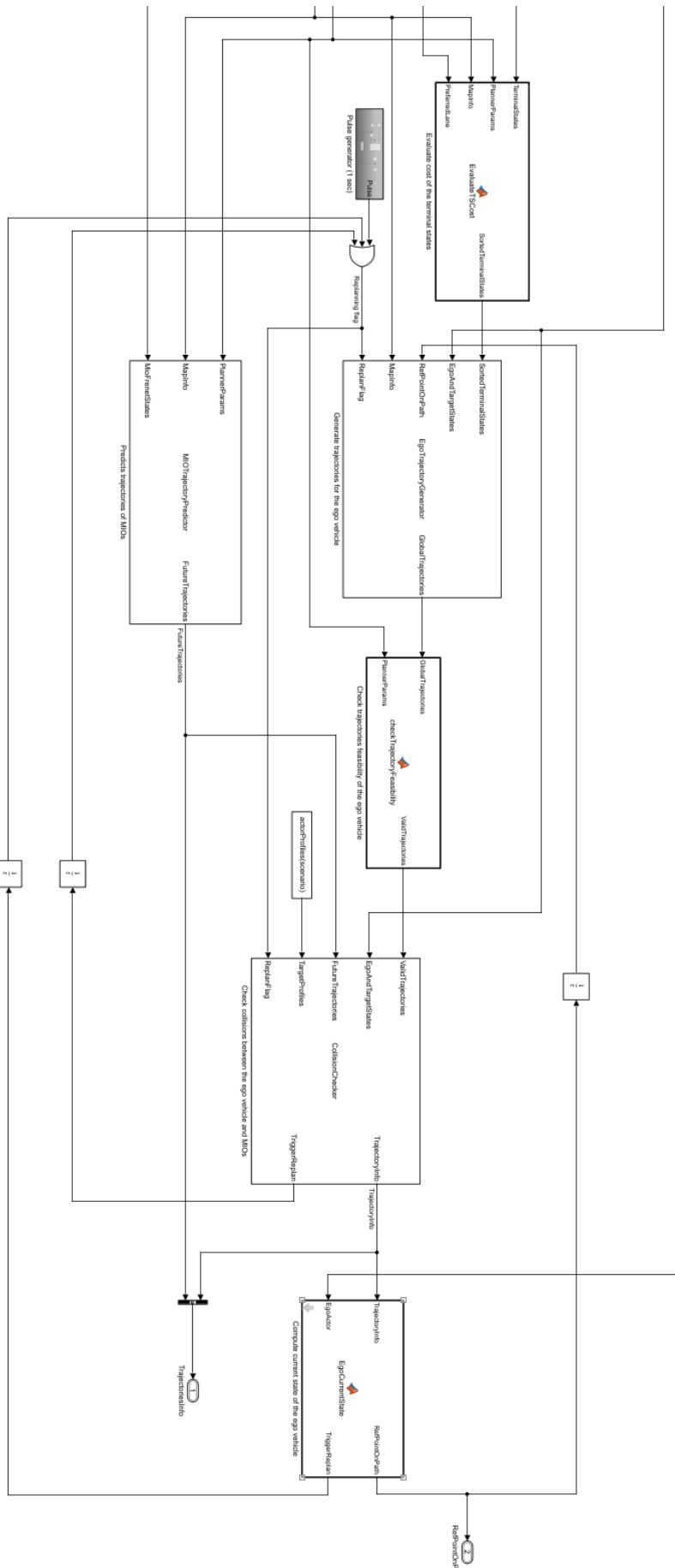


Figure 4.27 - Scheme of the planning system of the Simulink model.

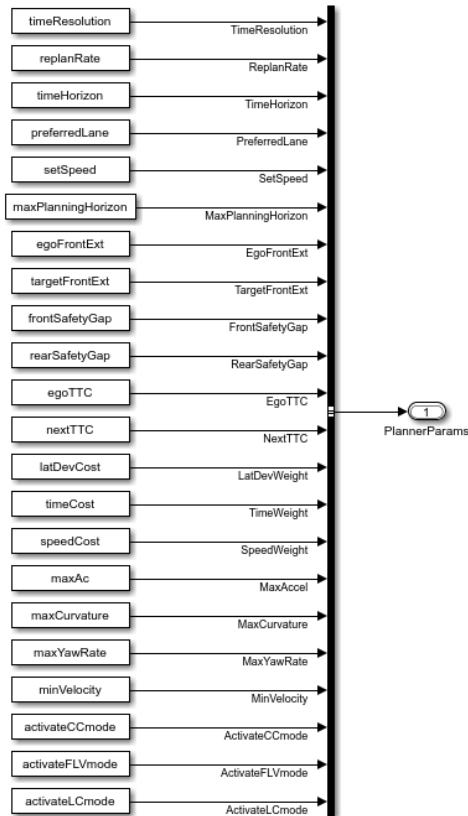


Figure 4.28 – “Planner parameters” subsystem.

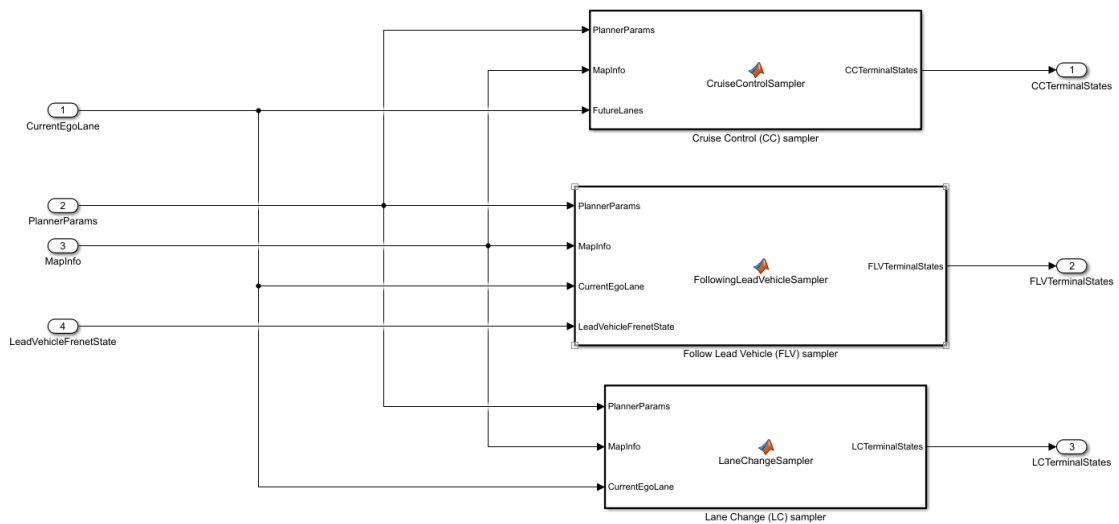


Figure 4.29 – “Generate terminal states for the ego vehicle” subsystem.

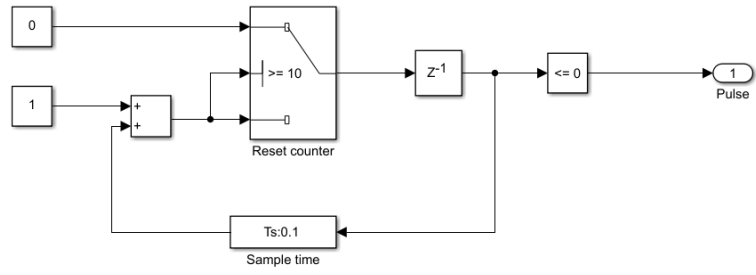


Figure 4.30 - "Pulse generator" subsystem.

4.5 Vehicle control

For path-following, the control system of this vehicle model uses an adaptive model predictive control (MPC) to keep the ego vehicle travelling along the center of a road (LKA) while maintaining a safe distance from a lead vehicle (ACC) to track the waypoints defined previously in path planning (upper level control). For that purpose, the Path Following Control System²⁹ block from the Model Predictive Control Toolbox³⁰ of Matlab was used, since this block combines the capabilities of the Lane Keeping Assist System³¹ and the Adaptive Cruise Control System³² blocks, as shown in Figure 4.31. The adaptive MPC controller is the core of this control system that computes optimal control actions while satisfying velocity, acceleration, and steering angle constraints.

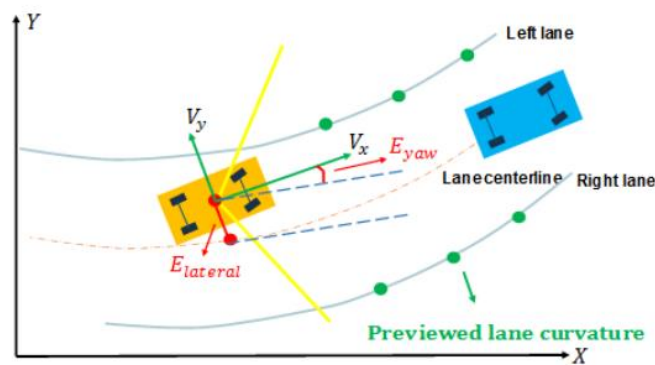


Figure 4.31 - Schematic of lane curvature estimation for LKA²⁹.

4.5.1 Adaptive model predictive control

The model predictive control predicts future behavior using a Linear-Time-Invariant (LTI) dynamic model. In a traditional MPC, the LTI prediction accuracy might degrade since the ego characteristics vary dramatically with time. Adaptive MPC can address this degradation by adapting the prediction model for changing operating conditions because it allows the models parameters to evolve with time. The adaptive MPC uses a fixed plant model structure that changes over a finite prediction horizon [121].

²⁹ <https://www.mathworks.com/help/mpc/ref/pathfollowingcontrolsystem.html>

³⁰ <https://www.mathworks.com/products/model-predictive-control.html>

³¹ <https://www.mathworks.com/help/mpc/ref/lanekkeepingassistssystem.html>

³² <https://www.mathworks.com/help/mpc/ref/adaptivecruisecontrolsystem.html>

The inputs for the plant are separated to indicate that u correspond to the front wheel steering angle and acceleration/deceleration command of the vehicle (controlled output of MPC), while v indicates the previewed curvature (measured disturbance). The structure of the MPC plant model is as follows

$$x(k + 1) = Ax(k) + B_u u(k) + B_d v(k) \quad (4.39)$$

$$y(k) = Cx(k) \quad (4.40)$$

where x is the state, k is time index (current control interval), u is the manipulated input (inputs that are adjusted by the MPC controller), v is the measured disturbance input, y is the output of the system, A is the state matrix, B_u and B_d are the input matrices corresponding to inputs u and v respectively, and C is the output matrix.

4.5.2 Path following control

In a cruise control system, the speed of the vehicle is controlled to a desired value using the throttle control input. When a preceding vehicle is detected, the ACC system determines whether or not the ego vehicle can continue to travel safely at the desired speed. The upper level and lower level controller of the longitudinal control system architecture for the cruise control is structured as shown in Figure 4.32. [122]

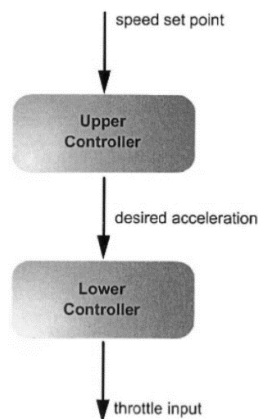


Figure 4.32 - Structure of cruise control system [122].

The upper level controller determines the desired longitudinal acceleration for the ego vehicle, where the speed of the vehicle should converge to the speed set by the

driver – in this vehicle model, converge to the speed defined for the ego vehicle in the driving scenario. The lower level controller (Figure 4.33) determines the throttle input required to track the desired acceleration, which due to the finite bandwidth associated with this controller, the vehicle is expected to track its desired acceleration imperfectly and thus it is required to incorporate a lag in tracking desired acceleration

$$\dot{x} = \frac{1}{\tau s + 1} \ddot{x}_{des} \quad (4.41)$$

where \dot{x} is the throttle input, \ddot{x}_{des} is the desired acceleration and τ is desired acceleration time constant, assumed to be $\tau = 0.5$ sec for analysis and simulation.

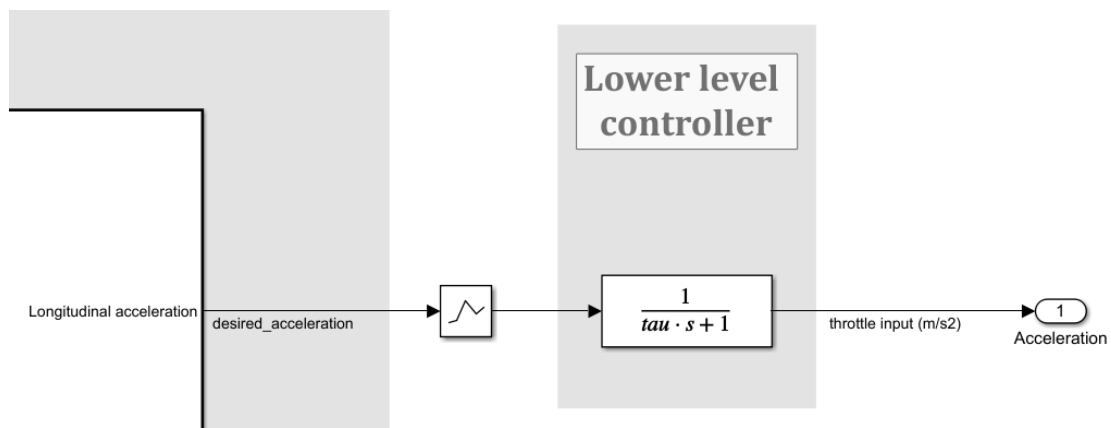


Figure 4.33 – Lower level controller of the Simulink file.

The predictive model of the adaptive cruise control system has the following predictive state-space model

$$A_1 = \begin{bmatrix} -\frac{1}{\tau} & 0 \\ 1 & 0 \end{bmatrix} \quad (4.42)$$

$$B_1 = \begin{bmatrix} \frac{1}{\tau} \\ 0 \end{bmatrix} \quad (4.43)$$

$$C_1 = [0 \ 1] \quad (4.44)$$

$$D_1 = 0 \quad (4.45)$$

In a lane-keeping assist system, the lateral deviation and relative yaw angle between the centerline of a lane and the ego car is measured, as well as the current lane

curvature and curvature derivative. The curvature in front of the ego vehicle (previewed curvature) can be calculated from these two last features. Thus, the previewed lane curvature [123] is achieved by assuming that $K(s)$ is a linearly varying function, so that the evolution of the ego vehicle state at each waypoint is calculated as

$$K_{i+1} = K_i + C_{i+1} + s_{i+1} \quad (4.46)$$

since,

$$s = V_x * t_{hor} \quad (4.47)$$

then,

$$s_{i+1} = V_x * t_{pred} \quad (4.48)$$

where K_{i+1} is the previewed curvature, K_i is the current curvature, C_{i+1} is the curvature slope (constant by definition), s_{i+1} is the predicted arc-length of the road, V_x is the longitudinal velocity and t_{pred} is the prediction time that is equal to the prediction horizon multiplied by the simulation sample time (T_s), represented by the gain “K” in Figure 4.34.

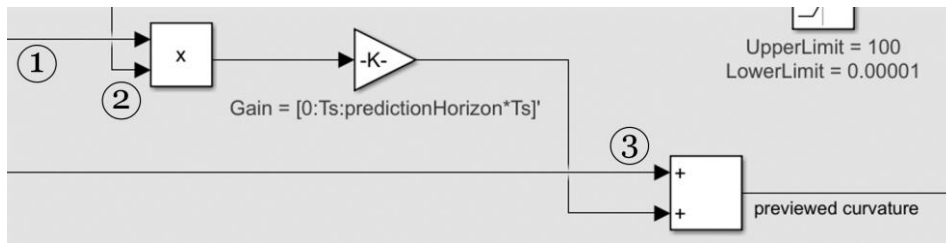


Figure 4.34 - Calculation of previewed curvature: 1) curvature derivative, 2) longitudinal velocity and 3) curvature.

In this way, the predictive model of the lane-keeping system has the following predictive state-space model

$$A_2 = \begin{bmatrix} \frac{-2(C_f + C_r)}{mV_x} & \frac{-V_x - 2(C_f l_f - C_r l_r)}{mV_x} \\ \frac{-2(C_f l_f - C_r l_r)}{I_{zz}V_x} & \frac{-2(C_f (l_f)^2 + C_r (l_r)^2)}{I_{zz}V_x} \end{bmatrix} \quad (4.49)$$

$$B_2 = \begin{bmatrix} \frac{2C_f}{m} \\ \frac{2C_f l_f}{I_{zz}} \end{bmatrix} \quad (4.50)$$

$$C_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (4.51)$$

$$D_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (4.52)$$

The LKA system keeps the ego car travelling along the centerline of the lanes on the road by adjusting the front steering angle of the ego vehicle, keeping lateral deviation and relative yaw angle close to zero.

As the path following control system combines these two previous systems, its predictive model results in the following predictive state-space model

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \quad (4.53)$$

$$B = \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} \quad (4.54)$$

$$C = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix} \quad (4.55)$$

$$D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \quad (4.56)$$

Therefore, the output of this combined model are the longitudinal acceleration and the steering angle. All values used in the controller are shown in Figure 4.35 and the vehicle model control system designed in the Simulink file can be found in Figure 4.36.

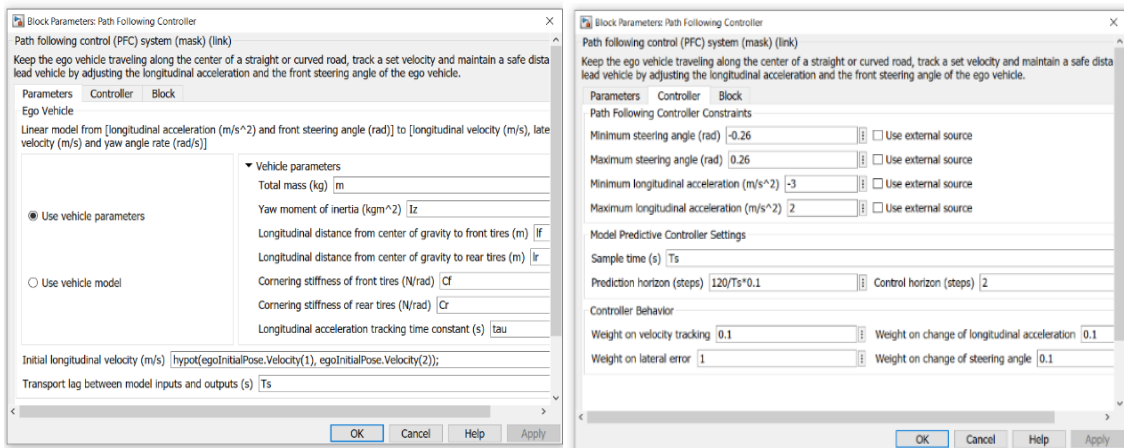


Figure 4.35 - Values of the path following control system.

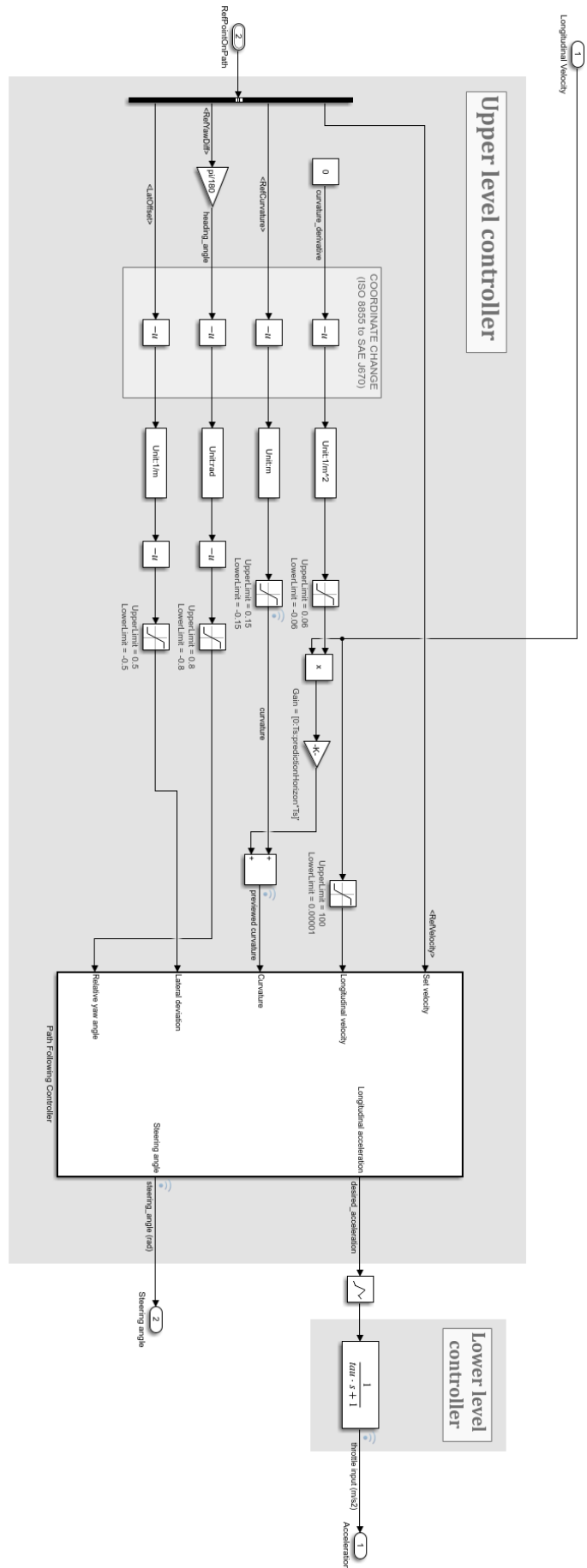


Figure 4.36 - Scheme of the control system of the Simulink model.

Chapter 5

Results and discussion

In this chapter the software-in-the-loop simulation procedure is presented with its results, and discussed in detail about the performance of the system. The results obtained in the simulations are evaluated for model validation.

To simulate the performance of the main vehicle in a low-traffic scenario and in a dense traffic scenario, and to reduce test cases, three driving scenarios were created. For the scenario with little traffic (driving scenario 1), only a two-lane road was tested since for little traffic the main vehicle has more space to make maneuvers like lane changes to avoid collisions, and therefore presents a very similar behavior and performance for roads with more than two lanes. On the other hand, in a scenario with dense traffic, the main vehicle has less room to maneuver. For this reason, the model developed was simulated for roads with three and four lanes - driving scenarios 2 and 3, respectively.

The simulation environment, as mentioned in Chapter 4, is defined with default lane lengths of 3.6 meters, where their respective centers are different for each driving scenario, as these depend on the number of lanes (see Figure 5.1). In the driving scenario with 2 lanes, the center of the right lane holds the value of -1.8 meters (half of the lateral length of the lane), while the center of the left lane holds the value of 1.8 meters. In the three-lane scenario the center of the right lane holds the value of -3.6 meters, the center of the middle lane has the value of 0 meters, and the center of the left lane holds the value of 3.6 meters. The four-lane scenario has two right lanes with lane centers of -5.4 meters (rightmost lane) and -1.8 meters, and two left lanes with lane centers of 1.8 meters and 5.4 meters (leftmost lane). Also, the first lane (lane 1) is always the rightmost lane of the road.

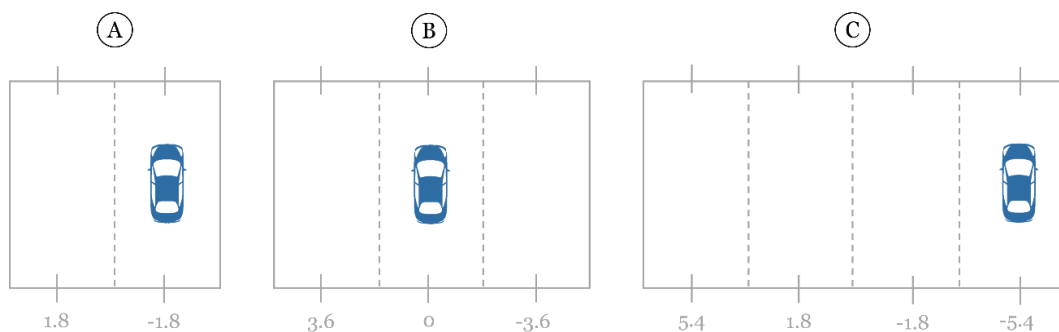


Figure 5.1 - Lane centers for: A) 2 lanes, B) three lanes and C) four lanes.

As all scenarios will be tested according to the performance of the ego vehicle, all other conditions such as relative position and speed of other vehicles in the scenario are defined in relation to it. The surrounding traffic is defined with a constant velocity and with a pre-defined path. Tables 5.1, 5.2 and 5.3 show the velocity and the positions (waypoints) defined for the target vehicles in each scenario.

Table 5.1 - Values defined for the target vehicles for the driving scenario 1.

Vehicle ID	Velocity	Initial position	Final position
1	23	[10 1.8 0]	[800 1.8 0]
2	17	[70 -1.8 0]	[800 -1.8 0]
3	22	[75 1.8 0]	[800 1.8 0]

Table 5.2 - Values defined for the target vehicles for the driving scenario 2.

Vehicle ID	Velocity	Initial position	Final position
1	17	[7 -3.6 0]	[800 -3.6 0]
2	17	[10 3.6 0]	[800 3.6 0]
3	19	[30 -3.6 0]	[800 -3.6 0]
4	15	[40 0 0]	[800 0 0]
5	17	[50 3.6 0]	[800 3.6 0]
6	15	[80 0 0]	[800 0 0]

Table 5.3 - Values defined for the target vehicles for the driving scenario 3.

Vehicle ID	Velocity	Initial position	Final position
1	20	[1 5.4 0]	[800 5.4 0]
2	17	[7 -1.8 0]	[800 -1.8 0]
3	18	[13 1.8 0]	[800 1.8 0]
4	21	[15 5.4 0]	[800 5.4 0]
5	21	[30 1.8 0]	[800 1.8 0]
6	24	[33 5.4 0]	[800 5.4 0]
7	16	[50 -5.4 0]	[800 -5.4 0]
8	18	[55 -1.8 0]	[800 -1.8 0]
9	22	[100 5.4 0]	[800 5.4 0]

All driving scenarios were created for the ego vehicle to achieve the most desired performance: make a lane change when a critical scenario arises, since this is the driving mode that uses the maximum capability of the implemented system to avoid collisions. This means that all driving scenarios are critical scenarios and, consequently, all scenarios perform at least one lane change (successfully or not).

Driving scenario 1 was built to evaluate the performance of the ego vehicle over time, on a two-lane road with little traffic, in which the main vehicle was initially placed in lane 1 (the rightmost lane of the road). On the other hand, driving scenario 2 was created to evaluate the vehicle performance on a three-lane road with dense traffic, with the main vehicle placed in lane 2 (middle lane). Driving scenario 3 aims to evaluate the vehicle's performance in a scenario of dense traffic on a four-lane road, where the vehicle was placed in lane 1. Figure 5.2 shows the ego vehicle in its initial lane in each scenario.

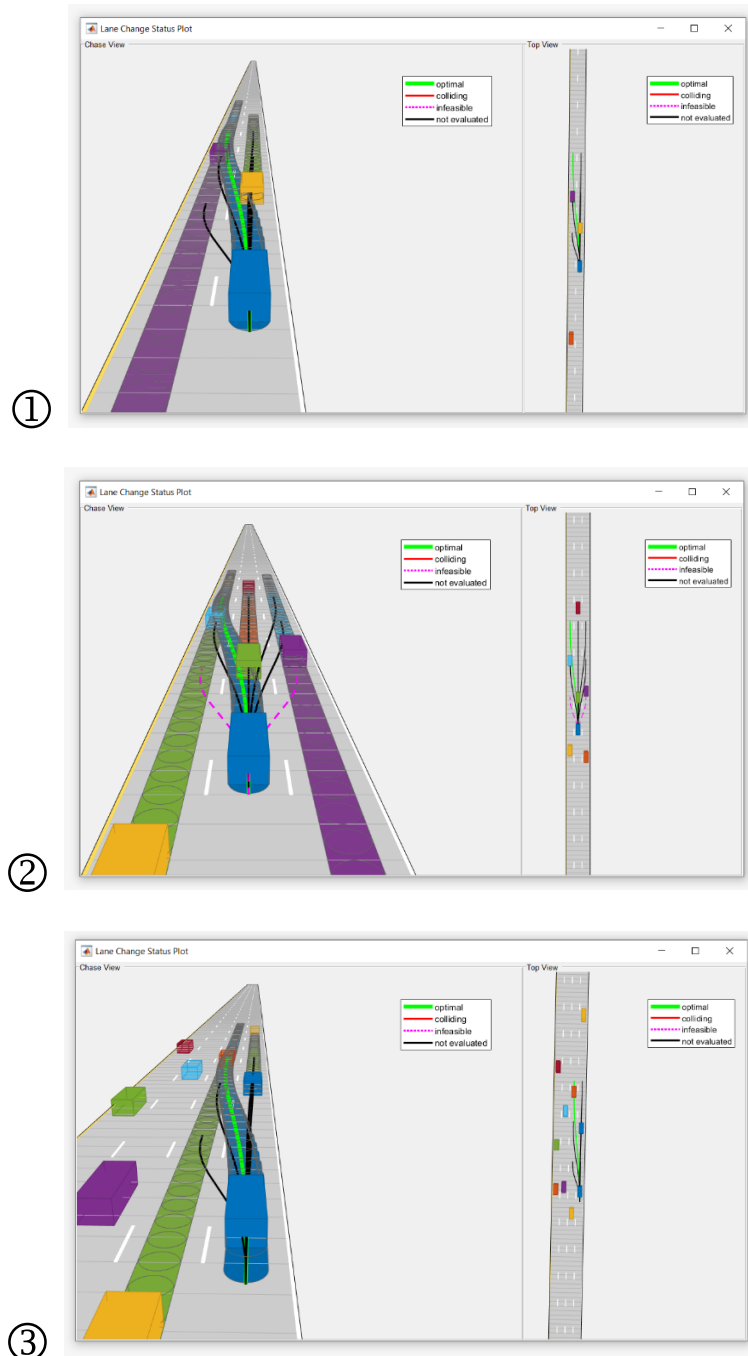


Figure 5.2 - The ego vehicle performing its first lane change in each scenario: (1) driving scenario 1; (2) driving scenario 2 and (3) driving scenario 3.

Each driving scenario was simulated for four different velocities of the ego vehicle, as shown in Figure 5.3.

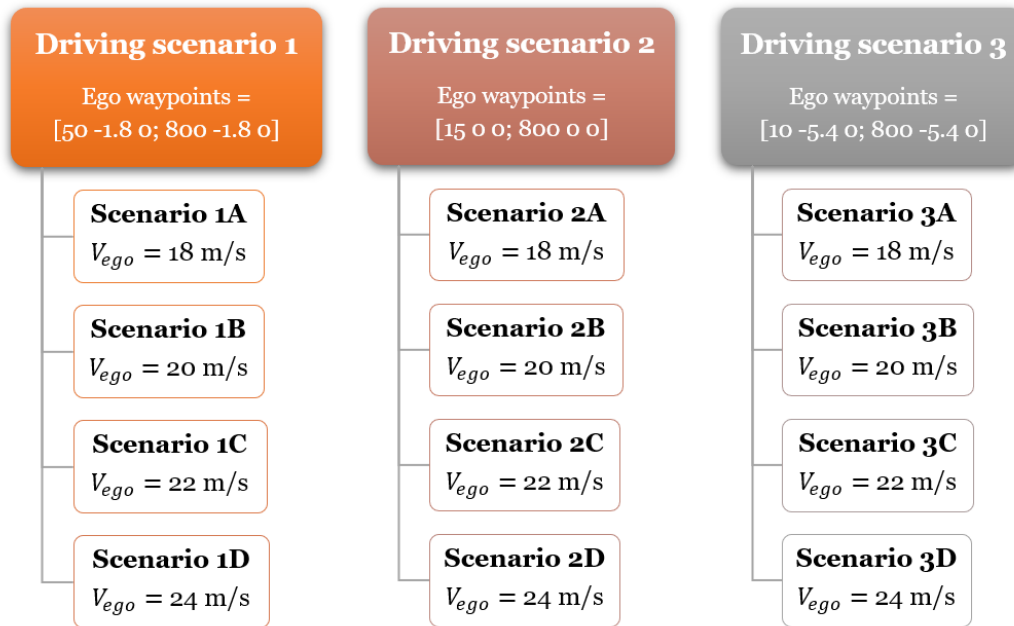


Figure 5.3 - Ego vehicle characteristics for the 12 scenarios considered for this study.

Collision occurrences happen in very extreme scenarios and when they happen the closed-loop simulation stops. Although the objective of this vehicle model is to test the performance of the ego vehicle for unpredictable situations that may arise (avoiding collisions), these extreme cases are also considered, so the performance of the system shows its maximum capabilities.

Criticality arises when the relative distance and relative velocity between the ego vehicle and the lead vehicle is reduced, generating different sets of results with regards to the scenario elements. Figures 5.4 to 5.27 show the performance of the ego vehicle in each simulation for further evaluation of the developed system.

The ego vehicle travels at a constant velocity, which means that its direction is also constant. This means that when the vehicle changes direction to perform a lane change maneuver, its speed decreases slightly. At that moment, the vehicle experiences acceleration to counteract the disturbance in velocity, where the control system activates the vehicle's accelerator to make the speed return to its defined value.

Steering angle and yaw angle determine whether the vehicle has turned into the left or right lane. When the steering wheel is rotated to the right, the steering angle is positive, when it is rotated to the left, the steering angle is negative [124]. Regarding the yaw angle, and in agreement with the Driving Scenario Designer app, when the vehicle turns to the left the angle is positive, when the vehicle turns to the right the angle is negative.

Scenario 1A: In driving scenario 1 for a vehicle speed of 18 m/s, at instant $t = 1.8$ sec the vehicle initiates a lane change maneuver as can be seen by the decrease in velocity (consequently increased acceleration) and the change in steering angle. From the yaw and steering angle graphs, it is possible to see that the vehicle performed a lane change to the left lane - the steering angle is negative and the yaw angle is positive.

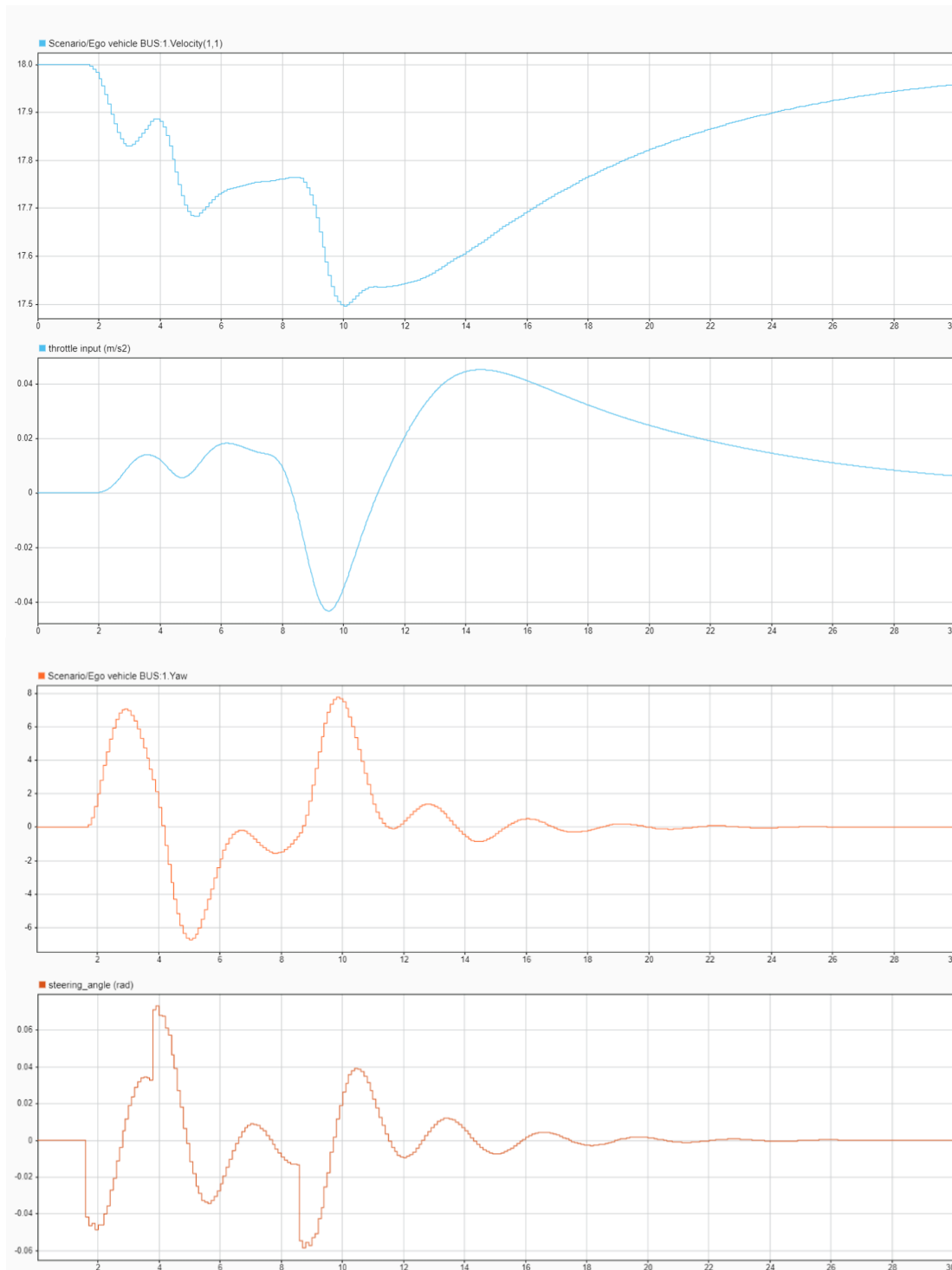


Figure 5.4 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1A.

At $t = 3.9$ sec, the vehicle performs a lane change to its right, returning to its initial lane (lane 1) as a faster target vehicle in lane 2 ($V_{target3} = 22$ m/s) approaches the main vehicle. As the return to lane 1 is not ideal due to the low speed vehicle ahead ($V_{target2} = 17$ m/s), at instant $t = 7.6$ sec the ego vehicle is forced to slow down to avoid a collision (Figure 5.5). The ego vehicle control system decelerates until the planning system provides a feasible trajectory for the vehicle to pursue, which happens at instant $t = 8.7$ sec where the vehicle makes its third lane change to its left. The ego vehicle remains in this lane until the end of the simulation. Due to the fact that the vehicle in front is faster, the ego vehicle does not need to change lanes, since a lane change affects the vehicle's performance. In this way, the main vehicle travels to the end of the simulation in follow the leading vehicle mode.

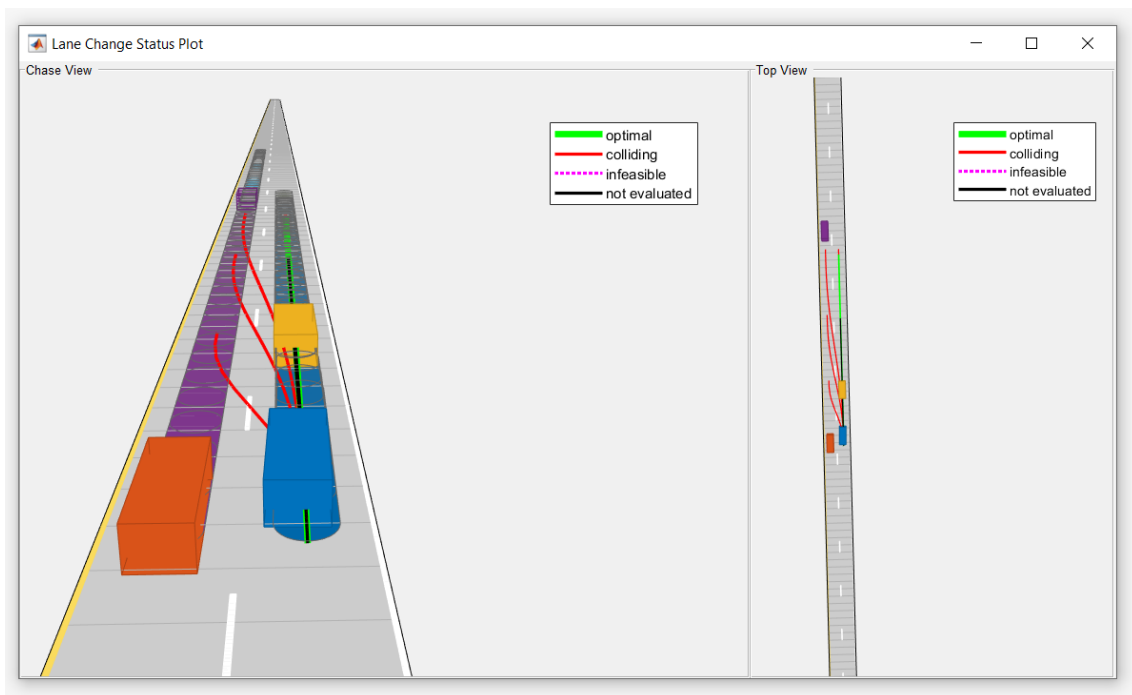


Figure 5.5 - Ego vehicle at instant $t = 7.6$ sec, when the vehicle starts to slow down to avoid a collision with the target vehicle ahead.

Scenario 1B: In driving scenario 1 for a vehicle speed of 20 m/s, and as in the previous scenario, exists a target vehicle ahead of the main vehicle that has a lower velocity ($V_{target2} = 17$ m/s). In this scenario, the ego vehicle lags behind the target vehicle 2 until a trajectory for a more optimal lane change emerges and, therefore, the ego vehicle control system decelerates slightly to avoid a collision. At instant $t = 2.9$ sec the vehicle performs a lane change maneuver to its left, and at this instant the acceleration increase to counteract the decrease in speed caused by the lane change manoeuvre. At instant $t = 21$ sec the vehicle returns to the right lane, remaining there until the end of the simulation. There are no target vehicles in the right lane ahead of the main vehicle, the only target vehicle in the right lane stayed behind the ego vehicle because it has a lower speed than the ego vehicle ($V_{target} = 17$ m/s), so the ego vehicle is in cruise control mode.

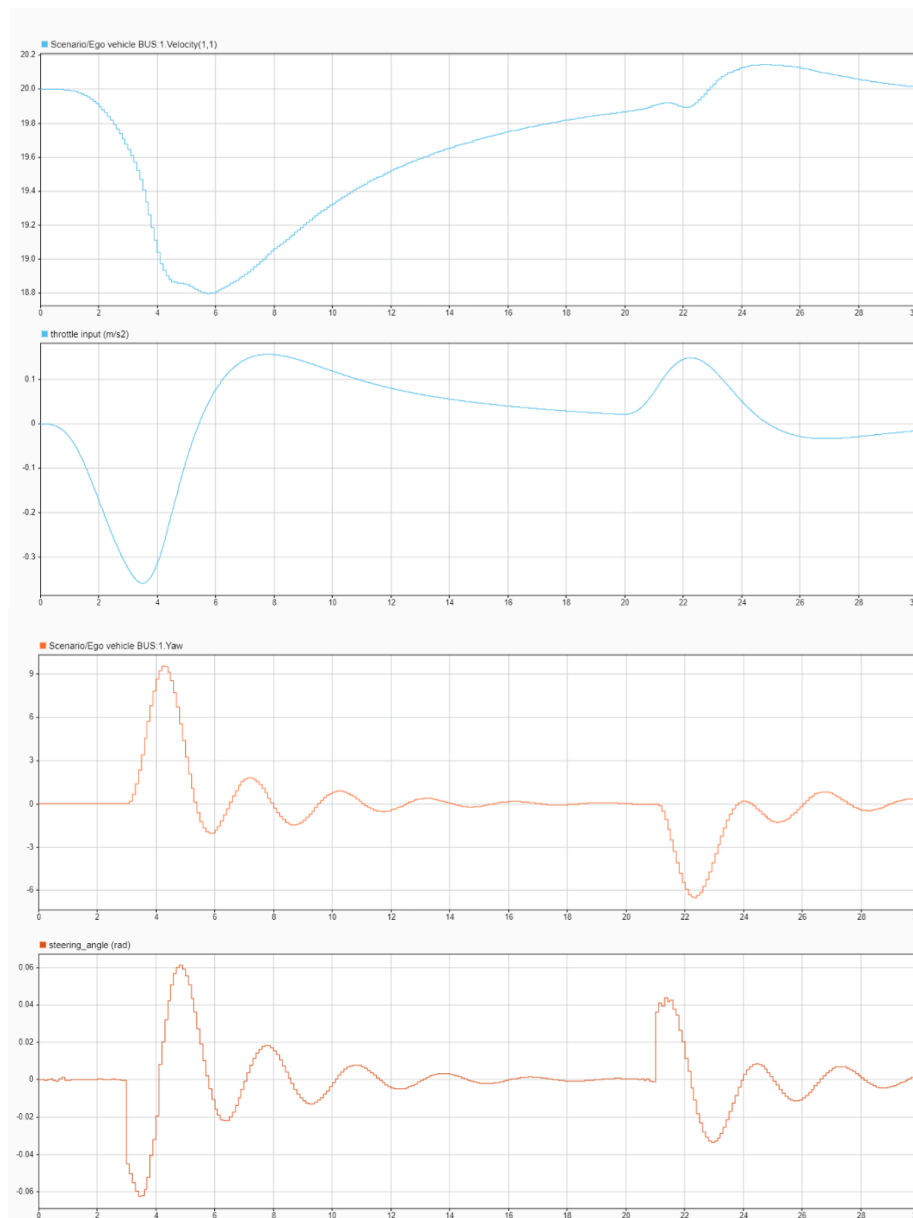


Figure 5.6 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1B.

Scenario 1C: In driving scenario 1 for a vehicle speed of 22 m/s, the ego vehicle behaves as in the previous scenario, where the only difference here is the times in which the two lane change maneuvers are initiated - in this scenario the ego vehicle starts both maneuvers earlier. The ego vehicle makes the first lane change to the left at instant $t = 0.1$ sec and returns to the right lane at instant $t = 4$ sec, where it remains until the end of the simulation in cruise control mode.

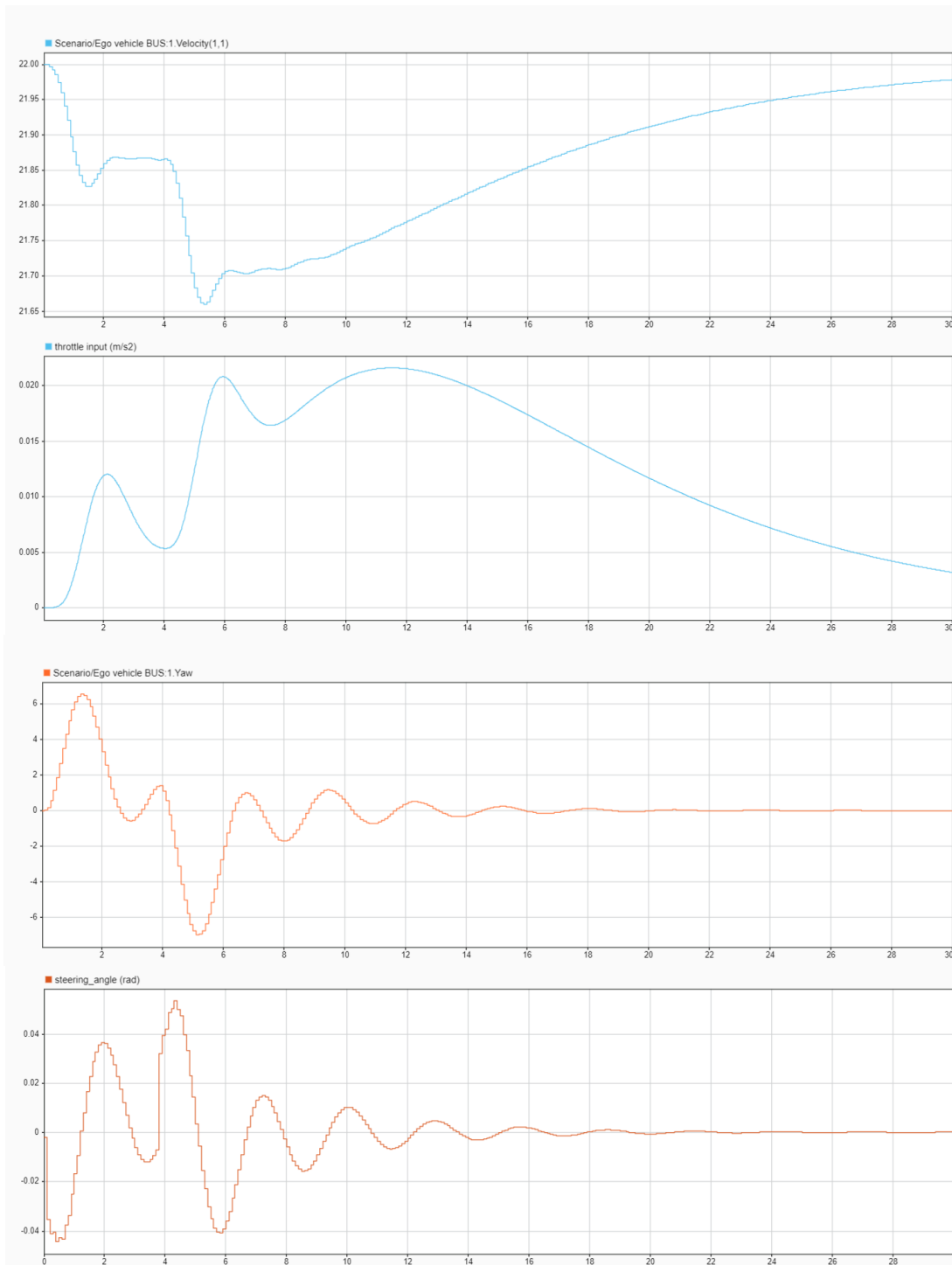


Figure 5.7 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1C.

Scenario 1D: In driving scenario 1 for a vehicle speed of 22 m/s, the ego vehicle behaves as in the last two scenarios. In this scenario, the first lane change to the left takes place at the same time as the previous scenario ($t = 0.1$ sec), while the second maneuver in which the vehicle returns to the right lane happens earlier, $t = 2.3$ sec. From that moment on, the vehicle remains in the right lane until the end of the simulation in cruise control mode.

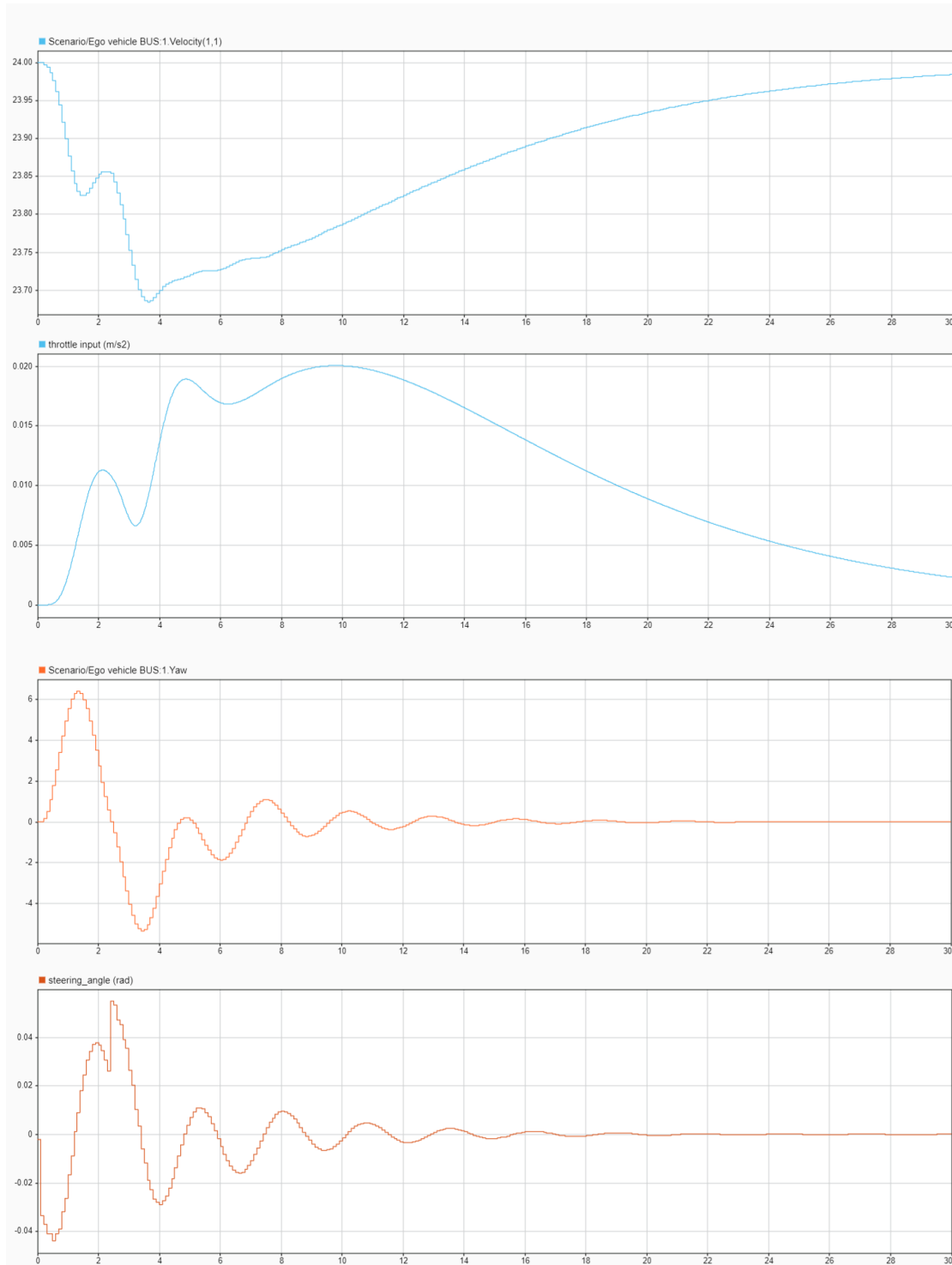


Figure 5.8 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 1D.

Scenario 2A: In driving scenario 2 for a vehicle speed of 18 m/s, at instant $t = 2$ sec the ego vehicle control system is forced to decelerate to avoid a collision with the slower vehicle ahead ($V_{target4} = 15$ m/s), until the planning system provides a better trajectory. At instant $t = 3.4$ sec the main vehicle gets a better trajectory in its right lane, for which a lane change maneuver is initiated. At this instant the velocity of the main vehicle decreases more sharply than before the lane change. After that, the vehicle performs two lane changes to its right: at instant $t = 12.4$ sec, returning to lane 2 (its initial lane), and at instant $t = 14$ sec, where it goes to the rightmost lane of the road, remaining there until the end of the simulation. In this lane there is a target vehicle that has a higher speed than the ego vehicle ($V_{target3} = 19$ m/s) and, therefore, this target vehicle is traveling in front of the ego vehicle during the entire simulation making the ego vehicle travelling in the follow the leading vehicle mode.

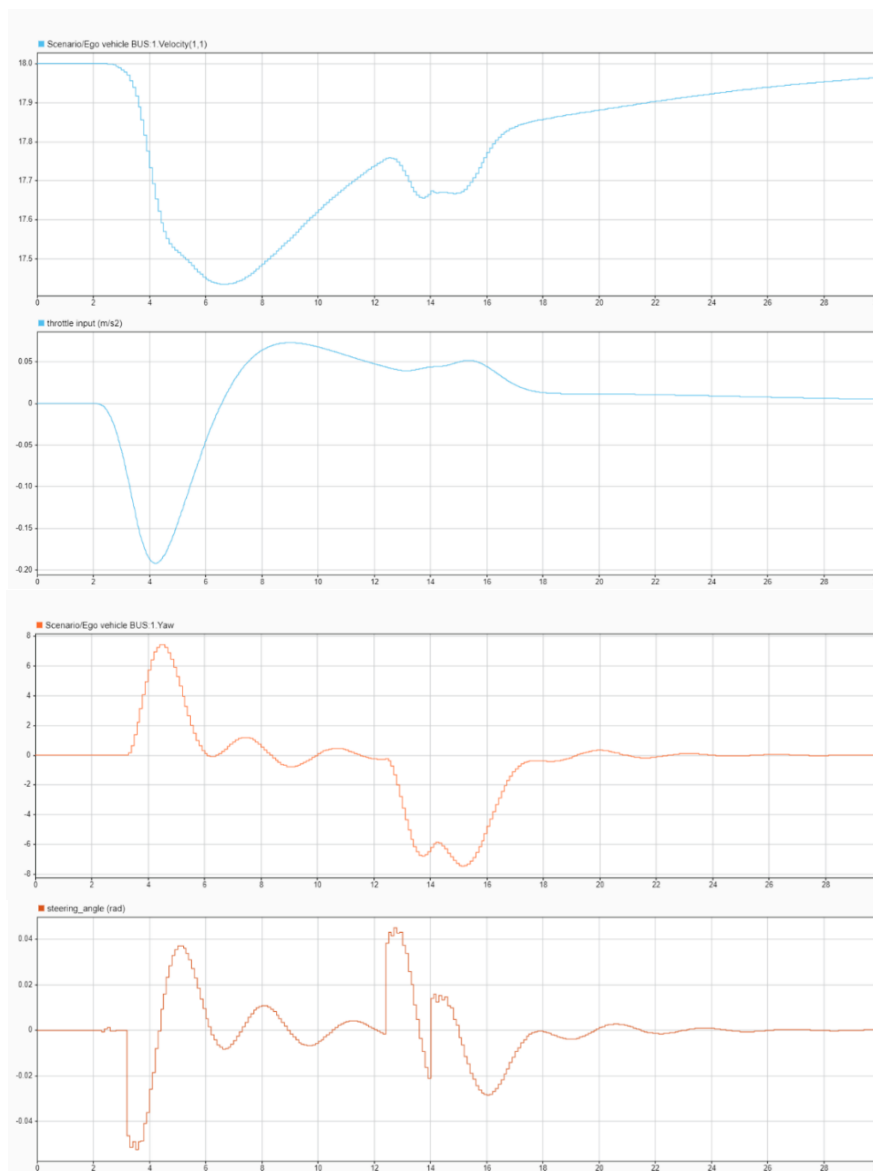


Figure 5.9 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2A.

Scenario 2B: In driving scenario 2 for a vehicle speed of 20 m/s, at instant $t = 0.1$ sec the ego vehicle performs a lane change to the left (lane 3). After that, the vehicle makes two lane changes to its right lane: at instant $t = 6.8$ sec the vehicle turns right, returning to its initial lane (lane 2), and then at instant $t = 11$ sec turns right again, going to the rightmost lane of the road (lane 1), as shown in Figure 5.11. The decrease in acceleration before each of these two previous lane changes is due to the control system needing to decelerate until the planning system obtains a better trajectory, since the vehicle in front is traveling at a slower velocity. Between $t = 11$ sec and $t = 14.7$ sec, the vehicle's steering angle signal suffers some disturbances, however, no lane change is made.

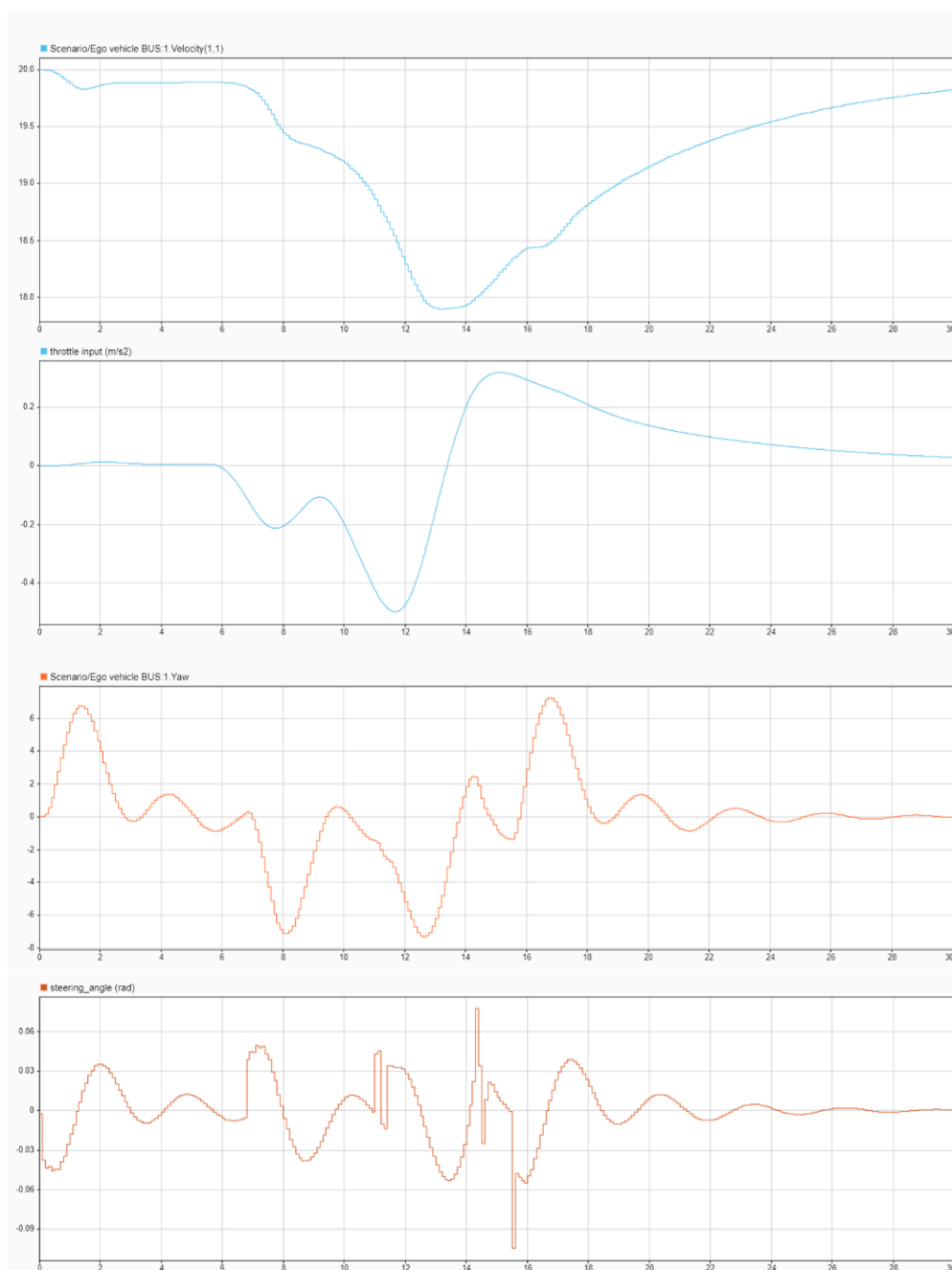


Figure 5.10 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2B.

These steering angle disturbances are due to the planning system being constantly updating the best trajectory for the ego vehicle over time, since the current one is not feasible considering the relative distance and relative velocity between the leading vehicle and the ego vehicle, as shown in Figure 5.12.

At instant $t = 15.5$ sec (Figure 5.13), the main vehicle is finally able to generate a feasible trajectory, where it makes a lane change to its left returning to the middle lane. The ego vehicle remains in this lane until the end of the simulation and as the only target vehicles traveling in the middle lane are target vehicles 4 and 6 which travel at a lower speed than the ego vehicle ($V_{\text{target4}} = V_{\text{target6}} = 15$ m/s), the ego vehicle travels in this lane in the cruise control mode.

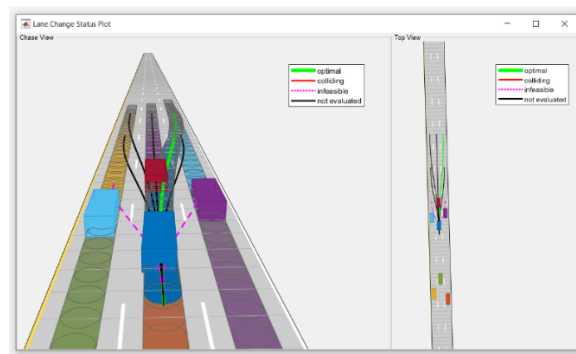


Figure 5.11 - Scenario 2B for instants $t = 11$ sec.

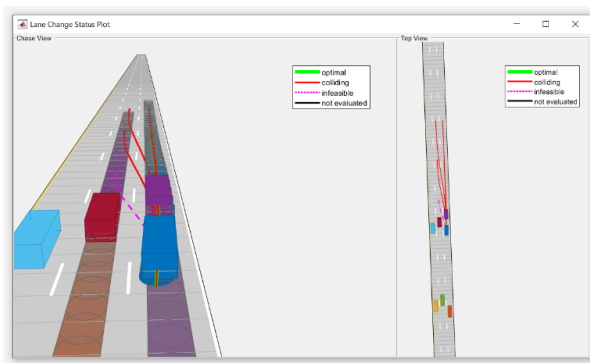


Figure 5.12 - Scenario 2B for instant $t = 13.5$ sec.

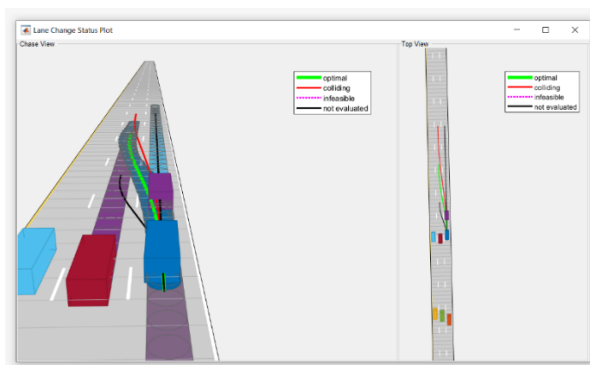


Figure 5.13 - Scenario 2B for instant $t = 15.5$ sec.

Scenario 2C: In driving scenario 2 for a vehicle speed of 22 m/s, the simulation results in an extreme case. At instant $t = 9.7$ sec the simulation stops because the main vehicle collides with the vehicle in front of it. Before the crash, the ego vehicle makes two lane changes: the first one at $t = 0.1$ sec where it turns into its left lane (lane 3), and the second one at $t = 3.4$ sec where performs a lane change to the right, returning to the middle lane. Although the vehicle slows down to avoid colliding with the target vehicle in front, the vehicle cannot find a collision-free trajectory and crashes, stopping the simulation.

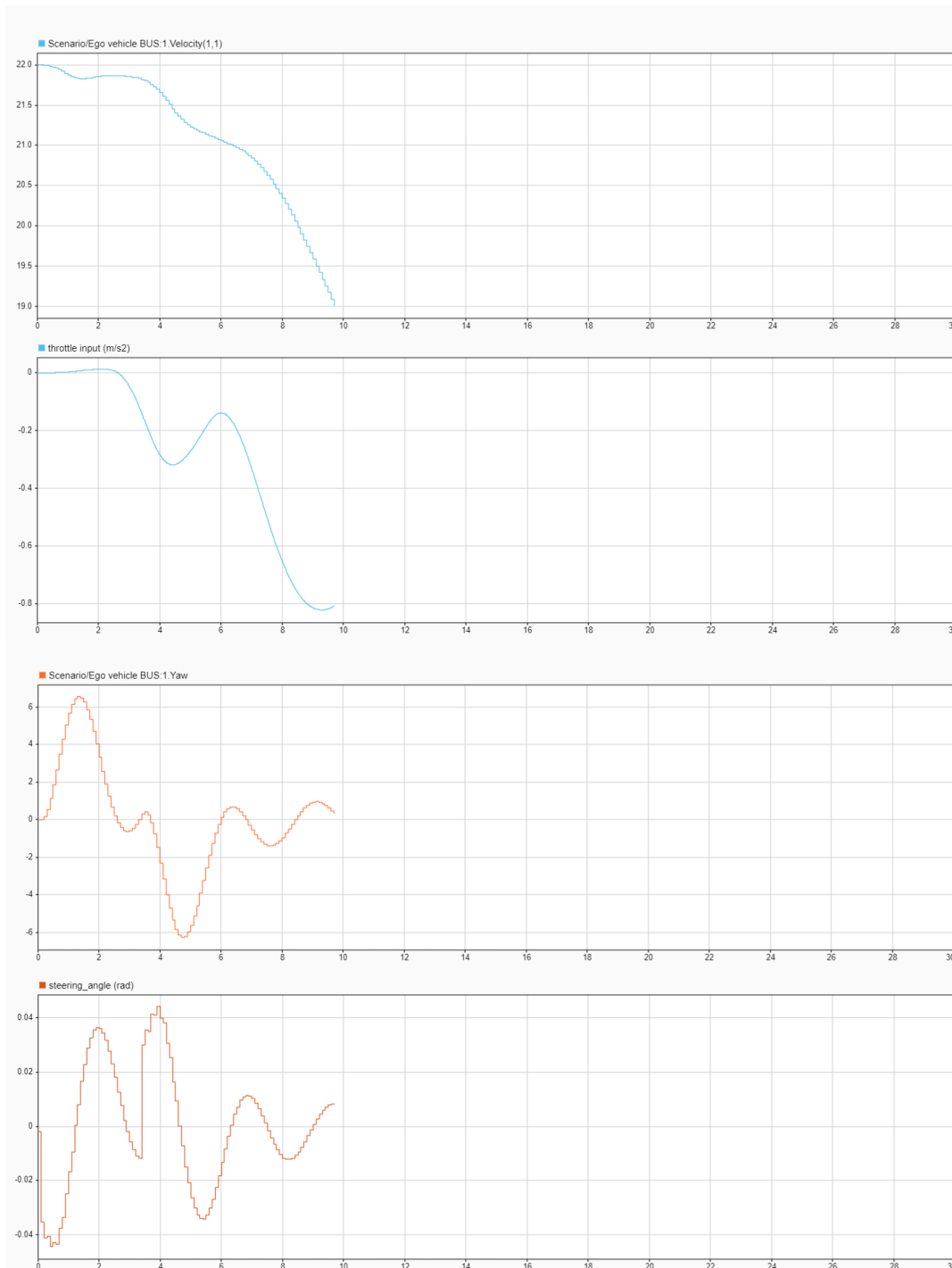


Figure 5.14 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2C.

Scenario 2D: In driving scenario 2 for a vehicle speed of 24 m/s, at instant $t = 0.1$ sec the ego vehicle makes a lane change for the left lane (lane 3) and until the planning system has a better trajectory, the control system decelerates to avoid a collision, since the target vehicle ahead has a lower velocity ($V_{target5} = 17$ m/s). At the instant $t = 2.2$ sec the ego vehicle changes lane to its right, returning to the middle lane. In the middle lane as in the previous two scenarios, the vehicle has difficulty finding feasible lanes to perform a lane change: in the instants $t = 3.6$ sec and $t = 4.4$ sec the ego vehicle starts to turn right and left, respectively, but it doesn't make a complete lane change – the vehicle only makes a complete lane change, to the right (lane 1), at instant $t = 5.2$ sec.

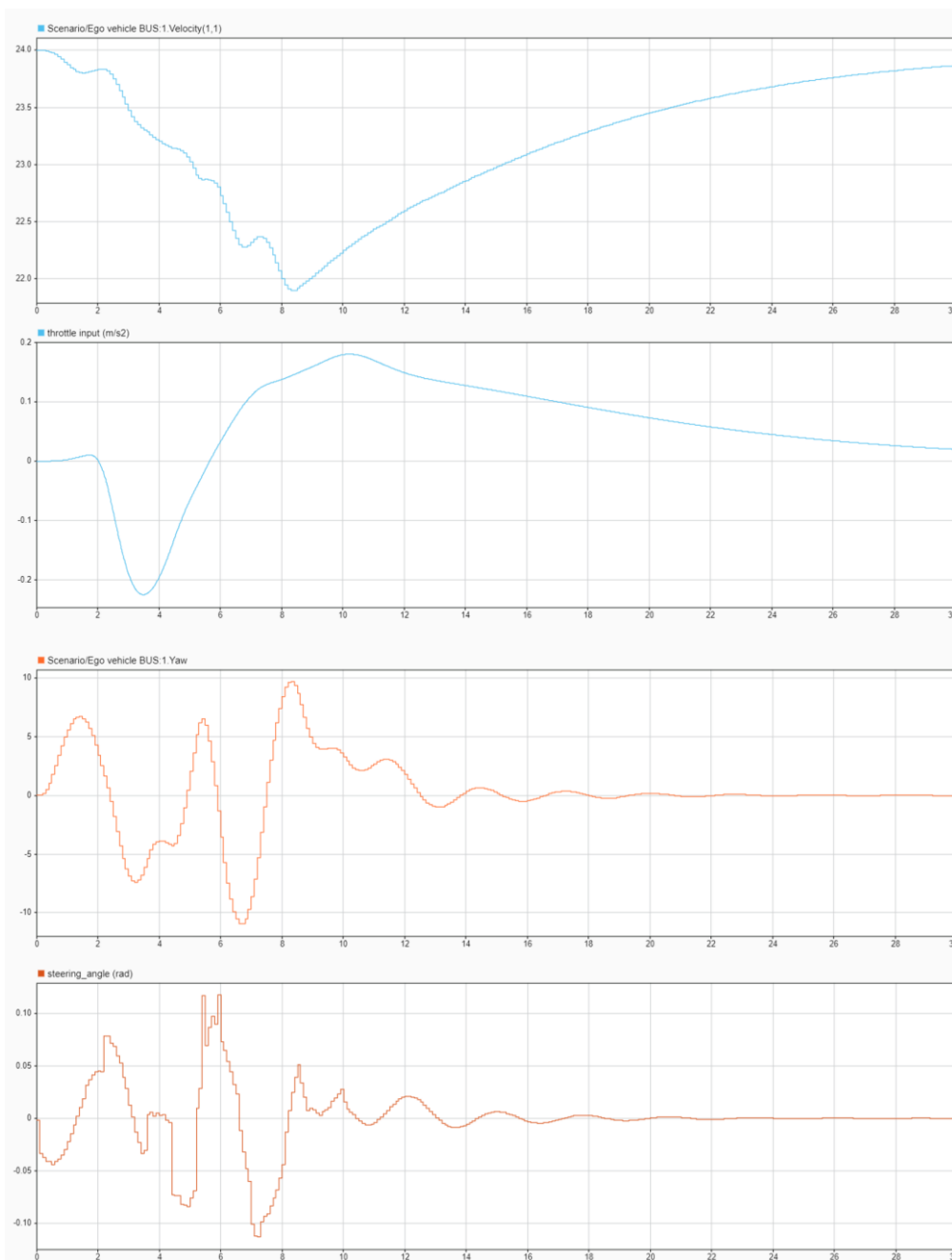


Figure 5.15 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 2D.

As illustrated in Figure 5.16, this difficulty is due to the fact that the planning system seeks the best trajectory based on the current position of the ego vehicle and the future positions of the target vehicles, estimating the relative distance and the relative velocity at each sample time to find the best feasible trajectory for the ego vehicle.

The ego vehicle makes two more lane changes, both to the left: at instant $t = 6.6$ sec the ego vehicle begins to perform a lane change to its left lane going to the middle lane, while at instant $t = 8.1$ sec the vehicle performs a new lane change to its left lane (lane 3). The ego vehicle remains in that lane until the end of the simulation in cruise control mode. The ego vehicle planning system decides that the leftmost lane is most suitable for it based on the relative distance and relative velocity between the main vehicle and the target vehicles in each lane, as shown in Figure 5.17.

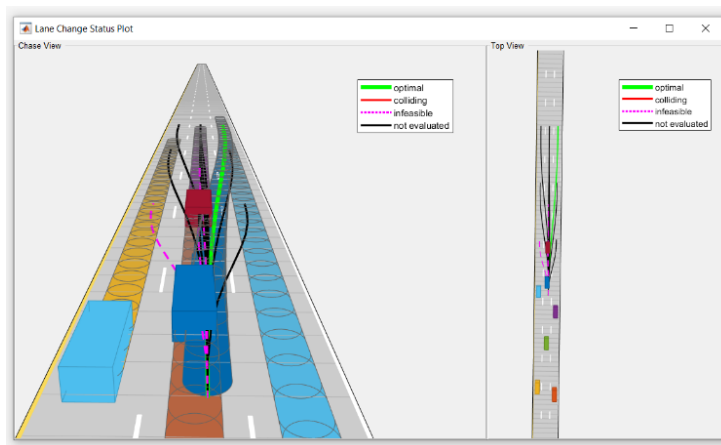


Figure 5.16 - Lane change to the right lane at instant $t = 5.2$ sec.

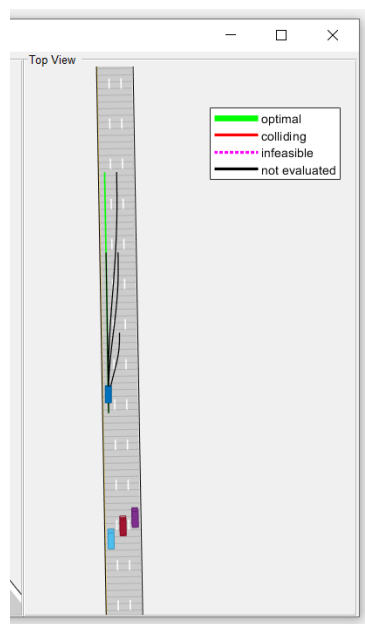


Figure 5.17 - Ego vehicle in cruise control mode at instant $t = 13$ sec.

Scenario 3A: In driving scenario 3 for a vehicle speed of 18 m/s, the ego vehicle only makes a lane change to its left lane (lane 2) at instant $t = 5.2$ sec. The vehicle remains in lane 2 in the mode of following the lead vehicle, since the velocity of the vehicle ahead has the same velocity as the ego vehicle and their relative distance is acceptable. Besides that, the main vehicle's right and left lanes do not present best trajectories for its current and future positions, as shown in the Figure 5.19 and Figure 5.20.

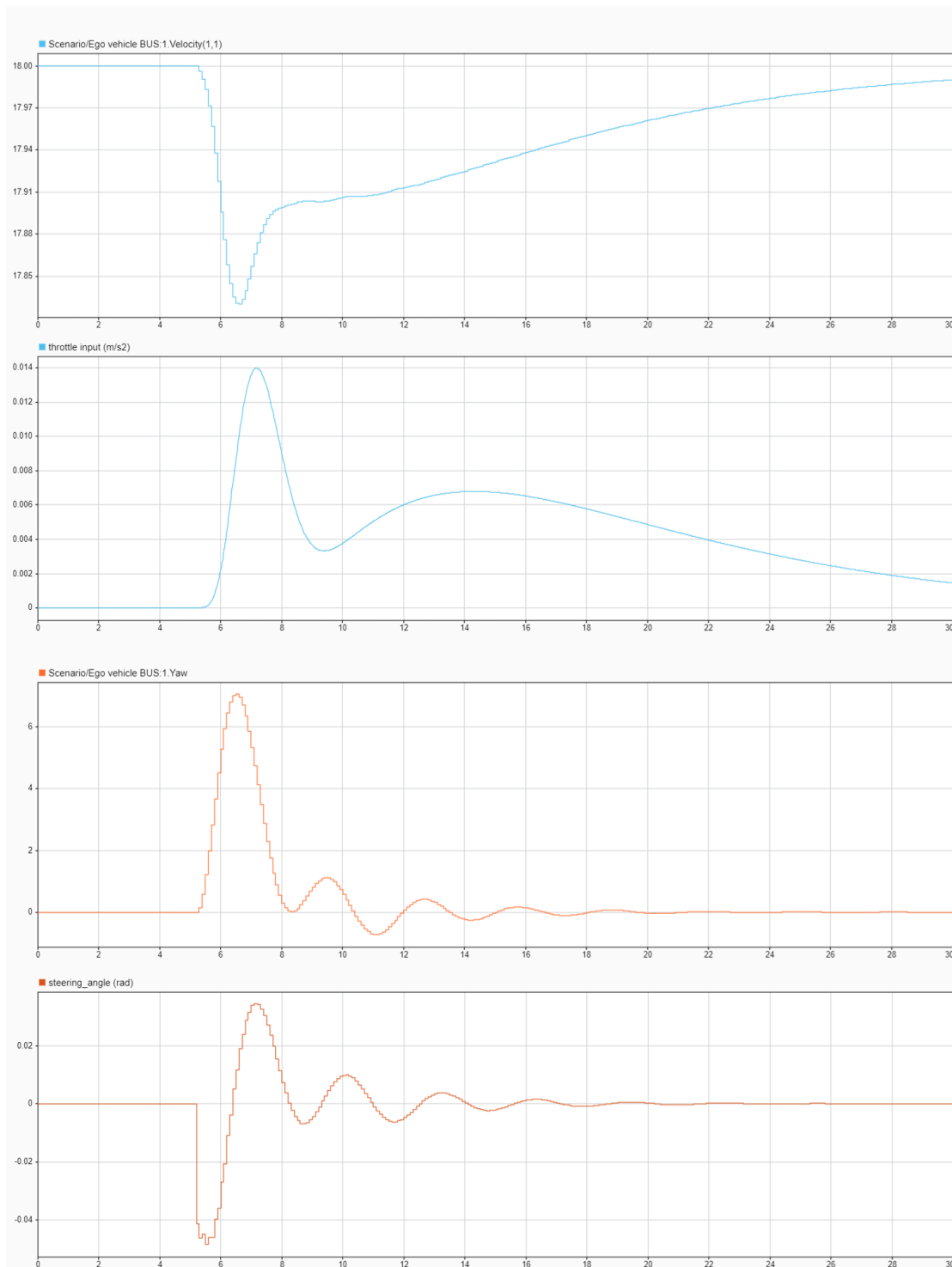


Figure 5.18 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3A.

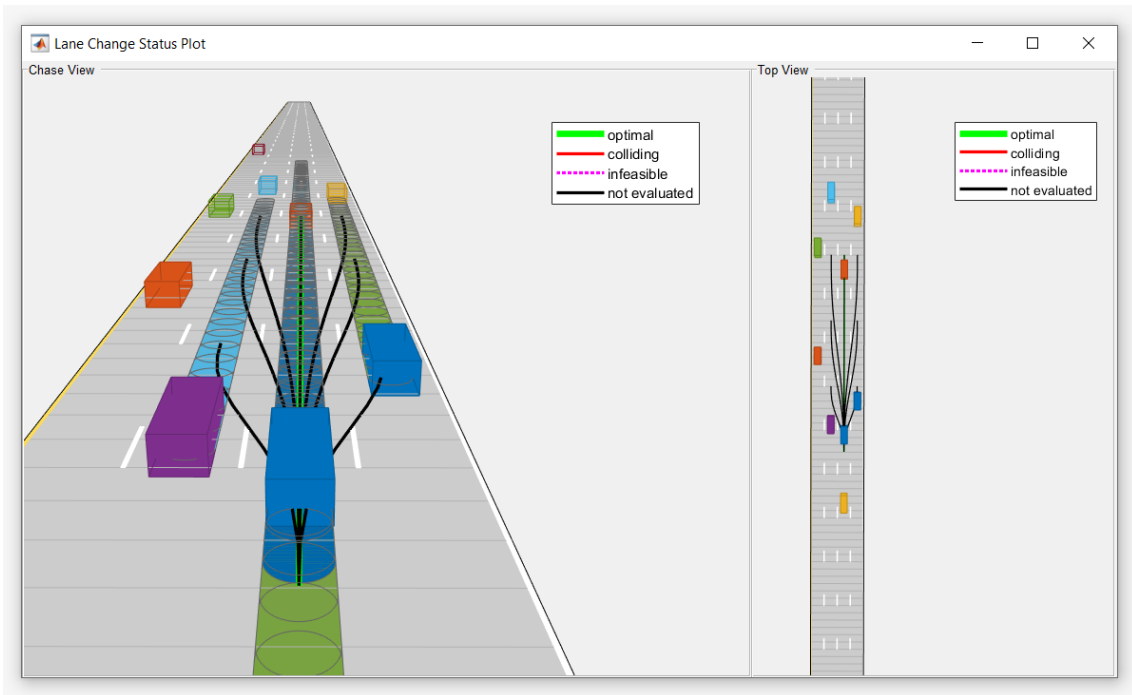


Figure 5.19 - Ego vehicle in following the lead vehicle mode at instant $t = 15.3$ sec.

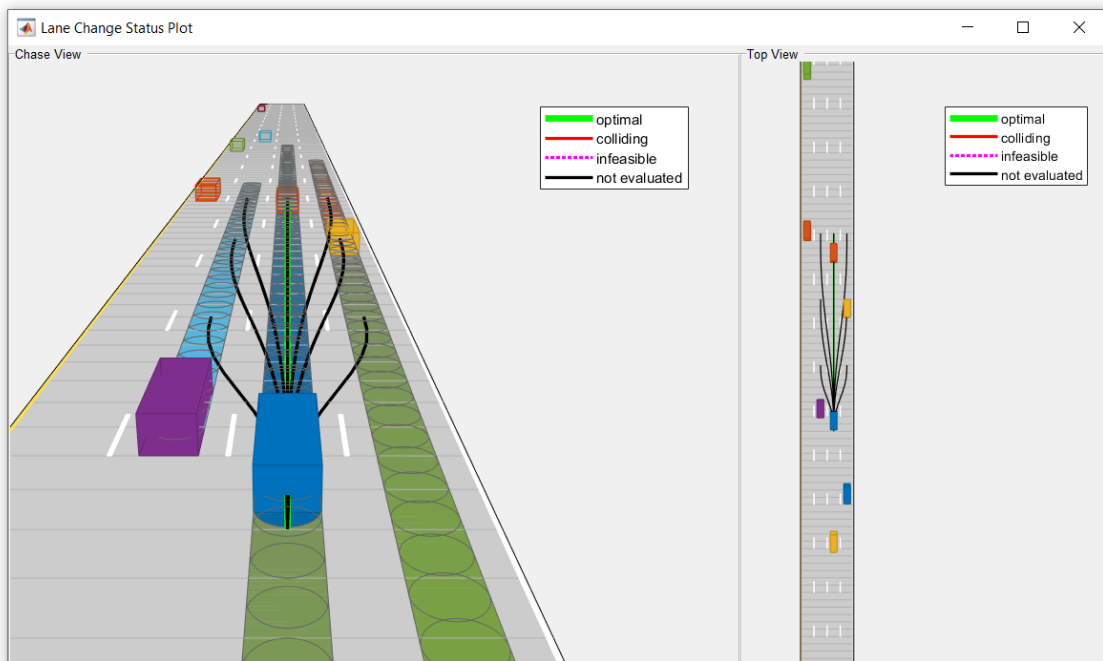


Figure 5.20 - Ego vehicle in following the lead vehicle mode at instant $t = 30$ sec.

Scenario 3B: In driving scenario 3 for a vehicle speed of 20 m/s, at instant $t = 2.4$ sec the ego vehicle makes a lane change for the left lane (lane 2) since the vehicle in front has a lower speed. At instant $t = 10.2$ sec the ego vehicle makes another lane change to its left, going into lane 3 of the road, where it remains until the end of the simulation in the mode of following the lead vehicle.

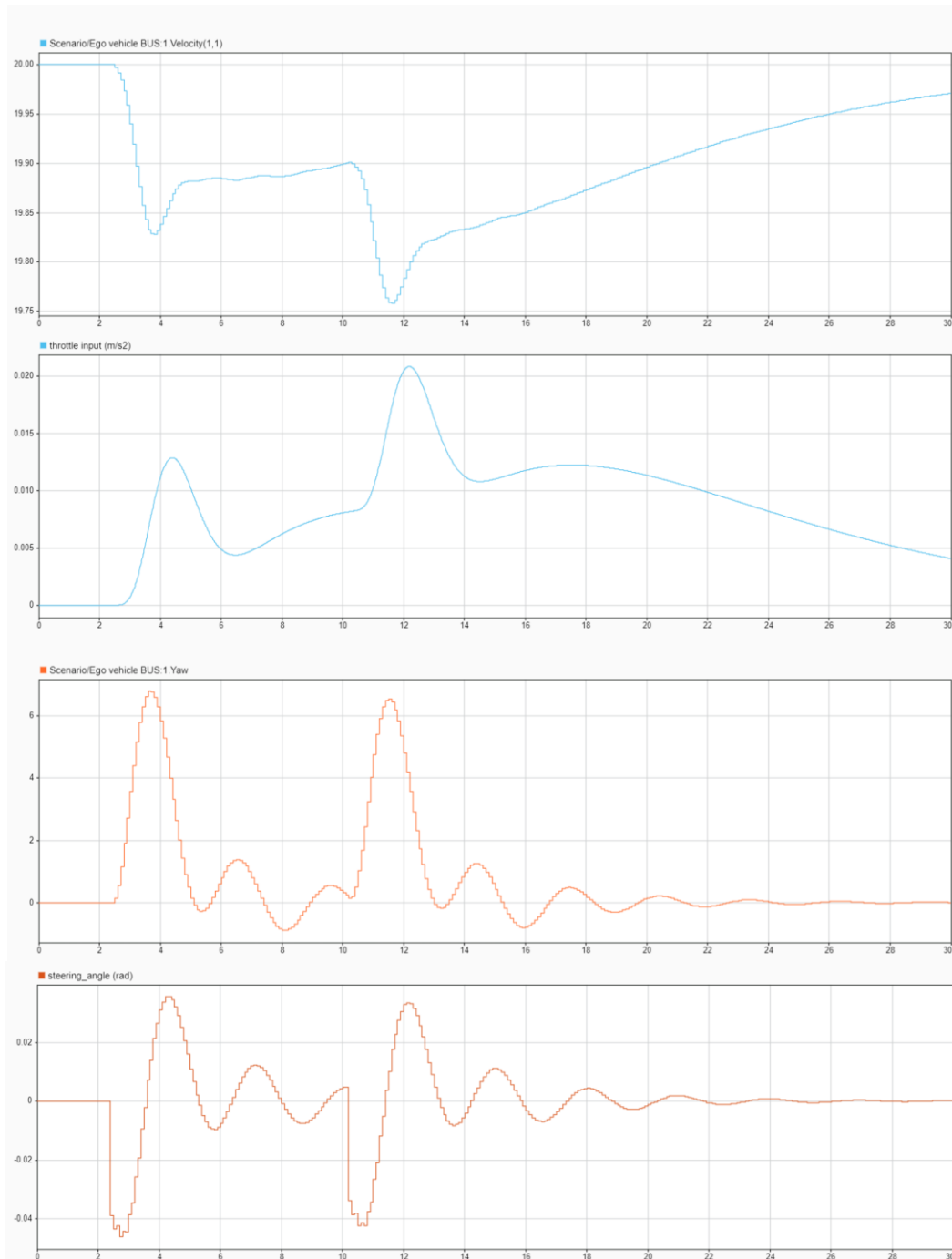


Figure 5.21 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3B.

Scenario 3C: In driving scenario 3 for a vehicle speed of 22 m/s, the steering angle and the yaw angle suffered some disturbances during the first 15 seconds of simulation. At instant $t = 1.7$ sec the ego vehicle makes a lane change from lane 1 to lane 2 and at instant $t = 6$ sec from lane 2 to lane 3.

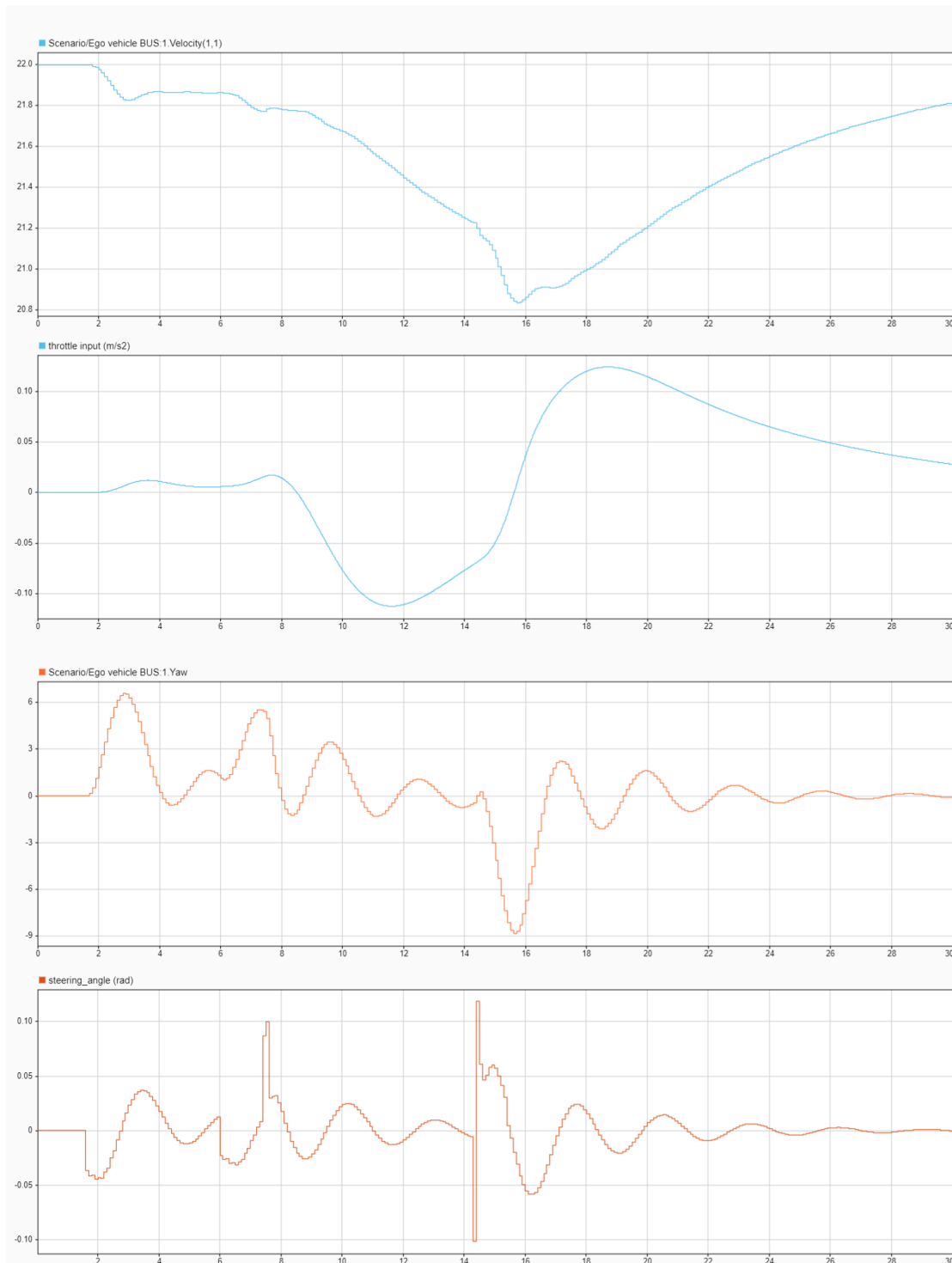


Figure 5.22 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3C.

Between instant $t = 6$ sec and $t = 14.5$ sec, the ego vehicle can't make a lane change based on the relative distance and relative velocity between the main vehicle and the target vehicles in the adjacent ego lanes according to their future positions, as shown in Figure 5.23. In this way, at $t = 8$ sec, the ego control system slows down so the vehicle stays behind the vehicle in front ($V_{target5} = 21$ m/s), until there is a feasible trajectory for the ego vehicle to follow.

At instant $t = 14.5$ sec, the main vehicle finally achieves a feasible trajectory to its right (Figure 5.24), returning to lane 2 where it remains until the end of the simulation in cruise control mode. At this instant, the acceleration of the ego vehicle begins to increase to counteract the decrease in vehicle speed due to the lane change maneuver.

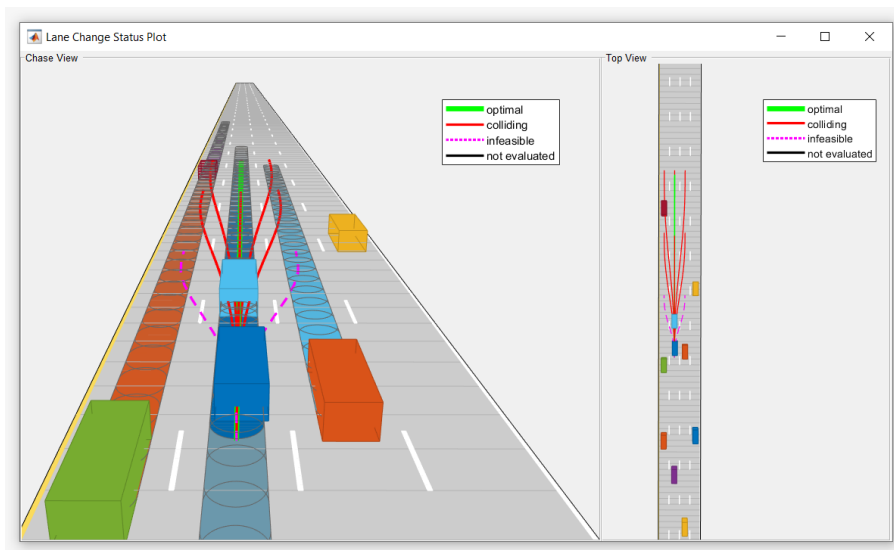


Figure 5.23 – The ego vehicle at instant $t = 11.8$ sec when it is unable to track a feasible trajectory.

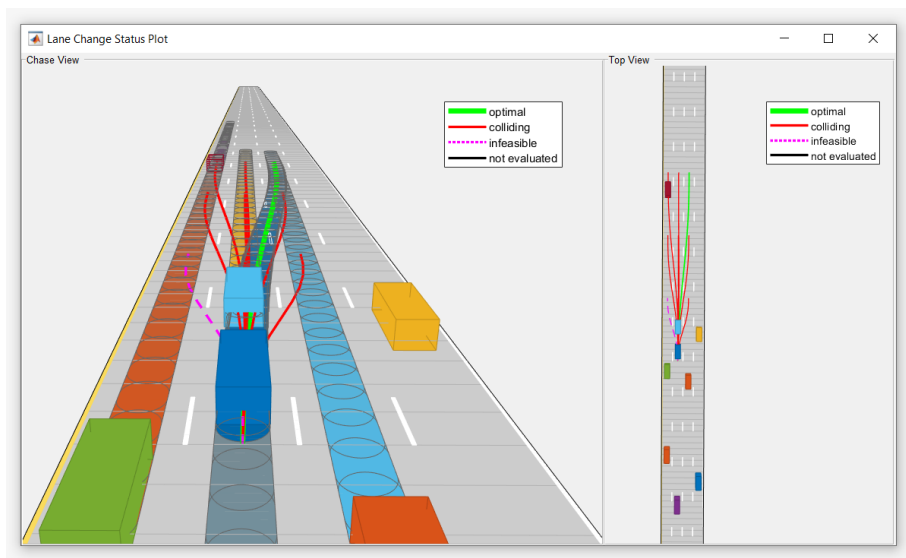


Figure 5.24 – The ego vehicle at instant $t = 14.5$ sec when it is finally able to track a feasible trajectory, making a lane change maneuver.

Scenario 3D: In driving scenario 3 for a vehicle speed of 24 m/s, at instant $t = 1.2$ sec the ego vehicle makes the only possible maneuver: a lane change to the left, due to the lower speed of the target vehicle ahead. However, this isn't the best trajectory for the future positions of the main vehicle, since in lane 2 the vehicle ahead moves at a lower speed than the ego.

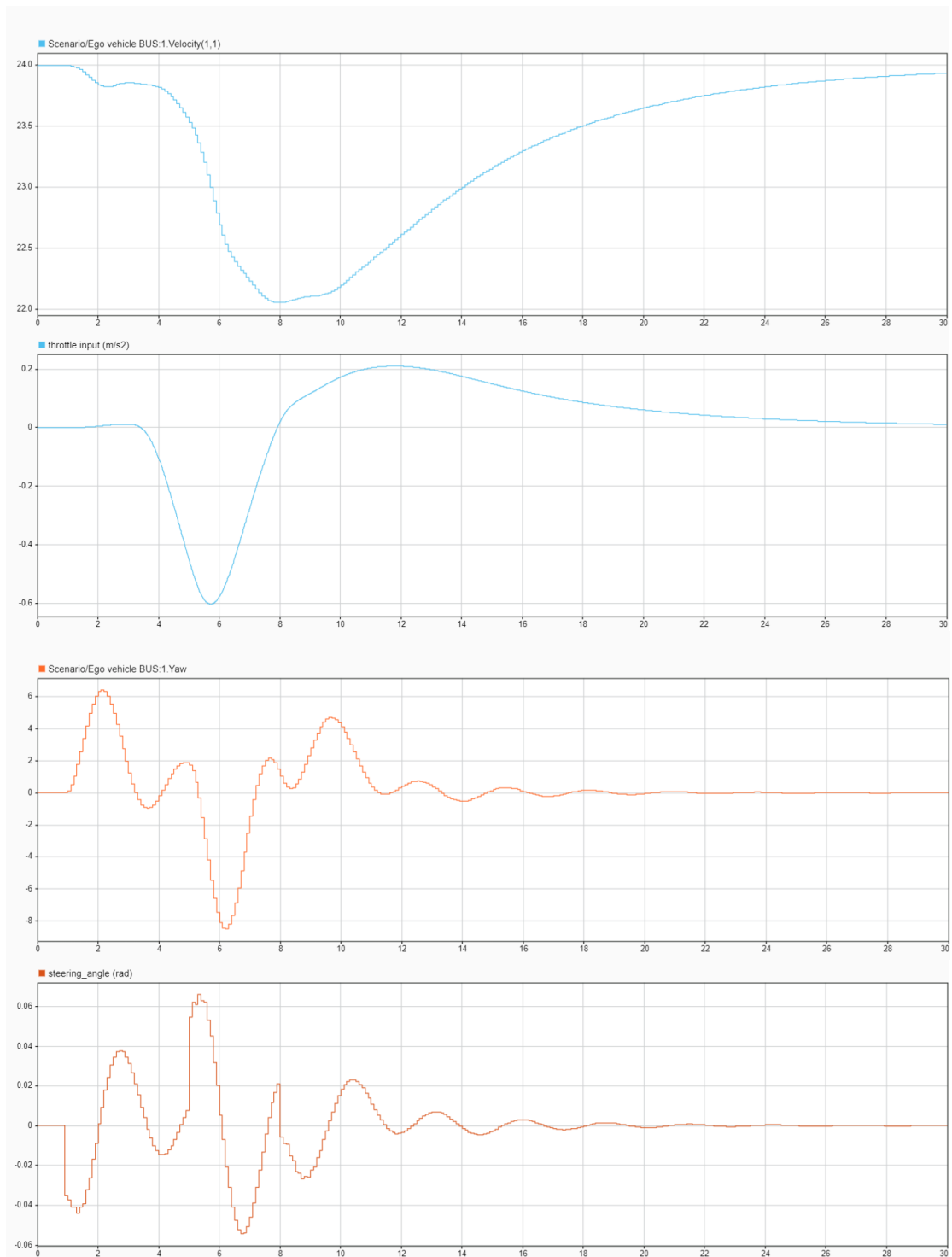


Figure 5.25 - Graph of velocity, acceleration, yaw angle and steering angle of the scenario 3D.

In this way, and as illustrated in Figure 5.26, the ego vehicle control system decelerates to avoid a collision until a better trajectory appears in the adjacent lanes, taking into account the relative distance and velocity. At instant $t = 5.2$ sec, the main vehicle gets a feasible trajectory on the left lane, so it performs a lane change returning to lane 1 (Figure 5.27). At this moment, the acceleration of the ego vehicle starts to increase to counteract the decrease in the vehicle velocity due to the lane change maneuver. At instant $t = 8.2$ sec the main vehicle makes a last lane change to its left, where remains until the end of the simulation in cruise control mode.

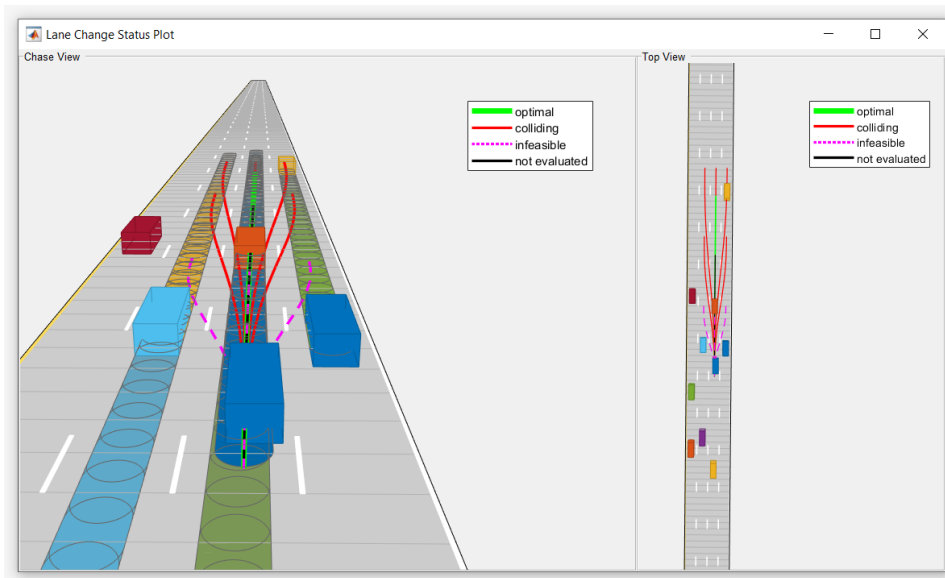


Figure 5.26 – The ego vehicle searching for a better trajectory at instant $t = 4.2$ sec.

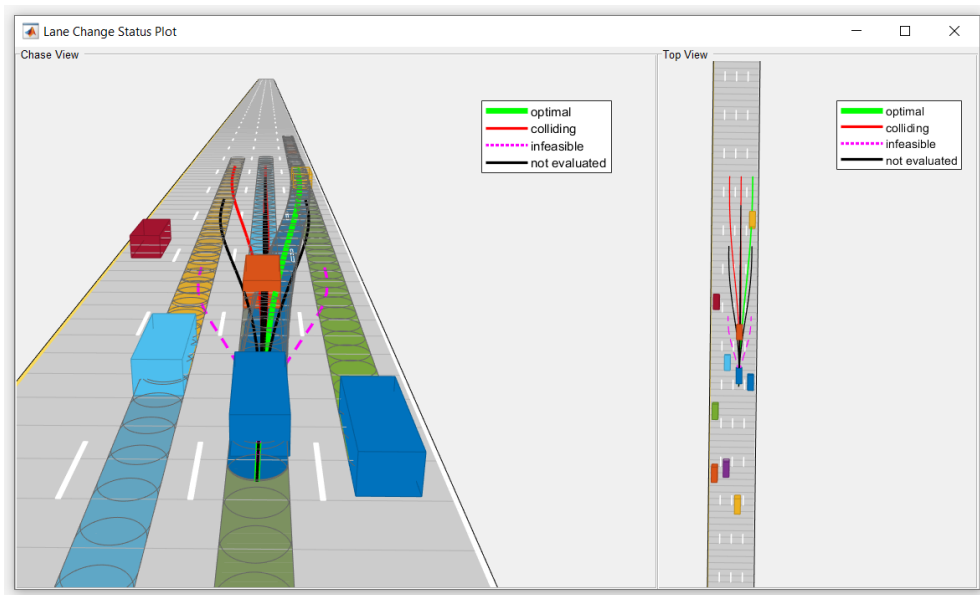


Figure 5.27 - Lane change in the instant $t = 5.2$ sec.

Table 5.4 summarizes information from the simulations, showing how many lane changes are performed by the ego vehicle in each scenario and whether the vehicle performs the other supported driving modes: cruise control and follow the leading vehicle.

Table 5.4 - Simulation results.

Scenario ID		Ego velocity	CC	FLV	LC	Collision
Driving scenario 1	Scenario 1A	18	✘	✓	3	No
	Scenario 1B	20	✓	✓	2	No
	Scenario 1C	22	✓	✓	2	No
	Scenario 1D	24	✓	✓	2	No
Driving scenario 2	Scenario 2A	18	✘	✓	3	No
	Scenario 2B	20	✓	✓	4	No
	Scenario 2C	22	✘	✓	2	Yes
	Scenario 2D	24	✓	✓	5	No
Driving scenario 3	Scenario 3A	18	✘	✓	1	No
	Scenario 3B	20	✘	✓	2	No
	Scenario 3C	22	✓	✓	3	No
	Scenario 3D	24	✓	✓	3	No

(✓) Used, (✘) Not used

Taking into account the results obtained, it was found that for the low-traffic scenario, with increasing speed, the main vehicle passes the slow vehicle in front earlier and more smoothly - the lane change maneuvers present both steering and yaw angles within a smaller range of values as the speed increases. This is due to the fact that in the scenario with low traffic density, the vehicle has enough space to maneuver with greater freedom, resulting in a smoother driving experience.

On the other hand, scenarios with high traffic density present different performances according to the number of lanes. In the dense three-lane traffic scenario, as the main vehicle velocity increases, the opposite of the driving scenario 1 happens. With increasing speed, the lane change maneuvers performed by the ego vehicle become less smooth, increasing the range of values of steering and yaw angles. With dense traffic, it becomes difficult for the vehicle to make lane change maneuvers, maintaining a constant speed, and an extreme case happens. As the number of lanes increases, no extreme cases unfold. In the four-lane traffic scenario, as the main vehicle velocity increases, the vehicle has to make more lane changes to avoid colliding with target vehicles in front, as shown in Table 5.4. Regarding steering and yaw angles, there is no gradual increase or decrease in their

range of values, as the main vehicle speed increases. For speeds of 18 and 20 m/s, both angles are included in smaller value intervals, while for speeds of 22 and 24 m/s the steering angle is included in slightly larger value intervals. In general, the trajectories for this four-lane scenario with dense traffic are smooth and without major disturbances in the performance of the ego vehicle (reduced driving aggressiveness), due to the fact that there are more lanes through which the vehicle can perform maneuvers to avoid collisions, maintaining a constant velocity.

The comparison of these 12 scenarios reveals that unsuccessful cases are less than successful cases, where only in one scenario a collision occur – scenario 2C. This means that based on the simulations performed, the developed model has a good performance in decision making. However, as these results are for very specific cases, further simulations with different criteria for scenario complexity would be needed.

The change in the value of the prediction horizon in the adaptive MPC was also tested as a possible variation in the study, since from the prediction horizon, the ego vehicle predicts the position of the target vehicles in its lane and in adjacent lanes. Increasing the prediction horizon increases the view of the ego vehicle in its front, detecting the lead vehicle earlier, however, changing the value of the prediction horizon did not prevent the collision in scenario 2C and did not alter any lane change maneuvers, nor the instants in which they took place.

Chapter 6

Conclusions and future works

Autonomous driving is one of the latest technological upgrades in the new world that can improve driver safety and reduce traffic congestion. However, developing this new platform is rife with challenges. Simulations cut the development time and cost since autonomous vehicles must be trained and tested to respond to all driving situations.

This dissertation provides an overview of autonomous and connected vehicle technology; verification, validation and testing of autonomous prototypes in which a survey of the techniques and tools used is carried out; and also, the implementation of a model-based framework to evaluate the performance of a vehicle model in some critical driving scenarios, through a software-in-the-loop simulation. The main objective of this model is to avoid any collision in order to ensure the vehicle's safety, while driving autonomously.

In our experiments we merged three main ADAS functions (CC, FLV and LC) into a Matlab & Simulink test bench, enabling the system to switch between them. The decision making in choosing the best driving mode for the ego vehicle over time is based on the relative distance and the relative velocity between the main vehicle and the target vehicles in the ego lane and in the adjacent lanes.

Predictions are vitally important for these type of systems that change dynamically over time. In this way, the core of this model is an adaptive MPC controller which controls the acceleration and steering of the vehicle to obtain the best possible performance.

Due to the countless scenarios that a road vehicle can encounter, in the design phase of this system it was necessary to choose only a few critical highway scenarios to test the performance of the model, which means that more simulations with different considerations are needed to validate the system as a safe system. Each system in the model – the planning system and the control system - was developed and tested independently (UT) and then integrated with each other to function as a group (IT), verifying if the integrated components meet the specifications.

Considering the limited time and simulation tools in this research, the obtained results are satisfying the expectations of the dissertation topic. However, as this field of study is really vast, there would be further research in this area that would complete this system to work in more complex scenarios.

For future work, it is suggested the optimization of some functions and the incorporation of others. To make the vehicle model more autonomous it is necessary to

include a sensing system (sensor fusion) so the vehicle is able to sense the environment around it. To test other types of roads, test changes of direction (e.g., scenarios with intersections), and test scenarios with traffic lights and crossings, it is necessary to integrate an autonomous braking system (AEB). Another possibility is the aggregation of a V2X communication system, so the vehicle becomes more independent according to the information it receives from the surrounding environment. Finally, move on to the next testing phase: the hardware-in-the-loop simulation.

References

- [1] M. Möstl, M. Nolte, J. Schlatow, and R. Ernst, “Controlling concurrent change – A multiview approach toward updatable vehicle automation systems,” *OpenAccess Ser. Informatics*, vol. 68, no. 4, pp. 1–4, 2019.
- [2] S. Grigorescu, T. Cocias, B. Trasnea, A. Margheri, F. Lombardi, and L. Aniello, “Cloud2edge elastic ai framework for prototyping and deployment of Ai inference engines in autonomous vehicles,” *Sensors (Switzerland)*, vol. 20, no. 19, pp. 1–21, 2020.
- [3] J. de Hoog, A. Janssens, S. Mercelis, and P. Hellinckx, “Towards a distributed real-time hybrid simulator for autonomous vehicles,” *Computing*, vol. 101, no. 7, pp. 873–891, 2019.
- [4] I. Kyriakopoulos, P. Jaworski, and S. Kanarachos, “DigiCAV project: Exploring a test-driven approach in the development of connected and autonomous vehicles,” in *2019 8th IEEE International Conference on Connected Vehicles and Expo, ICCVE 2019 - Proceedings*, 2019, pp. 1–6.
- [5] A. Zlocki, F. Fahrenkrog, M. Benmimoun, J. Josten, and L. Eckstein, “Evaluation of Automated Road Vehicles,” in *Road Vehicle Automation*, 2014, pp. 197–208.
- [6] M. Hamad, M. Tsantekidis, and V. Prevelakis, “Red-zone: Towards an intrusion response framework for intra-vehicle system,” in *VEHITS*, 2019, pp. 148–158.
- [7] A. Chattopadhyay and K. Y. Lam, “Security of autonomous vehicle as a cyber-physical system,” *2017 7th Int. Symp. Embed. Comput. Syst. Des. ISED 2017*, vol. 2018-Janua, pp. 1–6, 2018.
- [8] A. Joshi, “Real-Time Implementation and Validation for Automated Path Following Lateral Control Using Hardware-in-the-Loop (HIL) Simulation,” *SAE Tech. Pap.*, vol. 2017-March, no. March, 2017.
- [9] SAE international, “Automated Driving,” *SAE Int.*, pp. 1–2, 2016.
- [10] M. A. Meyer, C. Granrath, G. Feyerl, J. Richenhagen, J. Kathes, and J. Andert, “Closed-loop platoon simulation with cooperative intelligent transportation systems based on vehicle-to-X communication,” *Simul. Model. Pract. Theory*, vol. 106, no. April 2020, p. 102173, 2021.
- [11] M. A. Lundteigen, M. Rausand, and I. B. Utne, “Integrating RAMS engineering and management with the safety life cycle of IEC 61508,” *Reliab. Eng. Syst. Saf.*, vol. 94, no. 12, pp. 1894–1903, 2009.
- [12] R. Bell, “Introduction to IEC 61508,” in *Conferences in Research and Practice in Information Technology Series*, 2005, vol. 55, pp. 3–12.
- [13] R. Debouk, “Overview of the 2nd Edition of ISO 26262: Functional Safety-Road

- Vehicles,” *Gen. Mot. Co.*, no. August, 2018.
- [14] F. Falcini and G. Lami, “Challenges in certification of autonomous driving systems,” *Proc. - 2017 IEEE 28th Int. Symp. Softw. Reliab. Eng. Work. ISSREW 2017*, pp. 286–293, 2017.
- [15] T. Menzel, G. Bagschik, and A. M. Maurer, “Scenarios for Development, Test and Validation of Automated Vehicles,” *IEEE Intell. Veh. Symp. Proc.*, vol. 2018-June, pp. 1821–1827, 2018.
- [16] F. McCaffery, R. V. O’Connor, and R. Messnarz, “Implementing Functional Safety Standards – Experiences from the Trials about Required Knowledge and Competencies (SafEUr),” *Commun. Comput. Inf. Sci.*, vol. 364 CCIS, no. June, 2013.
- [17] C. Schmittner, G. Griessnig, and Z. Ma, “Status of the Development of ISO / SAE 21434,” in *Systems, Software and Services Process Improvement.*, 2018, pp. 504–513.
- [18] M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Romer, “A security metric for structured security analysis of cyber-physical systems supporting SAE J3061,” *2016 2nd Int. Work. Model. Anal. Control Complex CPS, CPS Data 2016*, 2016.
- [19] G. Macher, C. Schmittner, O. Veledar, and E. Brenner, “ISO/SAE DIS 21434 Automotive Cybersecurity Standard - In a Nutshell,” in *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops. SAFECOMP 2020. Lecture Notes in Computer Science*, vol. 12235, no. September, 2020, pp. 123–135.
- [20] V. Sembera, “ISO / SAE 21434 Setting the Standard for Connected Cars ’ Cybersecurity,” *Trend Micro Res.*, 2020.
- [21] R. Yan, J. Yang, D. Zhu, and K. Huang, “Design verification and validation for reliable safety-critical autonomous control systems,” *Proc. IEEE 23rd Int. Conf. Eng. Complex Comput. Syst. ICECCS*, pp. 170–179, 2018.
- [22] M. Hirz, “An approach supporting integrated modeling and design of complex mechatronics products by the example of automotive applications,” *WMSCI 2018 - 22nd World Multi-Conference Syst. Cybern. Informatics, Proc.*, vol. 3, no. July, pp. 161–166, 2018.
- [23] VDA QMC Working Group 13 / Automotive SIG, “Automotive SPICE Process Assessment / Reference Model,” p. 132, 2017.
- [24] M. R. Zofka, S. Klemm, F. Kuhnt, T. Schamm, and J. M. Zollner, “Testing and validating high level components for automated driving: Simulation framework for traffic scenarios,” *IEEE Intell. Veh. Symp. Proc.*, vol. 2016-Augus, no. Iv, pp. 144–150, 2016.
- [25] C. Sánchez *et al.*, “A survey of challenges for runtime verification from advanced application domains (beyond software),” *Form. Methods Syst. Des.*, vol. 54, no. 3, pp. 279–335, 2019.
- [26] R. Cuer, L. Piétrac, E. Niel, S. Diallo, N. Minoiu-Enache, and C. Dang-Van-Nhan, “A

- formal framework for the safe design of the Autonomous Driving supervision,” *Reliab. Eng. Syst. Saf.*, vol. 174, no. May 2017, pp. 29–40, 2018.
- [27] E. De Gelder and J. P. Paardekooper, “Assessment of Automated Driving Systems using real-life scenarios,” *IEEE Intell. Veh. Symp. Proc.*, no. Iv, pp. 589–594, 2017.
- [28] A. Christensen *et al.*, “Key Considerations in the Development of Driving Automation Systems,” *Proc. 24th Int. Tech. Conf. Enhanc. Saf. Veh.*, pp. 1–14, 2015.
- [29] F. J. Belmonte, S. Martin, E. Sancristobal, J. A. Ruiperez-Valiente, and M. Castro, “Overview of Embedded Systems to Build Reliable and Safe ADAS and AD Systems,” *IEEE Intell. Transp. Syst. Mag.*, no. February 2020, pp. 2–14, 2020.
- [30] F. Falcini and G. Lami, “Deep learning in automotive: Challenges and opportunities,” *Commun. Comput. Inf. Sci.*, vol. 770, pp. 279–288, 2017.
- [31] J. Zhang and J. Li, “Testing and verification of neural-network-based safety-critical control software: A systematic literature review,” *Inf. Softw. Technol.*, vol. 123, no. December 2019, 2020.
- [32] M. Alloghani, D. Al-Jumeily, J. Mustafina, and J. A. Hussain, Abir and Ahmed, “A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science,” in *Supervised and Unsupervised Learning for Data Science*, no. January, 2019, pp. 3–21.
- [33] A. Khan, A. Sohail, U. Zahoor, and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020.
- [34] L. A. Curiel-Ramirez, R. A. Ramirez-Mendoza, J. Izquierdo-Reyes, M. R. Bustamante-Bello, and S. A. Navarro-Tuch, “Hardware in the loop framework proposal for a semi-autonomous car architecture in a closed route environment,” *Int. J. Interact. Des. Manuf.*, vol. 13, no. 4, pp. 1647–1658, 2019.
- [35] A. Eskandarian, C. Wu, and C. Sun, “Research Advances and Challenges of Autonomous and Connected Ground Vehicles,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 2, pp. 683–711, 2020.
- [36] S. D. Pendleton *et al.*, “Perception, planning, control, and coordination for autonomous vehicles,” *Machines*, vol. 5, no. 1, pp. 1–54, 2017.
- [37] T. Duy Son, A. Bhave, and H. Van Der Auweraer, “Simulation-Based Testing Framework for Autonomous Driving Development,” *Proc. - 2019 IEEE Int. Conf. Mechatronics, ICM 2019*, pp. 576–583, 2019.
- [38] Y. Jia, L. Guo, and X. Wang, “Real-time control systems,” in *Transportation Cyber-Physical Systems*, Elsevier Inc., 2018, pp. 81–113.
- [39] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, “A systematic review of perception system and simulators for autonomous vehicles research,” *Sensors*

(Switzerland), vol. 19, no. 3, 2019.

- [40] K. G. Panda, D. Agrawal, A. Nshimiyimana, and A. Hossain, "Effects of environment on accuracy of ultrasonic sensor operates in millimetre range," *Perspect. Sci.*, vol. 8, pp. 574–576, 2016.
- [41] R. Stiawan, A. Kusumadjati, N. S. Aminah, M. Djamal, and S. Viridi, "An Ultrasonic Sensor System for Vehicle Detection Application," *J. Phys. Conf. Ser.*, vol. 1204, no. 1, pp. 4–10, 2019.
- [42] J. Z. Varghese and R. G. Boone, "Overview of Autonomous Vehicle Sensors and Systems," *Int. Conf. Oper. Excell. Serv. Eng.*, pp. 178–191, 2015.
- [43] S. Winkler, H. Yu, and Z. Zhou, "Tangible mixed reality desktop for digital media management," *Stereosc. Displays Virtual Real. Syst. XIV*, vol. 6490, no. 64901, p. 64901S, 2007.
- [44] D. O. F. Imu, "Miniature Inertial Measurement Unit," in *Space Microsystems and Micro/nano Satellites*, 2018, pp. 233–293.
- [45] R. D. Christ and R. L. Wernli, "Navigational Sensors," in *The ROV Manual*, Elsevier Ltd, 2014, pp. 453–475.
- [46] A. Schneider and H. Feussner, "Tracking and Navigation Systems," *Biomed. Eng. Gastrointest. Surg.*, pp. 443–472, 2017.
- [47] A. Sahoo, S. K. Dwivedy, and P. S. Robi, "Advancements in the field of autonomous underwater vehicle," *Ocean Eng.*, vol. 181, no. March, pp. 145–160, 2019.
- [48] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5515 LNCS, pp. 117–139, 2009.
- [49] S. Aradi, "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," *arXiv*, pp. 1–14, 2020.
- [50] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano, "Control Architecture Design for Autonomous Vehicles," *2018 IEEE Conf. Control Technol. Appl. CCTA 2018*, pp. 404–411, 2018.
- [51] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A Review of Motion Planning for Highway Autonomous Driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 1826–1848, 2020.
- [52] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH J.*, pp. 1–14, 2014.
- [53] A. Al Jawahiri, "Spline-based Trajectory Generation for Autonomous Truck-Trailer Vehicles in Low Speed Highway Scenarios," 2018.
- [54] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp.

1135–1145, 2016.

- [55] Q. Li, Z. Zeng, B. Yang, and T. Zhang, “Hierarchical route planning based on taxi GPS-trajectories,” *2009 17th Int. Conf. Geoinformatics, Geoinformatics 2009*, 2009.
- [56] D. González, J. Pérez, R. Lattarulo, V. Milanés, and F. Nashashibi, “Continuous curvature planning with obstacle avoidance capabilities in urban scenarios,” *2014 17th IEEE Int. Conf. Intell. Transp. Syst. ITSC 2014*, pp. 1430–1435, 2014.
- [57] C. Samak, T. Samak, and S. Kandhasamy, “Control strategies for autonomous vehicles,” *arXiv*, 2020.
- [58] H. Badis and A. Rachedi, “Modeling tools to evaluate the performance of wireless multi-hop networks,” in *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications*, Elsevier Inc., 2015, pp. 653–682.
- [59] D. Elliott, W. Keen, and L. Miao, “Recent advances in connected and automated vehicles,” *J. Traffic Transp. Eng. (English Ed.)*, vol. 6, no. 2, pp. 109–131, 2019.
- [60] E. Fazeldehkordi, I. S. Amiri, and O. A. Akanbi, “Literature Review,” in *A Study of Black Hole Attack Solutions*, 2016, pp. 7–57.
- [61] M. Fleury, N. N. Qadri, M. Altaf, and M. Ghanbari, “Robust Video Streaming over MANET and VANET,” in *Streaming Media Architectures, Techniques, and Applications*, no. January, 2011.
- [62] J. Deng, L. Yu, Y. Fu, O. Hambolu, and R. R. Brooks, “Security and Data Privacy of Modern Automobiles,” in *Data Analytics for Intelligent Transportation Systems*, 2017, pp. 131–163.
- [63] NHTSA, “Vehicle-to-Vehicle Communication Technology Fact Sheet,” pp. 1–4, 2014.
- [64] J. Barrachina *et al.*, “Road side unit deployment: A density-based approach,” *IEEE Intell. Transp. Syst. Mag.*, vol. 5, no. 3, pp. 30–39, 2013.
- [65] T. Nadeem and P. Shankar, “A Comparative Study of Data Dissemination Models for VANETs,” in *2006 Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services*, 2006, pp. 1–10.
- [66] A. Paul, N. Chilamkurti, A. Daniel, and S. Rho, “Evaluation of vehicular network models,” in *Intelligent Vehicular Networks and Communications*, 2017, pp. 77–112.
- [67] O. Gama *et al.*, “Evaluation of push and pull communication models on a VANET with virtual traffic lights,” *Inf.*, vol. 11, no. 11, pp. 1–20, 2020.
- [68] R. Shaibani and A. Zahary, “Survey of Context-Aware Video Transmission over Vehicular Ad-Hoc Networks (VANETs),” *EAI Endorsed Trans. Mob. Commun. Appl.*, vol. 4, no. 15, p. 156089, 2019.
- [69] S. Ur Rehman, M. A. Khan, T. A. Zia, and L. Zheng, “Vehicular Ad-Hoc Networks (VANETs) - An Overview and Challenges,” *J. Wirel. Netw. Commun.*, vol. 2013, no. 3, pp. 29–38, 2013.

- [70] F. Cunha *et al.*, “Data communication in VANETs: Protocols, applications and challenges,” *Ad Hoc Networks*, vol. 44, pp. 90–103, 2016.
- [71] K. Heineke, A. Ménard, F. Södergren, and M. Wrulich, “Development in the mobility technology ecosystem—how can 5G help?,” *Automotive & Assembly*, no. June, pp. 1–10, 2019.
- [72] K. Kiela *et al.*, “Review of V2X-IoT standards and frameworks for ITS applications,” *Appl. Sci.*, vol. 10, no. 12, 2020.
- [73] P. Coppola and F. Silvestri, “Autonomous vehicles and future mobility solutions,” in *Autonomous Vehicles and Future Mobility*, Elsevier Inc., 2019, pp. 1–15.
- [74] S. Vadi, R. Bayindir, A. M. Colak, and E. Hossain, “A review on communication standards and charging topologies of V2G and V2H operation strategies,” *Energies*, vol. 12, no. 19, pp. 1–28, 2019.
- [75] L. Li, W. L. Huang, Y. Liu, N. N. Zheng, and F. Y. Wang, “Intelligence testing for autonomous vehicles: A new approach,” *IEEE Trans. Intell. Veh.*, vol. 1, no. 2, pp. 158–164, 2016.
- [76] H.-P. Schöner, “Simulation in development and testing of autonomous vehicles,” in *18th Stuttgart International Symposium*, 2018, pp. 225–237.
- [77] M. Luckcuck, M. Farrell, L. Dennis, C. Dixon, and M. Fisher, “Formal Specification and Verification of Autonomous Robotic Systems: A Survey,” *ACM Comput. Surv.*, vol. 52, no. October 2019, p. 41, 2018.
- [78] Intellias, “Testing Automotive Control Software,” 2008.
- [79] O. Stephen and K. Oriaku, “Software Development Methodologies: Agile Model Vs V-Model,” *Int. J. Eng. Tech. Res.*, vol. 2, no. 11, pp. 108–113, 2014.
- [80] B. Liu, H. Zhang, and S. Zhu, “An incremental V-model process for automotive development,” *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 0, pp. 225–232, 2016.
- [81] S. Balaji and M. Murugaiyan, “Waterfall vs v-model vs agile : A comparative study on SDLC,” *Int. J. Inf. Technol. Bus. Manag.*, vol. 2, no. 1, pp. 26–30, 2012.
- [82] O. Emmerich, H. Wang, A. Darlington, G. Garcia, and B. Gao, “A Systems Engineering Framework and Application to an Open Automated Driving Platform,” in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, 2020, pp. 2393–2398.
- [83] H. J. Vishnukumar, B. Butting, C. Muller, and E. Sax, “Machine learning and deep neural network - Artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation,” *2017 Intell. Syst. Conf.*, no. September, pp. 714–721, 2017.
- [84] C. Sippl, F. Bock, C. Lauer, A. Heinz, T. Neumayer, and R. German, “Scenario-based systems engineering: An approach towards automated driving function development,”

- SysCon 2019 - 13th Annu. IEEE Int. Syst. Conf. Proc.*, pp. 1–8, 2019.
- [85] S. Ulbrich *et al.*, “Testing and Validating Tactical Lane Change Behavior Planning for Automated Driving,” in *Automated Driving: Safer and More Efficient Future Driving*, 2017, pp. 451–471.
- [86] H. K. N. Leung and P. W. L. Wong, “A study of user acceptance tests,” *Softw. Qual. J.*, vol. 6, no. 2, pp. 137–149, 1997.
- [87] S. Hallerbach, Y. Xia, U. Eberle, and F. Koester, “Simulation-based Identification of Critical Scenarios for Cooperative and Automated Vehicles,” *SAE Tech. Pap.*, vol. 2018-April, pp. 1–12, 2018.
- [88] G. Kanter and J. Vain, “Model-based testing of autonomous robots using TestIt,” *J. Reliab. Intell. Environ.*, vol. 6, no. 1, pp. 15–30, 2020.
- [89] C. Galko, R. Rossi, and X. Savatier, “Vehicle-hardware-in-the-loop system for ADAS prototyping and validation,” *Proc. - Int. Conf. Embed. Comput. Syst. Archit. Model. Simulation, SAMOS 2014*, no. Samos Xiv, pp. 329–334, 2014.
- [90] A. Soltani and F. Assadian, “A Hardware-in-the-Loop Facility for Integrated Vehicle Dynamics Control System Design and Validation,” *IFAC-PapersOnLine*, vol. 49, no. 21, pp. 32–38, 2016.
- [91] A. Albers and T. Düser, “Implementation of a Vehicle-in-the-Loop Development and Validation Platform,” *FISITA World Automot. Congr.*, pp. 1–10, 2010.
- [92] V. De Oliveira Neves, M. E. Delamaro, P. C. Másiero, C. C. T. Mendes, and D. F. Wolf, “Structural testing of autonomous vehicles,” *Proc. Int. Conf. Softw. Eng. Knowl. Eng. SEKE*, vol. 2013, no. January, pp. 200–205, 2013.
- [93] M. Gyllenhammar *et al.*, “Towards an Operational Design Domain That Supports the Safety Argumentation of an Automated Driving System,” *10th Eur. Congr. Embed. Real Time Syst.*, pp. 1–10, 2020.
- [94] P. Koopman and F. Fratrik, “How many operational design domains, objects, and events?,” *CEUR Workshop Proc.*, vol. 2301, pp. 1–4, 2019.
- [95] A. M. Shaaban, C. Schmittner, T. Gruber, A. B. Mohamed, G. Quirchmayr, and E. Schikuta, “Ontology-based model for automotive security verification and validation,” *ACM Int. Conf. Proceeding Ser.*, 2019.
- [96] Y. Li, J. Tao, and F. Wotawa, “Ontology-based test generation for automated and autonomous driving functions,” *Inf. Softw. Technol.*, vol. 117, no. October 2019, 2020.
- [97] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, “Testing advanced driver assistance systems using multi-objective search and neural networks,” *ASE 2016 - Proc. 31st IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 63–74, 2016.
- [98] A. Gambi, M. Mueller, and G. Fraser, “Automatically testing self-driving cars with search-based procedural content generation,” *ISSTA 2019 - Proc. 28th ACM SIGSOFT*

- Int. Symp. Softw. Test. Anal.*, pp. 273–283, 2019.
- [99] A. Calo, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, “Generating Avoidable Collision Scenarios for Testing Autonomous Driving Systems,” *Proc. - 2020 IEEE 13th Int. Conf. Softw. Testing, Verif. Validation, ICST 2020*, pp. 375–386, 2020.
- [100] R. Ben Abdesslem, S. Nejati, L. C. Briand, and T. Stifter, “Testing vision-based control systems using learnable evolutionary algorithms,” *Proc. - Int. Conf. Softw. Eng.*, pp. 1016–1026, 2018.
- [101] S. R. K B and S. J., “Model Checkers - Tools and Languages for System Design - A Survey,” *Comput. Sci. Inf. Technol. (CS IT)*, pp. 39–51, 2016.
- [102] K. S. Swanson, A. A. Brown, S. N. Brennan, and C. M. Lajambe, “Extending driving simulator capabilities toward Hardware-in-the-Loop testbeds and remote vehicle interfaces,” *IEEE Intell. Veh. Symp. Work. (IV Work.)*, pp. 115–120, 2013.
- [103] R. Marinescu, C. Seceleanu, H. Le Guen, and P. Pettersson, “A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs,” in *Advances in Computers*, vol. 98, 2015, pp. 89–140.
- [104] J. Arcile, R. Devillers, and H. Klaudel, “VerifCar: a framework for modeling and model checking communicating autonomous vehicles,” *Proc. 19th Int. Conf. Auton. Agents Multiagent Syst. (AAMAS 2020), Auckland, New Zeal.*, vol. 33, no. 3, pp. 2126–2127, 2019.
- [105] A. Chakrapani Rao, A. C. Rajeev, and A. Yeolekar, “Applying design verification tools in automotive software V&V,” *SAE 2011 World Congr. Exhib.*, 2011.
- [106] M. Althoff and S. Lutz, “Automatic Generation of Safety-Critical Test Scenarios for Collision Avoidance of Road Vehicles,” *IEEE Intell. Veh. Symp. Proc.*, vol. 2018-June, no. Iv, pp. 1326–1333, 2018.
- [107] E. Cioroica, F. Pudlitz, I. Gerostathopoulos, and T. Kuhn, “Simulation methods and tools for collaborative embedded systems: with focus on the automotive smart ecosystems,” *SICS Software-Intensive Cyber-Physical Syst.*, vol. 34, no. 12, p. 11, 2019.
- [108] J. M. Ramírez-Cortés, P. Gómez-Gil, J. Martínez-Carballido, and F. López-Larios, “A LabVIEW-based Autonomous Vehicle Navigation System using Robot Vision and Fuzzy Control,” *Ing. Investig. y Tecnol.*, vol. 12, no. 2, pp. 129–136, 2011.
- [109] C. E. Tuncali, G. Fainekos, D. Prokhorov, H. Ito, and J. Kapinski, “Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components,” *2018 IEEE Intell. Veh. Symp.*, no. Iv, pp. 1555–1562, 2018.
- [110] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, “CARLA: An open urban driving simulator,” *Proc. 1st Annu. Conf. Robot Learn.*, pp. 1–16, 2017.
- [111] R. Lattarulo, J. Pérez, and M. Dendaluze, “A complete framework for developing and testing automated driving controllers,” *IFAC Pap. OnLine*, vol. 50, no. 1, pp. 258–263,

2017.

- [112] S. A. Fayazi, A. Vahidi, and A. Luckow, "A Vehicle-in-the-Loop (VIL) verification of an all-autonomous intersection control scheme," *Transp. Res. Part C Emerg. Technol.*, vol. 107, no. December 2017, pp. 193–210, 2019.
- [113] M. Renner, N. Münzenberger, J. Von Hammerstein, S. Lins, and A. Sunyaev, "Challenges of vehicle-to-everything communication. Interviews among industry experts," *Proc. 15th Int. Conf. Bus. Inf. Syst. Wirtschaftsinformatik*, no. March, 2020.
- [114] M. Barbier *et al.*, "Validation of perception and decision-making systems for autonomous driving via statistical model checking," *IEEE Intell. Veh. Symp. Proc.*, vol. 2019-June, no. Iv, pp. 252–259, 2019.
- [115] B. Xu, Q. Li, T. Guo, and D. Du, "A scenario-based approach for formal modelling and verification of safety properties in automated driving," *IEEE Access*, vol. 7, pp. 140566–140587, 2019.
- [116] B. Van Acker, B. J. Oakes, M. Moradi, P. Demeulenaere, and J. Denil, "Validity frame concept as effort-cutting technique within the verification and validation of complex cyber-physical systems," in *MODELS '20: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–10.
- [117] T. D. Gillespie, *Fundamentals of vehicle dynamics (Vol. 114)*. SAE Technical Paper. 1992.
- [118] K. Yoneda, T. Iida, T. H. Kim, R. Yanase, M. Aldibaja, and N. Suganuma, "Trajectory optimization and state selection for urban automated driving," *Artif. Life Robot.*, vol. 23, no. 4, pp. 474–480, 2018.
- [119] J. Hou, G. F. List, and X. Guo, "New algorithms for computing the time-to-collision in freeway traffic simulation models," *Comput. Intell. Neurosci.*, vol. 2014, 2014.
- [120] Y. Tateyama *et al.*, "Observation of driver's behavior at narrow roads using immersive car driving simulator," *Proc. - VRCAI 2010, ACM SIGGRAPH Conf. Virtual-Reality Contin. Its Appl. to Ind.*, no. December, pp. 391–396, 2010.
- [121] T. Takahama and D. Akasaka, "Model Predictive Control Approach to Design Practical Adaptive Cruise Control for traffic jam," *Int. J. Automot. Eng.*, vol. 9, no. 3, pp. 99–104, 2018.
- [122] F. F. Ling, *Vehicle Dynamics and Control. Mechanical Engineering Series*. Springer, Boston. 2006.
- [123] P. F. Lima, M. Trincavelli, J. Martensson, and B. Wahlberg, "Clothoid-based model predictive control for autonomous driving," *2015 Eur. Control Conf. ECC 2015*, no. April 2016, pp. 2983–2990, 2015.
- [124] E. J. C. Nacpil, Z. Wang, R. Zheng, T. Kaizuka, and K. Nakano, "Design and evaluation of

a surface electromyography-controlled steering assistance interface,” *Sensors (Switzerland)*, vol. 19, no. 6, 2019.