

University of Southampton Research Repository

Copyright © and Moral Rights for this thesis and, where applicable, any accompanying data are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis and the accompanying data cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content of the thesis and accompanying research data (where applicable) must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holder/s.

When referring to this thesis and any accompanying data, full bibliographic details must be given, e.g.

Thesis: Fahad Alotaibi (2024) “A Rigorous Analysis Template Process to Capture the Safety Properties of Self-Driving Vehicle Systems”, University of Southampton, School of Electronics and Computer Science, PhD Thesis, pages 1-315.

Data: Fahad Alotaibi (2024) “A Rigorous Analysis Template Process to Capture the Safety Properties of Self-Driving Vehicle Systems”.

UNIVERSITY OF SOUTHAMPTON

**A Rigorous Analysis Template Process to
Capture the Safety Properties of
Self-Driving Vehicle Systems**

by

Fahad Alotaibi

ORCID: [0000-0001-8545-907X](https://orcid.org/0000-0001-8545-907X)

Thesis for the degree of Doctor of Philosophy

in the

**Faculty of Engineering and Physical Science
School of Electronics and Computer Science**

10 March 2024

University of Southampton

Abstract

Faculty of Engineering and Physical Science
School of Electronics and Computer Science

Doctor of Philosophy

**A Rigorous Analysis Template Process to Capture the Safety Properties of
Self-Driving Vehicle Systems**

by **Fahad Alotaibi**

Self-Driving Vehicles (SDVs) are seen as a significant advancement in the automotive domain, hinting at a future where human drivers might be rendered obsolete. However, even with the advancements in SDV technology, the need for human drivers is still recognised. The incorporation of human drivers into SDVs introduces unique and significant challenges. The significance of human driver and SDV interactions cannot be overstated, especially when the SDV relies on the human driver as a fallback option during hazardous driving events. To address this critical aspect, this thesis presents a methodology termed the Rigorous Analysis Template Process (RATP).

RATP establishes an analytical journey to develop a comprehensive framework ensuring safety and optimal cooperation between human drivers and SDV systems. It represents an evolution in existing work on analysing system safety and provides a more rigorous systematic strategy for SDV systems. It involves both systematic analysis and formal methods to evaluate safety in SDV systems.

Drawing strength from a combination of both systematic analysis and formal methods, RATP adeptly identifies high-level safety requirements and develops a rigorous model to investigate issues and assumptions that may arise during the operations of SDV systems. One of the key benefits of RATP is its modularity, offering researchers and developers the ability to systematically analyse system behaviours from a high-abstraction view down to a more detailed view. The conclusion of this research presents a robust set of modelling patterns that act as a blueprint for the future development of SDV systems.

RATP is demonstrated with a case study that explores the various functionalities of an SDV system to evolve the methodology into a mature state. Finally, this thesis presents a discussion on future improvements that could be undertaken to develop the methodology further.

Contents

List of Figures	ix
List of Tables	xi
Declaration of Authorship	xiii
Acknowledgements	xv
Abbreviations	xvii
1 Introduction	1
1.1 Problem Statement	2
1.1.1 Complexity of an SDV System	3
1.1.2 Sharing Mechanisms of an SDV Control	3
1.1.3 Insight into System Behaviours During Dynamic Driving Tasks	3
1.2 Research Aims and Questions	4
1.2.1 Research Question 1	5
1.2.2 Research Question 2	5
1.2.3 Research Question 3	6
1.2.4 Research Question 4	6
1.3 Research Plan	7
1.4 Contribution	8
1.5 Outline of Thesis	9
2 Overview: Self-Driving Vehicle Systems	11
2.1 SDV Human and Machine Interactions	11
2.2 Taxonomy Requirements	14
2.2.1 Automation Levels	18
2.2.1.1 Traditional Classifications of Autonomy	18
2.2.1.2 Classifications of Autonomy for SDV systems	19
2.3 Generic System Architecture	21
2.4 Discussion	23
2.5 Conclusion	24
3 Safety Analysis Methodologies	27
3.1 Hazard Identification Methods	27
3.1.1 Hazard and Operability Analysis	27
3.1.2 Failure Mode and Effects Analysis	29

3.1.3	Fault Tree Analysis	30
3.1.4	Systems Theoretic Process Analysis	31
3.1.4.1	Theoretical Background of the STPA Method	31
3.1.4.2	Overview of the STPA Method	32
3.2	Industrial Safety Methodologies	35
3.2.1	Safety Standards	35
3.2.2	Waymo: Safety Methodology	36
3.2.3	GM: Safety Methodology	37
3.3	Formal Methodologies	38
3.3.1	Overview of Formal Methods: Definition, Advantages and Limitations	38
3.3.2	Formal Specification Approaches	39
3.3.2.1	Event-B Modelling Method	40
3.3.2.2	Logic-based Approaches	43
3.3.2.3	Other Approaches	44
3.4	Discussion	46
3.5	Conclusion	47
4	Case Study	49
4.1	Automated Lane Centring	49
4.1.1	ALC Features for Lane Stability in SDV	50
4.1.2	DMS Features for Safeguarding Driver Alertness in SDV	51
4.1.3	Further Analysis of the ALC on the OpenPilot Platform	52
4.1.3.1	Overview of OpenPilot Software with LKA, ACC and DMS Features	53
4.1.3.2	LKA and ACC Control in OpenPilot	54
4.1.3.3	DMS in OpenPilot	55
4.2	DDT Features Derived from the ALC System	55
4.2.1	Feature 1: High-Level Interactions Between ALC System and Human Drivers	56
4.2.2	Feature 2: The LKA and DMS Functions in the ALC System	56
4.2.3	Feature 3: The LKA and ACC Functions in the ALC System	57
4.2.4	Feature 4: Modelling the Driver Reactions in the ALC System	58
4.3	Conclusion	58
5	Rigorous Analysis Template	59
5.1	Justifications	59
5.1.1	Motivation Behind Automation Aspects in Template	60
5.1.2	Hazard Identification Methods	61
5.1.3	Formal Methods	62
5.2	Rigorous Analysis Template	62
5.2.1	Development of a Template	63
5.2.2	Template Instantiation Process for ALC system	65
5.2.3	Analysis Process	67
5.2.3.1	Systems Theoretic Process Analysis	67
5.2.3.2	Modelling in Event-B	72
5.3	Discussion	75

5.4	Related Work	76
5.5	Conclusion	77
6	Rigorous Analysis Template Process	79
6.1	Systematic Analysis Steps of the RATP	80
6.1.1	Instantiating System Boundary Diagrams	80
6.1.2	Identifying the Purpose of Analysis	82
6.1.3	Creating/Refining Hierarchical Control Structure	83
6.1.4	Identifying Unsafe Control Actions	84
6.1.5	Specifying/Refining the Event-B Model	86
6.2	Modelling Patterns of SDV Systems	87
6.2.1	Abstraction Level (Layer 0)	88
6.2.2	High-Level Analysis Process (Layer 1)	91
6.2.3	Second Analysis Process (Layer 2)	95
6.2.4	Third Analysis Process (Layer 3)	107
6.3	Discussion	115
6.4	Advantages, Concerns and Related Work	116
6.4.1	Advantages	116
6.4.2	Limitations	117
6.4.3	Related Work	118
6.4.3.1	STPA Steps in RATP	118
6.4.3.2	Formal Methods with STPA	119
6.5	Conclusion	120
7	LKA and DMS Functions in the ALC System	121
7.1	Modelling Pattern of the ALC System	121
7.1.1	Abstraction Level (ALC-Layer 0)	122
7.1.2	High-Level Analysis Process (ALC-Layer 1)	125
7.1.3	Second Analysis Process (ALC-Layer 2)	129
7.1.4	Third Analysis Process (ALC-Layer 3)	142
7.2	Conclusion	148
8	LKA and ACC Functions in the ALC System	151
8.1	Adapting ALC Layers for ACC Functionality	152
8.1.1	Instantiating System Boundary Diagram	152
8.1.2	Identifying the Purpose of Analysis	152
8.1.3	Updating Hierarchical Control Structure	154
8.1.4	Identifying Unsafe Control Actions	154
8.1.5	Updating the Event-B Model	157
8.1.6	Proof Statistics	162
8.2	Discussion	163
8.3	Evaluation of RATP Method in the ALC Case Study	164
8.4	Conclusion	165
9	Modelling the Driver Reactions	167
9.1	Intervention Timing Pattern	168
9.1.1	Layer 4: Intervention Timing Pattern	168
9.2	Modelling Driver Reactions in the ALC System	177

9.2.1	ALC-Layer 4: Modelling the Intervention Timing Pattern	177
9.3	Discussion	184
9.4	Related Work	185
9.5	Conclusion	186
10	Conclusions	187
10.1	Contributions	187
10.2	Limitations	189
10.3	RATP Methodology: Lessons from ALC Case Study	189
10.3.1	Analysing Dynamic Driving Tasks of SDV Systems	189
10.3.2	Bridging STPA and Event-B: A Dual Strategy Examination	190
10.3.3	Modelling New Behaviours and Features	191
10.4	Research Opportunities	192
Appendix A	The NHTSA Safety Protocol	195
Appendix B	Automated Lane Centring in OpenPilot	199
Appendix B.1	Recording Behaviours using Python Scripts	199
Appendix B.2	Assessing and Understanding Recorded Behaviours	203
Appendix B.3	Driver Monitoring System in Autonomous Vehicles	206
Appendix B.3.1	Sensitive Monitoring Feature	207
Appendix C	Modelling High-Level System Aspects in ALC System	209
Appendix C.1	Context c0	209
Appendix C.2	Machine m0	210
Appendix C.3	Machine m1	212
Appendix D	Modelling Patterns for SDV Systems	217
Appendix D.1	Abstract Level	218
Appendix D.1.1	Context c0	218
Appendix D.1.2	Machine m0	218
Appendix D.2	First Refinement	220
Appendix D.2.1	Context c1	220
Appendix D.2.2	Machine m1	221
Appendix D.3	Second Refinement	223
Appendix D.3.1	Context c2	223
Appendix D.3.2	Machine m2	226
Appendix D.4	Third Refinement	233
Appendix D.4.1	Machine m3	233
Appendix E	Modelling LKA and DMS Functions in the ALC System	239
Appendix E.1	Abstract Level	240
Appendix E.1.1	Context c0	240
Appendix E.1.2	Machine m0	240
Appendix E.2	First Refinement	242
Appendix E.2.1	Context c1	242
Appendix E.2.2	Machine m1	242
Appendix E.3	Second Refinement	244

Appendix E.3.1	Context c2	244
Appendix E.3.2	Machine m2	246
Appendix E.4	Third Refinement	255
Appendix E.4.1	Machine m3	255
Appendix F	Modelling LKA, DMS and ACC Functions in the ALC System	263
Appendix F.1	Abstract Level	264
Appendix F.1.1	Context c0	264
Appendix F.1.2	Machine m0	264
Appendix F.2	First Refinement	265
Appendix F.2.1	Context c1	265
Appendix F.2.2	Machine m1	266
Appendix F.3	Second Refinement	269
Appendix F.3.1	Context c2	269
Appendix F.3.2	Machine m2	270
Appendix F.4	Third Refinement	280
Appendix F.4.1	Machine m3	280
Appendix G	Intervention Timing Pattern	287
Appendix G.1	Layer 4: Driver Reactions in Modelling Patterns	287
Appendix G.1.1	Intervention Timing Pattern	287
Appendix G.2	ALC-Layer 4: Driver Reactions in the ALC System	291
Appendix G.2.1	Machine m4	292
References		303

List of Figures

1.1	Research plan activities	7
2.1	DDT vs traditional driving tasks	17
2.2	Types of human interaction with autonomy, based on stage of autonomous functions and Sheridan and Verplank’s taxonomy approach	20
2.3	Comparison of NHTSA and SAE approaches to classification of autonomy levels	21
2.4	Abstract components of SDVs	23
3.1	Major components of an abstract control structure based on STAMP [86, p. 75]	32
3.2	Overview of basic STPA steps, adapted from [84]	34
3.3	Finite-state automata of IARA’s vehicle for handling pedestrians at pedestrian crossings	45
4.1	Relationships between OpenPilot’s internal modules	53
5.1	Overview of processes in RAT	63
5.2	Template for SDV systems	65
5.3	Instantiating process, from generic template to ALC concrete template	68
5.4	Control structure of high-level system interactions for an ALC system	70
6.1	Systematic process of RATP approach	81
6.2	System boundary diagram, adapted from [84, p. 17].	81
6.3	System boundary diagrams in RATP approach	82
6.4	Overview of defining the purpose of analysis, from [84, p. 16]	82
6.5	Overview of defining the purpose of analysis in RATP approach	83
6.6	Simple form of control structure, adapted from [84, p. 24]	84
6.7	Overview of constructing the control structure in RATP approach	85
6.8	Overview of defining unsafe control actions, adapted from [84, p. 42].	85
6.9	Overview of defining unsafe control actions in RATP approach	86
6.10	Overview of specifying formal models in RATP approach	87
6.11	System boundary diagram in RATP Layer 0	88
6.12	Control structure in RATP Layer 0	89
6.13	System boundary diagram in RATP Layer 1	92
6.14	Control structure in RATP Layer 1	93
6.15	System boundary diagram in RATP Layer 2	96
6.16	Control structure in RATP Layer 2	98
6.17	System boundary diagram in RATP Layer 3	108

6.18	Control structure in RATP Layer 3	109
7.1	System boundary diagram in ALC Layer 0	123
7.2	Control structure in ALC Layer 0	124
7.3	System boundary diagram in ALC Layer 1	126
7.4	Control structure in ALC Layer 1	127
7.5	System boundary diagram in ALC Layer 2	131
7.6	Control structure in ALC Layer 2	132
7.7	System boundary diagram in ALC Layer 3	143
7.8	Control structure in ALC Layer 3	144
8.1	System boundary diagram incorporating speed identification	153
8.2	Control structure incorporating speed identification	155
9.1	System boundary diagram in RATP Layer 4	169
9.2	Control structure in RATP Layer 4	170
9.3	Creation of intervention timer	173
9.4	Time progression in intervention timer	174
9.5	Human's response window time before alert notification	175
9.6	Human's response window time after alert notification	176
9.7	System boundary diagram incorporating intervention timing pattern	178
9.8	Control structure in ALC Layer 4	179
Appendix B.1	Testing ALC system in OpenPilot under diverse weather settings	204
Appendix B.2	Desired path probability over the time period, illustrating changes in lane detection scores	204
Appendix B.3	Changes in the steering angle over time, demonstrating adjustments in the vehicle's physical wheels	205
Appendix B.4	OpenPilot's 'Take Control' request during excessive steering adjustment	206
Appendix B.5	Summary of the detection process of a sensitive monitoring feature based on the Dlib algorithm	208
Appendix D.1	Modelling Patterns Prover Statistics.	217
Appendix E.1	Modelling LKA and DMS functions, prover statistics	239
Appendix F.1	Modelling LKA, DMS and ACC functions, prover statistics	263
Appendix G.1	Intervention timing pattern, prover statistics.	288
Appendix G.2	Prover statistics for modelling driver reactions in ALC system	291

List of Tables

2.1	Summary of automation levels, adapted from [101]	25
3.1	Example of HAZOP keywords and guidewords, taken from [15]	28
5.1	Generic autonomous controller aspects	64
5.2	Generic human fallback aspects	65
5.3	Autonomous controller aspects.	66
5.4	Human fallback aspects	67
5.5	Unsafe control actions in steering variable of ALC system	70
5.6	Unsafe control actions in awareness level of driver	71
5.7	Unsafe control actions in driver correction action	71
5.8	Safety requirements (SR1 to SR9) in formal model.	75
6.1	Unsafe control actions in RATP Layer 0	90
6.2	Control actions and feedback loops in formal model for RATP Layer 0	90
6.3	Safety requirement (SR0.1) in formal model for RATP layer 0	91
6.4	Unsafe control actions in RATP Layer 1	94
6.5	Control actions and feedback loops in formal model for RATP Layer 1	94
6.6	Safety requirements(SR1.1 and SR1.2) in formal model for RATP Layer 1	95
6.7	Unsafe control actions of perception component in RATP Layer 2	98
6.8	Unsafe control actions of decision component in RATP Layer 2	99
6.9	Unsafe control actions of control component in RATP Layer 2	100
6.10	Control actions and feedback loops in formal model for RATP Layer 2	101
6.11	Safety requirements (SR2.1 to SR2.8) in formal model for RATP Layer 2	107
6.12	Unsafe control actions of DMS component in RATP Layer 3	110
6.13	Unsafe control actions of driver intervention in RATP Layer 3	111
6.14	Control actions and feedback loops in formal model for RATP Layer 3	111
6.15	Safety requirements (SR3.1 to SR3.5) in formal model for RATP Layer 3	115
7.1	Unsafe control actions in ALC Layer 0	124
7.2	Control actions and feedback loops in formal model for ALC Layer 0	124
7.3	Safety requirement (SR0) in formal model for ALC Layer 0.	125
7.4	Unsafe control actions in ALC Layer 1	128
7.5	Control actions and feedback loops in formal model for ALC Layer 1	128
7.6	Safety requirements (SR1.1 and SR1.2) in formal model for ALC Layer 1	130
7.7	Unsafe control actions of perception component in ALC Layer 2	133
7.8	Unsafe control actions of decision component in ALC Layer 2	134
7.9	Unsafe control actions of control component in ALC Layer 2	135
7.10	Control actions and feedback loops in formal model for ALC Layer 2	135

7.11	Safety requirements(SR2.1 and SR2.10) in formal model for ALC Layer 2	141
7.12	Unsafe control actions of DMS component in ALC Layer 3	145
7.13	Unsafe control actions of driver intervention in ALC Layer 3	145
7.14	Control actions and feedback loops in formal model for ALC Layer 3 . .	146
7.15	Safety requirements (SR3.1 to SR3.5) in formal model for ALC Layer 3 .	148
8.1	Unsafe control actions in speed variable of ALC system	155
8.2	New unsafe control actions in the perception component	156
8.3	New unsafe control actions in the control component	157
8.4	Control actions and feedback loop incorporating speed identification . .	157
8.5	Safety requirements (SR1 to SR7) in formal model for ACC functionality	162
8.6	Proof statistics of ALC case study	163
9.1	Unsafe control actions in RATP Layer 4	171
9.2	Unsafe control actions in ALC Layer 4	180
9.3	Feedback loop linked to ALC Layer 4	180
9.4	Safety requirements (SR4 to SR6) in formal model for ALC Layer 4 . . .	183
Appendix A.1	Conceptual details of the NHTSA safety framework.	196
Appendix A.2	Conceptual details of the NHTSA safety framework (continued).	197

Declaration of Authorship

I, **Fahad Alotaibi**, declare that this thesis entitled *A Rigorous Analysis Template Process to Capture the Safety Properties of Self-Driving Vehicle Systems* and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: [11, 12, 13];

Signed:.....

Date:.....

Acknowledgements

First and foremost, I wish to express my deepest gratitude to my supervisor, Dr Son Hoang. His extensive knowledge, patience, and guidance have been invaluable throughout this academic journey. It has been of great privilege and honour working under his supervision.

I would also like to thank my second supervisor, Professor Michael Butler. His continued support and expert advice have made a substantial impact on my PhD research.

I am grateful to Dr Abdolbaghi Rezazadeh and Dr Nawfal Fadhel, who served as my internal examiners during the initial phases of my PhD. Additionally, I extend my thanks to Professor Yamin Ait Ameer for his valuable feedback during my final PhD viva. Lastly, I would like to express my appreciation to all my friends and colleagues in the Cyber Physical Systems (CPS) research group.

I am forever indebted to my family for their unconditional love, support and encouragement throughout my research. A special thanks to my brother, Juwayid, for taking care of our family during my time abroad.

Finally, I gratefully acknowledge the generous funding provided by King Saud University, without which this work would not have been possible. Their support has been instrumental in bringing this research to fruition.

This thesis was completed because of the help from these individuals. Their contributions are something I will always appreciate.

Abbreviations

<i>AAA</i>	American Automobile Association
<i>ACC</i>	Adaptive Cruise Control
<i>ADAS</i>	Advanced Driver Assistance Systems
<i>AI</i>	Artificial Intelligence
<i>ASIL</i>	Automotive Safety Integrity Level
<i>ALC</i>	Automated Lane Centring
<i>AS</i>	Autonomous System
<i>CA</i>	Control Action
<i>DDT</i>	Dynamic Driving Task
<i>DMS</i>	Driver Monitoring System
<i>FMEA</i>	Failure Mode and Effects Analysis
<i>FRA</i>	Facial Recognition Algorithm
<i>FTA</i>	Fault Tree Analysis
<i>FSA</i>	Finite-State Automata
<i>FL</i>	Feedback Loop
<i>GM</i>	General Motors
<i>HAZOP</i>	Hazard and Operability Analysis
<i>HMI</i>	Human-Machine Interface
<i>HITLS</i>	Human-In-The-Loop System
<i>IIC</i>	Industrial Internet Consortium
<i>ISO</i>	International Organization for Standardization
<i>LDW</i>	Lane Departure Warning
<i>LKA</i>	Lane Keeping Assist
<i>NHTSA</i>	National Highway Traffic Safety Administration
<i>NTSB</i>	National Transportation Safety Board
<i>ODD</i>	Operational Design Domain
<i>OEDR</i>	Object and Event Detection and Response
<i>PHA</i>	Preliminary Hazard Analysis
<i>POs</i>	Proof Obligations
<i>RAT</i>	Rigorous Analysis Template
<i>RATP</i>	Rigorous Analysis Template Process
<i>RPN</i>	Risk Priority Number

<i>SAE</i>	Society of Automotive Engineers
<i>SDV</i>	Self-Driving Vehicle
<i>STPA</i>	Systems Theoretic Process Analysis
<i>STAMP</i>	Systems Theoretic Accident Model and Processes
<i>SL</i>	System Loss
<i>SH</i>	System Hazard
<i>SC</i>	Safety Constraint
<i>SR</i>	Safety Requirement
<i>UCA</i>	Unsafe Control Action

Chapter 1

Introduction

Advances in the field of Artificial Intelligence (AI) are a key source for the development of Self-Driving Vehicles (SDVs). Being an Autonomous System (AS), an SDV involves multiple subsystems communicating and working together to achieve its goal [50]. SDVs rely on AI to perceive the environment, demonstrate the operational conditions and make driving decisions. At their basis, therefore, SDVs are intended to replace the actions of human drivers. The main aim of SDVs is to reduce collisions and improve passenger safety. According to the U.S. National Highway Traffic Safety Administration (NHTSA) [8], most road accidents are caused by driver behaviour. The NHTSA reports that 94% of crashes involve driver errors, while in only a few cases, extra factors, such as weather, may lead the driver to have a collision.

Although the idea of SDVs brings new technology for commercialisation, SDVs are not yet ready for consumers to operate them on public roads [132]. The main obstacle to doing so is ensuring the safe operations of an SDV in its environment. According to the Industrial Internet Consortium (IIC) [91], safety is defined as the optimal guarantee for ensuring the operation of a system ‘*without unacceptable risks either by physical injury or damage to the health of people and public properties*’. The concept of safety in SDVs is complicated and dependent upon multiple factors, including social and technical. According to a survey study of a thousand drivers conducted by the American Automobile Association (AAA) [45], nearly a quarter of participants would be fearful of riding in an SDV. Therefore, the acceptance of SDV technology remains low.

Moreover, an SDV is a *safety-critical system* and may jeopardise the lives of passengers in the vehicle and people in the street or damage public properties, such as transportation infrastructure [8]. When investigating an Uber self-driving crash, the National Transportation Safety Board (NTSB) [100] found that the accident was caused by internal components of the SDV system when the AI module failed to detect a victim. The SDV system was implemented to give a human driver control of the vehicle in unmanaged areas. However, the driver was distracted and did not react at the appropriate time.

Traditionally, automotive designers have built their safety strategy upon the concept that the human driver is essentially responsible for safety [81]. According to International Organization for Standardization (ISO) 26262 (Road Vehicles – Functional Safety) [72], the human driver is responsible for the safety at system level. In the same context, ISO has been lately, released a safety standard called ISO 21448 (Road Vehicles – Safety of the Intended Functionality), which aims to cover the safety hazards of a system with no component failure, such as when a human driver does not react appropriately when the SDV issues a request for intervention [73].

Human–machine interaction is a key issue in the design of intelligent systems such as SDV systems, especially if the interactions are related to the safety of the systems. According to the automation levels standard of the International Society of Automotive Engineers (SAE) [101], the design of SDVs is organised around human and machine interactions in which the human driver may play a fallback role to take control of the vehicle in hazardous events. Therefore, the SDV system is also responsible for engaging with humans – the people inside the vehicle – in the dynamic driving tasks. The interactions between the human driver and SDV system are considered to be critical tasks in terms of operating SDVs.

In this thesis, we undertake an analysis of the behaviours of an SDV system using both informal and formal methods. The integration of these methods could provide a solution to examine complex systems. Informal methods play a pivotal role by systematically analysing the system to identify critical requirements. However, because these approaches often lack formal aspects, formal methods are incorporated to improve the quality of the analysis. This integration ensures that potentially dangerous events are mitigated at the design level.

The following sections describe the challenges associated with the development and assessment of SDVs (Section 1.1), the research aims and questions (Section 1.2), the research plan (Section 1.3), the research contributions (Section 1.4) and the outline of this thesis (Section 1.5).

1.1 Problem Statement

This section discusses the research concerns related to SDV systems explored in this study. The primary challenge faced by SDV developers is the need to establish effective techniques for analysing and ensuring safety. This challenge encompasses various aspects, including legal regulations, system capabilities, and human driver involvement [32]. For example, notable SDV accidents highlight the limitations of system capabilities, such as Waymo’s SDV experiencing a crash due to its inability to detect obstacles during a lane change manoeuvre [33]. Conversely, Uber’s SDV faced criticism when

a pedestrian fatality occurred due to the human driver's delayed reaction when intervention was necessary [100].

In summary, three key challenges arise in developing SDVs from a safety engineering perspective: (1) the complexity of SDV systems; (2) interactions between SDV systems and human drivers; and (3) unclear safety constraints. These challenges are interrelated, and are summarised in the following subsections.

1.1.1 Complexity of an SDV System

The complexity of Self-driving Vehicles (SDVs) is determined by the automation levels and their features. According to the SAE [101], automation levels are classified from 0 to 5. Levels 1 to 3 (semi-automation) involve a human driver for driving tasks, while levels 4 and 5 (high-level automation) do not require human involvement. Although each level represents an increase in vehicle autonomy and a decrease in the required human input, challenges persist, particularly regarding safety, regulatory compliance, and technological capabilities [34]. For instance, human drivers may still need to intervene to ensure safety when SDVs require assistance.

1.1.2 Sharing Mechanisms of an SDV Control

The collaboration between the human driver and the SDV system in handling dynamic driving tasks varies depending on the level of automation. Hence, ensuring a seamless transition of vehicle control between them is essential. The roles of both parties may evolve based on the automation level. Initially, the human driver may play a passive role in supervising the SDV system's operation. However, if the SDV system fails to operate safely, the autonomous controller may prompt the human driver to take an active oversight role [16]. The necessity for investigating active oversight was underscored by the 2018 Uber accident [100], where the human driver did not promptly respond when the SDV requested intervention.

1.1.3 Insight into System Behaviours During Dynamic Driving Tasks

Lastly, developers of SDVs face a significant challenge in understanding safety constraints and requirements across various scenarios and potential faults [79]. Solving this challenge requires a thorough comprehension of system behaviours during dynamic driving tasks, crucial for ensuring the safe operation of SDV systems and enhancing their reliability and security. Therefore, it is crucial to identify the precise requirements and assumptions related to system capabilities, such as obstacle detection and intervention requests. This challenge is evident in incidents like Waymo's SDV

accident [33], where limitations in obstacle detection led to a collision, and the Uber accident, where a delayed human response highlighted the importance of timely intervention when the SDV requires human driver intervention [100].

1.2 Research Aims and Questions

This section outlines the research aims and questions that are derived from the previously summarised research problems.

The main aim of this thesis is to develop a methodology to analyse the dynamic behaviours of SDV systems, particularly when an SDV system requires human intervention.

The key objective is to augment safety in autonomous vehicles by emphasising the roles of both human drivers and the SDV system during dynamic driving tasks. Second, our methodology is based on a combination of informal and formal methods. The application of informal methods is to identify the safety requirements that the developers of the SDV system should consider during development of the autopilot software. Conversely, the formal method can verify the safe transition from an SDV system to a human driver by proving the consistency of the safety requirements derived from the informal method. The third objective is to expand a methodology for ensuring system safety against identified hazardous driving events at various abstraction levels. This strategy aims to systematically identify a set of safety requirements to guide the development of a formal model, from the high-level abstraction view to a more detailed concrete view. Last, the fourth objective is to employ the developed methodology on cases of various sizes and complexities, ideally those involving SDV systems with multiple functionalities.

Specifically, in this study we mainly address the following Research Questions (RQs):

- **RQ1:** What are the critical features in SDVs that define the roles of human drivers and autonomous systems in ensuring safety during autonomous operation?
- **RQ2:** How can an analytical framework be designed to systematically highlight the roles of both a human driver and an SDV system during autonomous operation?
- **RQ3:** How can a methodology be developed to analyse the complexity of system safety during autonomous operation?
- **RQ4:** To what extent does the proposed methodology demonstrate its utility when applied to a case study of varying sizes and complexity, especially those involving interaction between human drivers and an SDV system?

Each research question is then further detailed in its own subsection.

1.2.1 Research Question 1

The research question, [RQ1](#), prompts an exploration of the intricate interactions between human drivers and autonomous systems within SDVs during autonomous operation. Two subquestions guide the investigation into relevant literature, enhancing the depth of analysis as follows:

- **RQ1.1:** What are the key automation features in SDV systems to define the responsibilities of both the human driver and the SDV during autonomous operation?
- **RQ1.2:** What tools and methods are currently identified in the literature for analysing safety in SDV systems?

Firstly, an exploration into the key automation features within SDV systems is essential. The derived research question, [RQ1.1](#), prompts an examination of the specific functionalities and features employed in SDVs that outline the responsibilities between human drivers and autonomous systems during autonomous operation. Understanding these features is key in comprehending the division of tasks and decision-making processes within the dynamic behaviours of SDV systems.

Secondly, the investigation encompasses an exploration of tools and methodologies available in the literature for analysing safety in SDV systems. The derived research question, [RQ1.2](#), seeks to identify and evaluate the various approaches to assess the safety performance of SDVs. It includes an exploration of how informal methods of hazard identification contribute to determining safety requirements, and the role of formal techniques in effectively documenting and defining these requirements.

1.2.2 Research Question 2

The research question, [RQ2](#), aims to elucidate how specific techniques contribute to systematically ensuring the dynamic behaviours of the SDV system. Key objectives include identifying the specific functionalities and features that shape and emphasise the interactions between human drivers and SDVs, as well as outlining processes for comprehensive analysis of system component interactions. Additionally, it provides a structured approach to understanding the automation aspects between an SDV and a human driver during autonomous operation.

1.2.3 Research Question 3

The research question, [RQ3](#), prompts the development of methodology to analyse the safety of SDV systems. Two subquestions guide this development:

- **RQ3.1:** How can a methodology be developed to systematically analyse the complexity of system safety from high-abstraction behaviours down to more detailed behaviours?
- **RQ3.2:** How can a methodology be developed to formally analyse human responses when the SDV may issue a request to intervene?

[RQ3.1](#) aims to identify the development of a methodology for analysing the complexity of system safety during autonomous operation in SDVs. This methodology studies critical features arising from system component interactions between human drivers and SDV systems, starting from a high abstraction level and descending to concrete detailed features. The objective is to establish a structured approach for comprehensively assessing safety within the SDV system.

On the other hand, [RQ3.2](#) focuses on developing a methodology for formally analysing human responses when SDVs request intervention. This involves capturing how human drivers react when prompted to take control of the SDV. The methodology seeks to formalise and systematise these responses, providing insights into the dynamics of human interaction with SDV systems during intervention scenarios. Addressing these two research questions can establish a comprehensive framework for ensuring safety and understanding human responses in SDV systems.

1.2.4 Research Question 4

[RQ4](#) aims to evaluate the effectiveness of the proposed methodology through case studies of varying sizes and complexities, particularly those involving interactions between human drivers and SDV systems. These case studies validate the findings from previous research questions and investigate how the methodology performs in real-world scenarios. They range from high-level abstract behaviours to more concrete actions where human drivers intervene in SDV systems. By examining the methodology's application across different contexts and complexities, its utility can be thoroughly assessed. This validation process ensures the methodology's robustness and modularity, addressing the multifaceted challenges inherent in human-SDV interaction during autonomous operation.

1.3 Research Plan

This thesis addresses the proposed research questions by exploring current methodologies within the existing literature. Specifically, it focuses on highlighting the autonomous features of SDVs and ensuring their safety. Following this, a suggested methodology is proposed that seeks to combine existing methods with novel ideas, aiming to address the previous research questions. Figure 1.1 illustrates three main activities involved in this study:

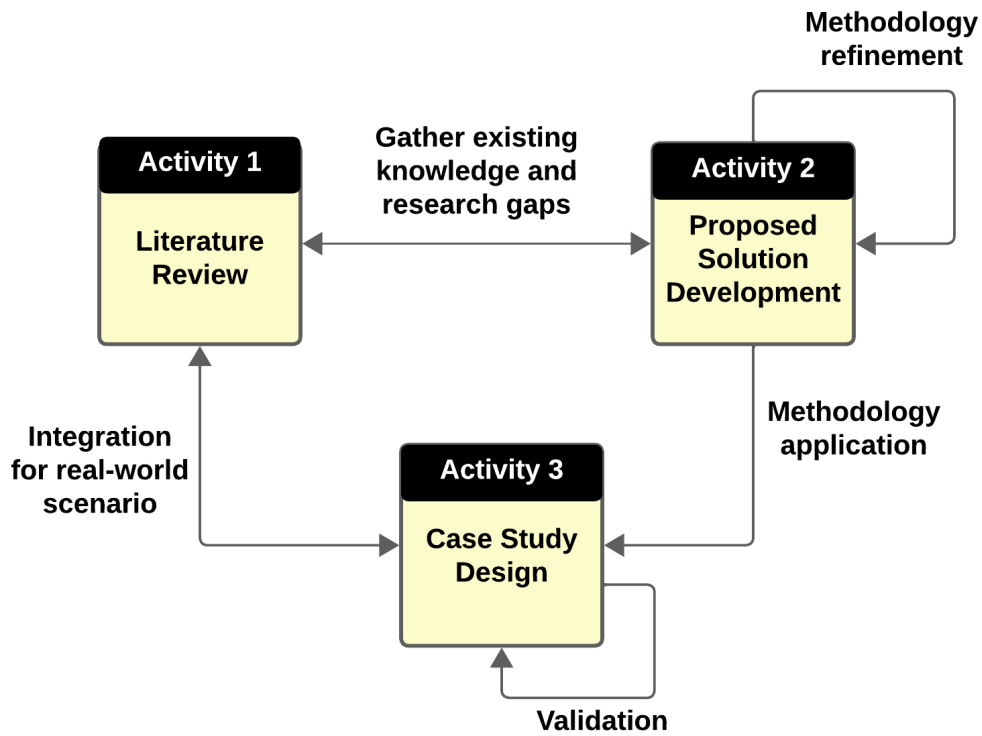


FIGURE 1.1: Research plan activities

1. **Literature review:** Examines existing research on SDV autonomy, focusing on safety measures and human intervention potential, to identify gaps and establish foundational knowledge.
2. **Solution development:** Iteratively develops a methodology for examining autonomy in SDV systems, considering scenarios requiring human intervention.
3. **Case study design:** Constructs real-world scenarios based on literature insights to validate the methodology, particularly in critical situations requiring human intervention, contributing to the advancement of SDV technology.

Overall, the research plan advances knowledge in the field of autonomy in SDV systems, facilitating a thorough investigation of the research problem and contributing to the advancement of SDV technology.

1.4 Contribution

The major contributions of this work can be summarised as follows:

1. The development of a Rigorous Analysis Template (RAT) framework that emphasises the automation aspects between an SDV and a human driver. This is especially crucial when the SDV engages the human driver as a fallback during hazardous scenarios.
2. The development of a step-by-step methodology called the Rigorous Analysis Template Process (RATP). This methodology extends the RAT, allowing for systematic development of a formal template to aid in analysing the behaviours of the SDV system at various levels of abstraction.
3. An intervention timing strategy, arising from the application of the RATP, has been developed. This strategy investigates the critical properties associated with moments when SDV systems might need to transfer control back to human drivers.
4. The validation and development of RATP were accomplished by a case study involving various functionalities of SDV systems.

Parts of this thesis have been already published in the following papers:

- Alotaibi, F., 2020. Improving trustworthiness of self-driving systems. In *Rigorous State-Based Methods: Proceedings of 7th International Conference, ABZ 2020, Ulm, Germany, May 2020*, 7 (pp. 405-408): Springer International Publishing [11]. This paper discusses the challenges, requirements and preliminary concepts for analysing complex systems, particularly SDV systems.
- Alotaibi, F., Hoang, T.S. and Butler, M., 2022. High-level rigorous template for analysing safety properties of self-driving vehicle systems. In *IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, June (pp. 1643-1648): IEEE [12]. In this paper, we present a methodology called the Rigorous Analysis Template (RAT) approach. It focuses on highlighting the automation interactions that occur between the SDV and the human driver.
- Alotaibi, F., Hoang, T.S. and Butler, M., 2023. A rigorous iterative analysis approach for capturing the safety requirements of self-driving vehicle systems. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1697-1702), June: IEEE [13]. In this paper, we present a methodology called the Rigorous Analysis Template Process (RATP) approach. It extends the RAT, allowing for the systematic development of a formal template that aids in analysing the behaviours of the SDV system at various levels of abstraction.

Additionally, the author of this thesis has contributed to the following publication:

- Asieh Salehi Fathabadi, Colin Snook, Dana Dghaym, Thai Son Hoang, Fahad Alotaibi and Michael Butler, 2023. Designing critical systems using hierarchical STPA and Event-B. In *Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May -June* (pp. 220–237) [116]. I contributed in the particular evaluation of the systematic analysis steps, focusing on their execution within the suggested analysis framework.

1.5 Outline of Thesis

This thesis is organised into 10 chapters. Chapter 2 provides the background information on SDV systems. Chapter 3 gives an overview of safety analysis methodologies, based on a variety of informal and formal analysis methods. Chapter 4 details a case study that involves our approach in the context of validation. Chapter 5 presents the RAT that highlights the automation aspects between an SDV and a human driver. Chapter 6 describes the RATP approach to gradually investigate the behaviours of the SDV system, from a high-level abstraction view to a more detailed concrete level. Chapters 7 and 8 validate the RATP approach through a case study. Chapter 9 presents how driver reactions can be involved in the RATP approach. Finally, Chapter 10 concludes this thesis.

Chapter 2

Overview: Self-Driving Vehicle Systems

In this chapter, the objective is to determine the key aspects of automation in Self-Driving Vehicle (SDV) systems. The Research Question (RQ) associated with this objective is [RQ1.1](#), and is formulated as follows:

RQ1.1: *What are the key automation features in SDV systems to define the responsibilities of both the human driver and the SDV during autonomous operation?*

Section [2.1](#) briefly discusses the design principles for autonomy, with a particular emphasis on human–machine interaction. Section [2.2](#) outlines the taxonomy requirements of SDVs, classifying autonomy into levels. Section [2.3](#) describes the critical system modules involved in developing SDV systems. Section [2.4](#) discusses automation features between human drivers and SDV systems to address [RQ1.1](#). Section [2.5](#) concludes this chapter.

Parts of this chapter were published in ABZ 2020 [[11](#)], particularly the discussion on automation in Section [2.4](#).

2.1 SDV Human and Machine Interactions

Autonomy has advanced cooperation between humans and machines. With the rapid development of advanced tools and techniques, the interactions between humans and autonomous machines are today becoming increasingly complex. Although automated machines are expected to work without human intervention, determining what rules and instructions are embedded into the automated system is challenging [[79](#)]. The main aim of an automated system is to reduce human operation by creating a fully

autonomous system [14]. Therefore, it is essential to understand what kind of operations can be applied by humans to understand how such a human behaviour can be concretely interpreted by a machine.

Humans can obtain knowledge from the mental model [134], a concept proposed by psychologist Craik [37], who states that the mental model constitutes ‘*cognitive-emotional representations*’ of the environment, objects and the relationships among multiple objects in the environment. Cognitive psychology scientists [57, 134] have also indicated that mental models are dynamic and can be achieved through three activities. *In the first activity*, mental models work in the memory (mind) and allow people to simulate potential actions and their consequences. *In the second activity*, the mental models generate assumptions or causal understandings of how system functions may work. *In the last activity*, mental models change over time due to experiences (e.g. systems are capable of continuous learning). These psychology concepts can be adopted into the technology field, especially when they shift from the theoretical to the practical level.

Considering the idea of SDVs from the perspective of the mental model, the machine (autonomous controller) would not only replace the human driver in operating the driving tasks but also cover the activities of a mental model in its design. Although the taxonomy approaches of SDVs attempt to answer the question of how mental model activities can be interpreted by SDVs, which is discussed in greater depth in Section 2.2 (Taxonomy Requirements), three design principles of autonomy were found in the literature of automotive field. These are explained in the following points.

1. **Semi-automation:** The main design principle is known as ‘*partial automation*’ or ‘*semi-automation*’, where humans and machines engage together to achieve a system goal [34]. Semi-automation architecture is often divided into levels that represent the sharing of the system’s features between the human operator and autonomous controller.
2. **Supervision:** Automation shifts functional requirements from the human operator to technical systems. However, the human operator may play a new role during the lifecycle of the automated system, that known as ‘*supervisory control*’ [96]. The idea of human supervision was introduced by Bainbridge et al. [16], and can be summarised as follows: 1) *expected and normal operations are performed autonomously*; and 2) *critical and unexpected situations are operated manually*. While the system operates autonomously to prevent human errors, there may be instances where it requires a human operator to be actively involved as a fallback in critical situations.
3. **A fallback option:** One of the most crucial aspects of designing autonomous systems is to provide sufficient information to humans [79]. This principle is especially important if the system assumptions involve a human fallback component

to handle emergency situations or malfunctions. For this reason the automated system must send useful information to humans so that they understand the current situation of the system, and the system must take into consideration their reaction time before it can turn the system to manual control.

These three principles are developed mainly to improve the safety of the entire system. In the literature, engagement between humans and machines is widely known as the Human-In-The-Loop System (HITLS) [52]. A HITLS can add advanced qualifications to the autonomous system model to make it the top '*safety pick*'. According to the Insurance Institute for Highway Safety [115], an SDV can become this when it is integrated with technology to prevent crashes. One of these integrated systems is an HITLS that aims to prevent collisions by creating a digital collaboration space between human operators and machines [21].

Despite the many studies on HITLS, such as on enhancing the user experience [115], few consider how to incorporate HITLS technology into autonomous systems even though this is an essential component for improving levels of safety and reliability in SDVs.

A technical report from Massachusetts Institute of Technology (MIT) [53] indicates that the human fallback component of HITLS is an important factor in all aspects of a safe driving mode. MIT's researchers also believe that the answer to the safety problems in driving manoeuvres can be solved through a variety of sensors, which focus on the eyes, head and hands of the human driver to ensure the responsiveness of the human fallback component. However, they point out that there are several challenges to implementing a HITLS within an SDV, including the following:

- The inconsistency of human behaviours to handle all forms of social distractions or problem-solving concerning in-road objects such as vehicles, pedestrians and cyclists.
- The variety of human driving styles and human abilities to use automation from either the user experience or usability perspective.
- The unknown limitations of the detection system and unreliable sensing devices to ensure the responsiveness of the human fallback component.
- The difficulty of identifying when a system failure might occur and the ability of the HITLS to notify a fallback driver about how to handle unpredictable driving scenarios.
- The existence of machine and human errors.

The concept of HITLS has been adopted by autonomous vehicle companies, such as Tesla and Volvo. According to both these companies [102, 131], their autopilot software

uses a Driver Monitoring System (DMS) to ensure the responsiveness of the human fallback component. The DMS can verify the awareness level of a human driver via two features. Volvo's autopilot software [131] is used as an active monitoring feature that requires a human driver to keep their hands on the steering wheel to operate the auto-driving mode. By contrast, Tesla's autopilot software models 3 and Y [56, 102] employ a sensitive monitoring feature that uses the in-car camera to detect and monitor human drivers when the advanced driver assistance system is in operation. For instance, one of the integrated systems for ensuring the intervention of a human driver is a Facial Recognition Algorithm (FRA) [140]. FRA is a type of sensitive monitoring system that can scan a human face to detect facial landmarks and recognise facial variable states, namely whether the eyes are open or closes, based on monitoring changes in a sequence of face images. However, Zhang and Yu [140] suggest that a single sensor camera might not be enough for a face detection system. They found that developing a complex Artificial Intelligence (AI) algorithm for identifying and classifying the massive amount of data provided by a sensor system would still only detect 61.29% of driver faults. While this is clearly not 100%, the FRA can still decrease the risk of vehicle crashes and substantially prevent financial and personal losses.

To integrate with the HITLS, SDVs require an additional layer for monitoring the human fallback component. In a case study [52], Lex Fridman found that SDVs can be designed around the human driver and, further, noted that the FRA can detect that a driver is distracted after approximately three seconds of not looking at the road. Therefore, he suggests that a new layer between a human driver and the SDVs is added to ensure that a human driver can take control of the vehicle when critical situations occur.

2.2 Taxonomy Requirements

The taxonomy requirements of SDVs are built on the idea that the human driver (human fallback component) and the SDV system (autonomous controller) can work together to achieve the driving tasks. Several well-known concepts are used to identify the differences between operating a vehicle by the human fallback component and by the autonomous controller. These concepts are explained as follows:

- **Operational Design Domain (ODD):** This determines under which conditions the autonomous controller performs a specific driving task [124]. These conditions are categorised into the following two groups: 1) environmental conditions, such as weather; and 2) operational conditions, such as the speed limit.
- **SDV system:** This indicates the hardware and software components that combine together to perform the DDT on either a limited or unlimited ODD [101, 34].

- **Dynamic Driving Task (DDT):** This is defined as the operational and tactical behaviours of the autonomous controller that are required to operate a vehicle in road traffic [101].

In the operation of SDVs, the Operational Design Domain (ODD) plays a crucial role. The ODD specifically outlines the conditions under which the autonomous controller can operate safely. These conditions include various environmental and operational factors, as well as geographical limitations [106, p. 8]. For instance, the ODD for certain SDVs may be confined to urban areas, function only during daylight, and require clear weather conditions. By defining these parameters, the ODD establishes critical boundaries, ensuring that the SDV makes dependable decisions for safe and efficient operation.

The SDV system is a complex combination of sensors, processors, and actuators. This system is designed to detect the driving environment using advanced technologies such as LiDAR, radar, cameras, and GPS [70]. Once it collects this environmental data, the system processes it to make decisions in real-time, subsequently controlling the vehicle's movements. In more detail, the autopilot software within the SDV system interprets the data from the sensors and anticipates the behaviour of other road users. This capability enables the SDV to respond suitably to various scenarios, encompassing navigation, emergency management, and adaptation to evolving conditions. The SDV system is in a state of continuous enhancement, integrating AI and advanced algorithms to progressively refine its decision-making capabilities over time.

The Dynamic Driving Task (DDT) model plays a crucial role in demonstrating the taxonomy requirements of SDV systems. The DDT is developed from traditional driving tasks or what operations are applied by humans in driving manoeuvres [95]. Both DDT and traditional driving tasks aim to modify vehicle control variables and perform driving tasks [34]. Michon [95] observes that the DDT encompasses the following three activities:

1. **Operational behaviours:** These allow SDV systems to make important observations about the actual driving task. These observations help to determine the driving decision. However, due to the variety of dynamic driving scenarios, identifying the actual decision for a specific vehicle manoeuvre is difficult. Therefore, the concept of Object and Event Detection and Response (OEDR) has been developed to represent the dynamism of driving tasks. According to an automotive manufacturer consortium (CAMP AVR) [34], the OEDR involves the perception and detection of any circumstances relevant to the actual driving task.
2. **Strategic behaviours:** The strategic behaviours refer to the planning strategy of the SDV systems, which aims to determine the driving decision and accomplish

a system goal. These behaviours include destination and route plans [95]. The destination plan contains the start and final location of a mission. By contrast, the route plans are updated during the mission due to the dynamics of the environment or the variety of driving scenarios. However, strategies for updating plans rely mainly on the operational behaviours of observing the environment and formulating the plan's strategy.

3. **Practical behaviours:** These are related to the functionalities that apply to physical/electronic control variables [34]. These functions manipulate the control variables of the vehicle to apply a specific driving decision. The two main groups of control variables are as follows: 1) *lateral variables*, which change the vehicle's position in a horizontal manner by using steer and orientation features; and 2) *longitudinal variables*, which move the vehicle vertically by modifying various features, such as speed, acceleration/deceleration and braking.

The activities of the DDT may explain how the autonomous controller makes driving decisions and controls the automated vehicle. As mentioned earlier, in the context of this chapter the autonomous controller is expected to replace the human driver. This assumption leads us back to the concept of mental models and how the human driver would perform traditional driving tasks. Therefore, we created the relationship diagram in Figure 2.1 to compare the activities of the DDT and human mental model.

In the first step, the human driver relies on his/her eyes to monitor the environment and identify the driving actions [57], while the autonomous controller attempts to do the same using the Object and Event Detection and Response (OEDR) system [34]. In the next step, before applying a driving action the human driver selects an action that supports their goal, while the autonomous controller aims to choose a safe action that supports the goal of a system. Finally, the last step for the human driver is to apply a driving action, while the autonomous controller specifies the control vehicle variables to apply a selected action.

DDT differs from traditional human driving tasks, and the changes in the driver's role can become the basis for establishing the taxonomy requirements of the driving automation system. According to the Society of Automotive Engineers (SAE) [101], gathering the taxonomy requirements of DDT is related to the human fallback component and can be explained in the following two main factors:

- **Driver attention requirements:** This relates to a human fallback driver who sits behind the steering wheel inside the SDV. The taxonomy requirements involve the required awareness level of the human fallback driver when the SDV systems are operated. For instance, an SDV system might not work until a human fallback component is aware of the performance of the DDTs.

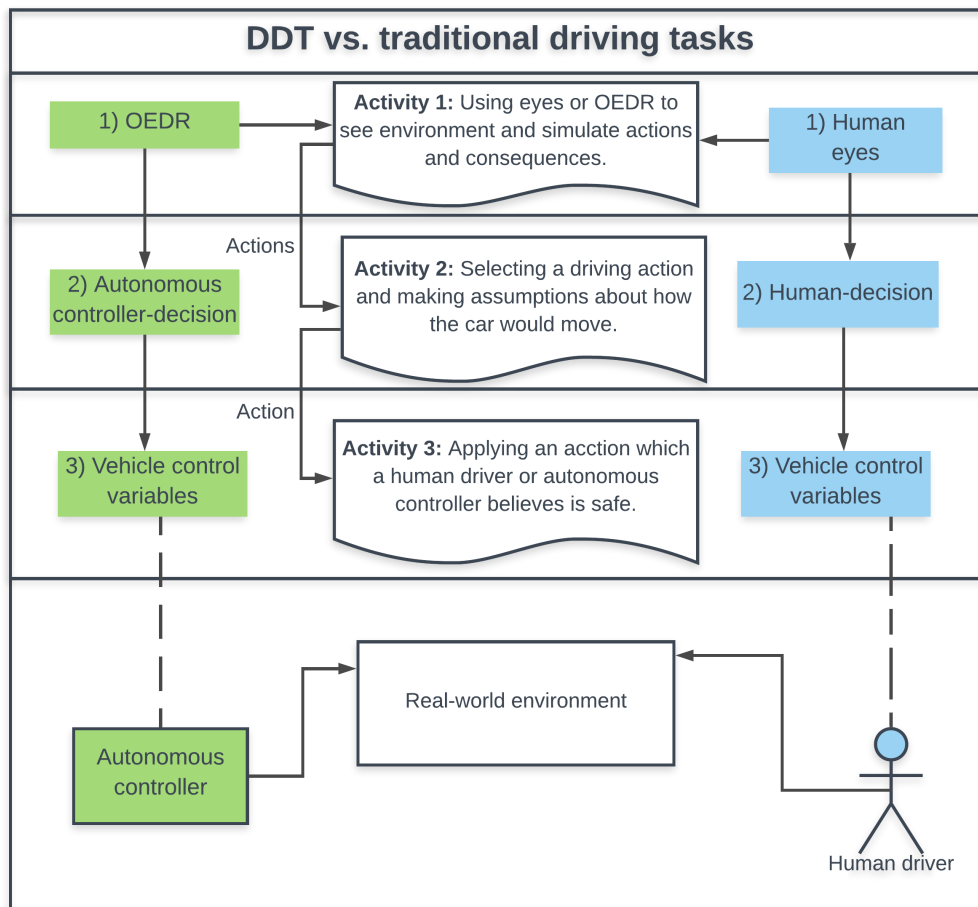


FIGURE 2.1: DDT vs traditional driving tasks

- **Driver action requirements:** This relates to the actions that are intended to be applied by the human fallback component. For instance, the human fallback driver might control the vehicle's speed, while the autonomous controller modifies the vehicle's steering angle. Therefore, the design of the autonomous operations affects the actions that the human fallback driver might perform in the SDV systems.

The tasks that might be performed by an autonomous controller [101, 14, 34] to accomplish the DDT include the following:

- **Lateral control – steering:** This refers to the task of steering and moving laterally on the road, such as turning right, left, going straight or taking a bend.
- **Longitudinal control – braking, accelerating:** These tasks relate to the position or velocity of the SDV along the roadway, modified through actions such as braking or acceleration.

- **OEDR task:** This demonstrates the ability of a system to detect and respond to any environmental events that may affect the performance of the DDT. In general, the OEDR works in conjunction with the specific ODD.
- **Planning task:** Although the immediate response is already part of OEDR, planning tasks include the goal of a system to be accomplished via long- and short-term plans. The planning task also involves the selection of automation functions (manoeuvres), such as lane changes and intersection crossings, as part of the planning strategies.

Considering the way the SDV system operates within its ODD to perform DDTs, well-known approaches have been proposed to classify autonomy into levels. In the next subsection, the automation levels and their approaches are outlined.

2.2.1 Automation Levels

In exploring the concept of autonomy, recognising its diverse classifications is crucial. This section highlights two distinct approaches: traditional methodologies (Subsection 2.2.1.1) and specific classifications for SDV systems (Subsection 2.2.1.2). Each approach offers a unique perspective, enhancing the overall understanding of how autonomy is classified and applied in various contexts.

2.2.1.1 Traditional Classifications of Autonomy

The traditional approaches to defining automation levels are mainly extended from the popular model of Sheridan and Verplank [119], which defines the responsibilities of humans and machines at various levels. At low levels (1–5) the computer offers assistance but the human makes decisions; at higher levels (7–10) the computer works autonomously and ignores human interventions.

Based on Sheridan and Verplank’s approach, Parasuraman et al. [105] developed the pipeline model of human decision-making. This aims to simplify autonomous functions into four stages: *sensory processing*; *perception*; *decision-making*; and *response selection*. As the human operator/driver may ignore the action proposed by the automated machine, Vagia and Rødseth [129] extend the four-stage model of Parasuraman et al. [105] with a new step, *execution*. These stages can be clarified as follows:

1. **Sensory processing:** This refers to the sensing and registration of input data. For instance, the SDV may use a camera to observe its environment.

2. **Perception:** This refers to the algorithms that can be applied to incoming data. For instance, the SDV may process an incoming image from a camera to identify the lane lines in its environment.
3. **Decision making:** This refers to the selection of a particular action if specific conditions exist. For instance, the SDV may attempt to change its position to be in the middle of lane lines if the lane lines have already been detected in stage 2.
4. **Response selection:** This refers to the action implementation of the selection made in stage 3. For instance, the SDV may actuate its selected action to continue in the middle of lane lines.
5. **Execution:** This function refers to the responsible actuator, whether a human or a machine. For instance, the SDV may actuate its selected action manually or autonomously.

A summary of Sheridan and Verplank's approach with the various stages of autonomous functions is shown in Figure 2.2 to indicate the possible human interaction with automation at various levels.

2.2.1.2 Classifications of Autonomy for SDV systems

There are two well-known classifications for SDV systems in the literature. Firstly, according to the National Highway Traffic Safety Administration (NHTSA) [21], the levels of autonomy demonstrated are based on the level of autonomous function versus human control. The main goal of the NHTSA's approach is to generate a common terminology across the fields of the autonomous vehicle industry [21]. In the second approach, proposed by the SAE [101], automation levels are identified by the following two factors: 1) degree of interactions between the human fallback driver and the SDV systems; and 2) the pre-attentive level of the human fallback driver required by the autonomous vehicle. A summary of the automation levels provided by the SAE is outlined in Table 2.1. For instance, automation levels 1 to 3 rely on a human driver to handle hazardous events that may happen in DDTs, while at automation levels 4 and 5 the SDVs are able to handle hazardous events in limited or unlimited environmental conditions. These levels are defined as follows:

1. **Level 0 – No Automation:** The human driver is responsible for all aspects of DDT.
2. **Level 1 – Driver Assistance:** The SDV system can assist with limited functions such as adaptive cruise control; however, the human driver is still responsible for most aspects of DDT.

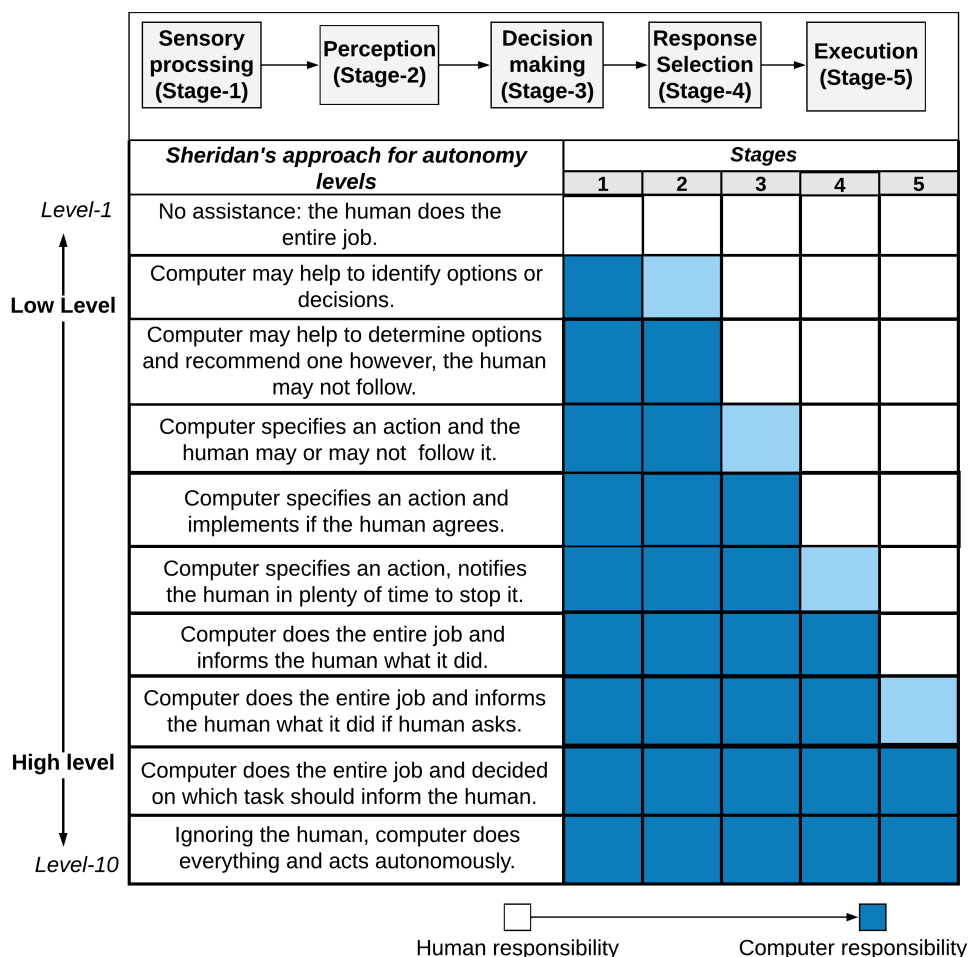


FIGURE 2.2: Types of human interaction with autonomy, based on stage of autonomous functions and Sheridan and Verplank’s taxonomy approach

3. **Level 2 – Partial Automation:** The SDV system can handle some DDTs, such as steering, accelerating and braking, but the driver must still be ready to take control at any time.
4. **Level 3 – Conditional Automation:** The SDV system can handle most DDTs; however, the human driver must still be ready to take control when the system requests it.
5. **Level 4 – High Automation:** The SDV system can handle all DDTs in certain ODDs, such as on a highway, and the human driver is not required to take control.
6. **Level 5 – Full Automation:** The SDV system can handle all DDTs in all ODDs, and no human driver is required.

Moreover, the NHTSA [21] and SAE [101] tackle the issue of taxonomy requirements for autonomy from different perspectives, as shown in Figure 2.3. The NHTSA focuses on the number of autonomous functions executed at each level, while the SAE highlights

the responsibility of the human driver during operation of the SDV systems. However, both approaches are limited by the OEDR system's ability to handle critical situations or unmanaged conditions, namely scenarios not covered by the ODD.

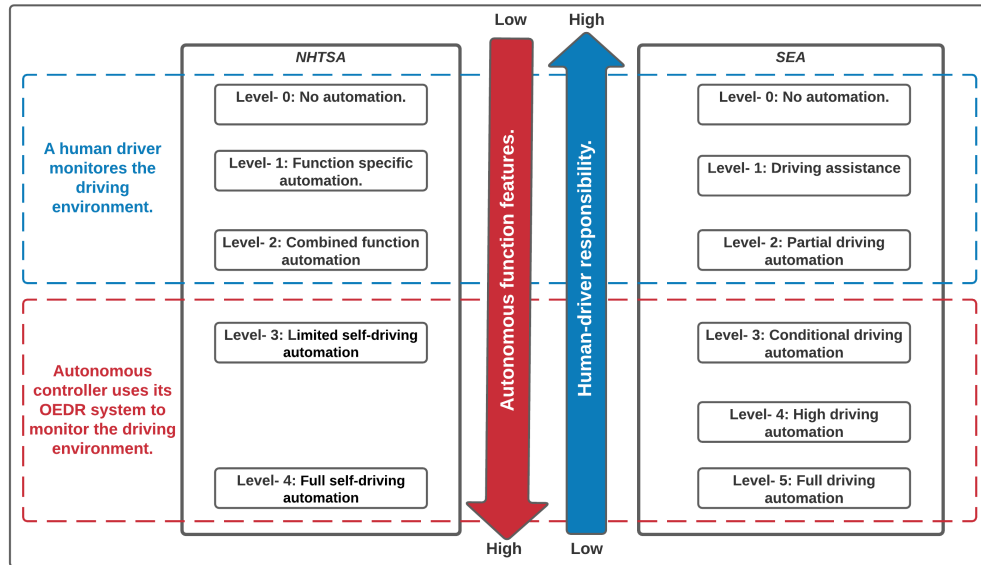


FIGURE 2.3: Comparison of NHTSA and SAE approaches to classification of autonomy levels

The SAE approach is similar to the early approach of Sheridan and Verplank [119]. Nevertheless, Sheridan and Verplank's approach involves more information about how humans and machines can work together to achieve a system goal, such as the possibility that the machine informs the human before applying the autonomous functions.

However, while notifications sent from machines to humans might be useful to verify the driving decision during the run time, the property of time for receiving and responding to a notification message is vital. As Badue et al. point out [14], SDV systems work in decision-horizon time, which is a limited time and requires a quick response from a human fallback component.

2.3 Generic System Architecture

Various studies, including the DARPA Challenges [128], Zong et al. [143] and Badue et al. [14], have shown that the SDV systems require the following three modules to perform the DDT: *perception*; *decision-making*; and *control* modules. These are summarised below; for more information, refer to [14, 128, 143].

- **Perception module:** This is intended to perform the OEDR task. It estimates, for the internal components of SDV systems, the vehicle's state (velocity and location). It is also integrated with a variety of sensors whose purpose is to create

an internal representation of the environment and its relevant objects, such as street views and static/dynamic obstacles. The perception module involves multiple subsystems that are mainly developed to achieve the following two tasks: 1) to navigate a vehicle without any collisions in its environment; and 2) estimate the vehicle's ability to create a plan to accomplish the system's goal using the decision-making module.

- **Decision-Making module:** This is to accomplish the planning task. The module is responsible for achieving the system's goal. For instance, the goal might be to navigate the automated vehicle from its current location to the destination location, or to keep the automated vehicle between the lane lines. These kinds of goals cannot be accomplished without the help of a perception module. Therefore, the decision-making module relies primarily on the outputs of the perception module to observe its target environment. However, unlike the perception module, as well as the vehicle's velocity and orientation the decision-making module produces a target path for a vehicle to follow.
- **Control module:** This is responsible for accomplishing the lateral and longitudinal tasks that actuate the vehicle in its environment. It receives the outputs of the decision-making module to specify the modifications to low-level physical/electronic vehicle components. For instance, the vehicle's velocity becomes a signal to the accelerator pedal, while the vehicle's orientation guides the change in steering wheel control.

The internal modules of SDVs work in a continuous iterative manner, based on the time spent observing the environment, proposing a driving decision and applying that proposed driving decision to its environment [14].

Based on the taxonomy requirements (SAE) [101], a human fallback component is also a part of an SDV system. Although high automation (Levels 4 and 5) supposes that SDVs work autonomously, without human intervention, the human fallback component plays an important role in lower automation levels (Levels 1–3). In assistance autopilot software (e.g. Tesla [102, 71], General Motors (GM) [98] and Volvo [131]), the HITLS integrates with the internal components of SDVs to become a fallback option when a system fails to operate as expected. Therefore, we created an abstract component diagram to show the main components of the SDV systems, as in Figure 2.4.

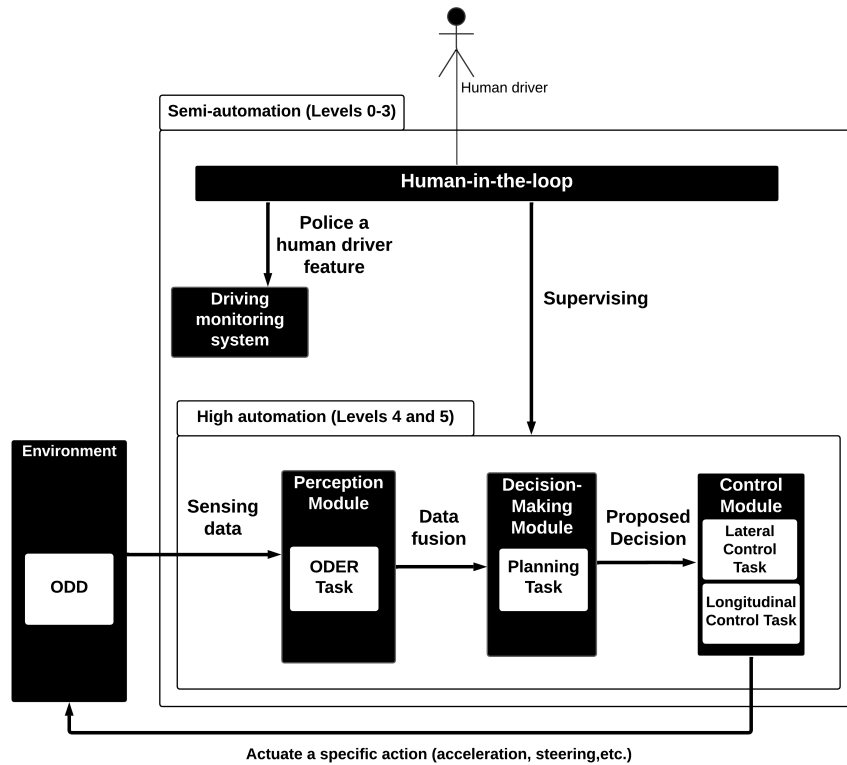


FIGURE 2.4: Abstract components of SDVs

2.4 Discussion

The main objective of this chapter is to review existing literature to identify the automation aspects of SDV systems relevant for addressing RQ1.1. The taxonomy approaches for SDV systems, such as the approaches proposed by the NHTSA [21] and the SAE [101], seem to be universal definitions that give only a description of the DDTs and their dynamic features, such as ODD, OEDR and fallback driver. These features mainly demonstrate the operational tasks that are intended to be performed by the autonomous controller.

However, autonomy creates a cooperation channel between humans and machines. The role of the human driver has been misleading, in taxonomy approaches. The advanced human interaction features have been previously specified and explained by Sheridan and Verplank [119] in a traditional approach to defining autonomy levels. For instance, a machine might sound an alarm and inform the human driver of its decision. The SDV systems are also responsible for monitoring the awareness level of the human driver to ensure the responsiveness of the human fallback component. Therefore, the SDV may also need to handle features such as alarms, notification messages and human intervention requests.

As outlined above, the abstract representation of SDV modules can be organised into perception, decision-making and control modules. However, to ensure the awareness

level of the human fallback component, a HITLS may also be adopted. The HITLS would add an advanced engagement channel for preventing autonomous vehicle crashes. For instance, a DMS may be a part of the SDV systems to demonstrate the awareness level of a driver. Finally, there is a need to include human fallback features in the taxonomy approaches, especially if an SDV assumes that the human driver is to be a fallback option for managing a hazardous event.

2.5 Conclusion

This chapter aimed to provide a foundational understanding of SDVs, with a specific focus on the aspect of autonomy. A fundamental concept of autonomy in SDVs involves the dynamics of interaction between humans and machines. From a narrative review of the literature, three key design principles emerged that enhance these interactions: 1) *semi-automation*, 2) *supervision* and 3) *the human fallback component*, which are used to improve engagement in human and machine interactions. These design principles are widely known in the literature as the HITLS. Various safety methodologies and techniques that could potentially assist in the analysis of SDVs are reviewed in the next chapter.

TABLE 2.1: Summary of automation levels, adapted from [101]

Automation levels		DDT features			Environmental conditions
Level	Description	Sustained lateral and longitudinal	OEDR	Fallback	ODD
Level- 0: No automation	A human driver performs all the DDT features.	Human driver	Human driver	Human driver	Limited
Level- 1: Driving assistance	A system can control lateral or longitudinal features but not both, and a human driver performs the remainder of the DDT.	Human driver and system	Human driver	Human driver	Limited
Level- 2: Partial driving automation	A system can control both lateral and longitudinal features, and a human driver performs the remainder of the DDT.	System	Human driver	Human driver	Limited
Level- 3: Conditional driving automation	A system can control lateral, longitudinal, and OEDR features. A human driver becomes a fallback-ready user if a system sends a request to intervene.	System	System	Fallback-ready user (becomes a human driver if a system issues a request to intervene)	Limited
Level- 4: High driving automation	A system can perform all DDTs; however, a human driver is responsible for managing a system outside its ODD.	System	System	System	Limited
Level- 5: Full driving automation	A system can perform all DDT in any environmental conditions.	System	System	System	Unlimited

Chapter 3

Safety Analysis Methodologies

In this chapter the objective is to investigate both the informal and formal methods found in the literature that could be employed to analyse safety in Self-Driving Vehicle (SDV) systems. The Research Question (RQ) linked to this aim is [RQ1.2](#), and is formulated as follows:

RQ1.2: *What tools and methods are currently identified in the literature for analysing safety in SDV systems?*

Section [3.1](#) explores the informal hazard identification methods used for studying system safety. Section [3.2](#) investigates the safety standards and methodologies utilised by autonomous vehicle companies. Section [3.3](#) reviews formal methodologies suitable for addressing various tasks and properties of SDVs. Section [3.4](#) presents a research discussion to address [RQ1.2](#), and Section [3.5](#) provides a conclusion to this chapter.

3.1 Hazard Identification Methods

This section considers various hazard identification methods that are particularly relevant to safety-critical systems, such as SDV systems.

3.1.1 Hazard and Operability Analysis

Hazard and Operability Analysis (HAZOP) [[38](#)] has been developed to identify system hazards based on possible behaviour deviations at the system level. The deviations are determined by analysing all processes and sub-processes through a set of specified guidewords. The advantage of using guidewords is to explore the characteristics of the

system under examination to create a consistent model. HAZOP has been adopted in a wide range of domains, including the automotive field.

In SDV systems the use of HAZOP is mainly related to functional characteristics, where the focus of analysis is on interpreting the guidewords to identify hazardous behaviours [62]. Although the HAZOP method is applicable to the identification of hazardous behaviours, there are currently few guidelines on how to interpret the guidewords. For instance, Bagschik et al. [15] define the keywords for groups of SDVs' modules, such as perception, planning and control, to generate the possible malfunctions of each module (keywords and corresponding guidewords are proposed in Table 3.1). However, due to other, unknown guidewords for multiple driving scenarios, the discovered guidewords should be considered only as a general guideline [62].

Moreover, the HAZOP method can be used at the functional level. For instance, according to Becker et al. [18], HAZOP for Automated Lane Centring (ALC) is applied to each of the 24 ALC sub-functions based on seven guidewords, including '*loss of function*' and '*more than intended*', to create a set of hazardous malfunctions (113, for ALC). However, this usage of HAZOP is less focused on system level in terms of the interactions between multiple system components.

McKelvey [92] finds that the application of HAZOP faces the following six primary issues that might affect the validity of its results:

1. Lack of field experience, especially within the team undertaking HAZOP studies.
2. Lack of comprehensive documentation around the system under examination.
3. Top-management decisions that result in projects being developed too quickly or spending insufficient time/money on the HAZOP process.
4. Failures resulting from repeating processes and procedures in similar projects but requiring review during the new project.
5. Lack of technical documentation due to the complexity of the system's characteristics.
6. Human error due to insufficient investigation or an unqualified HAZOP team.

TABLE 3.1: Example of HAZOP keywords and guidewords, taken from [15]

Keyword	Guidewords
Perception	Too large, Too small, Non-existent, None
Planning	Physically not possible, Conflicting, Not relevant
Action	Too large, Too small, Wrong, Absent

3.1.2 Failure Mode and Effects Analysis

Failure Mode and Effects Analysis (FMEA) is a similar technique to HAZOP and aims to address the safety risks for systems on multiple scales. It is a bottom-up approach that was developed by the United States military in the 1940s [108].

FMEA is considered to provide a deeper analysis than other hazard identification methods in this domain of the literature, as it focuses on the low and high components of the system under examination. Although there are various methodologies for applying FMEA [25], the main idea is to evaluate and minimise safety risks by identifying failure modes in low-level processes [121]. The failure modes can be categorised by a specific Risk Priority Number (RPN). The identification of an RPN is calculated as the product function of a failure mode determined by the score of the following three properties [108, 125]:

1. **Severity of the effect of failure:** This indicates the possible risk to the overall operation of the system, from low-level to high-level components.
2. **Probability of occurrence:** This states the frequency number of failure modes.
3. **Ease of detection for each failure mode:** This determines the ability of the system to detect whether a failure mode has been implemented or continuously involved in the operation of the entire system.

FMEA has been adopted into the domain of SDV systems. For instance, Tokody et al. [125] used an FMEA worksheet to discuss the future impacts of SDVs. Discovering failure modes, such as *'Turning off human intervention'*, is linked to an appropriate RPN depending on scores related to severity, detectability and occurrence.

However, Gilchrist [55] argues that, *'though the model itself is of great use, the calculation of the RPN lacks a proper model as a base and thus is internally inconsistent and potentially misleading'*. In other words, the calculation of an RPN is inaccurate due to an unclear score of severity, detectability and occurrence, all of which may mislead any attempts to reduce risks.

Despite being used across multiple applications, FMEA has several limitations, resulting in the following criticisms [121, 122].

1. FMEA is only capable of identifying known faults based on the previous field experience of the development team.
2. The identification or removal of unwanted events is based on the exploration of all paths and combinations of a complex system; therefore it is time-consuming, and the exploration process itself may be misleading.

3. FMEA can be adopted only in the later stages of development, when the design of the system has been fixed.

3.1.3 Fault Tree Analysis

Fault Tree Analysis (FTA) is a reliability and hazard method that analyses the potential failures of a system. It is based on Boolean logic to estimate the failure probability of a single system component through a top-down exploration mechanism. The causes of failure are investigated through a graphical/logical tree that involves a combination of Boolean gates and events towards a single component of the system under examination [47].

FTA has been used in many domains, including SDV systems. For instance, Duran et al. [44] used the FTA method to analyse the impact of perception module failure on overall system success rates. The following is a summary of the FTA approach:

1. The assumption of severity levels is categorised as Levels 1 to 4, where Level 1 indicates '*no loss of anything*' and Level 4 indicates '*either high loss of human lives or environmental damage*'.
2. The probabilities of failure are calculated using the Bayesian belief network that shows conditional relationships between random variables in a directed graph.
3. The hazard identification process is based on the severity levels and probabilities of failure. Each hazard is classified using a qualitative method known as Preliminary Hazard Analysis (PHA). Like the RPN, the PHA is a qualitative method that calculates the product function of the severity and probability of failure to organise the risks into predefined arbitrary groups, such as *minor*, *major*, *critical* and *catastrophic*.

The implementation of FTA, however, has shown some limitations in terms of flexibility and applicability with regard to the domain of software-intensive systems [121, 17], including the following:

1. The outcome of hazard analysis is highly dependent on the experience of the engineers. Therefore, due to a lack of experience, the FTA method may be inapplicable to new technologies, such as SDV systems.
2. Due to the unknown hazard states leading to component failure, the identification of all possible trees may be inaccurate. For instance, in the top-level event (cause) of '*wrong lane lines detection*', FTAs need to consider all possible components that fail, either in isolation or in combination with all other components.

3.1.4 Systems Theoretic Process Analysis

Systems Theoretic Process Analysis (STPA) [84] is a hazard identification method that analyses a system involving a control structure. STPA was based on the Systems Theoretic Accident Model and Processes (STAMP) model, which is considered to be the theoretical background of STPA. Although there have been many extended versions of STPA, such as STPA-SafeSec [54] and STPA-Sec [138], the traditional STPA provided by Leveson [86, 84] is reviewed. Overviews of the original STAMP and STPA are given in the following subsections.

3.1.4.1 Theoretical Background of the STPA Method

STAMP is a set of assumptions about how accidents occur. An accident is defined as '*an unwanted and unplanned loss event*'. A loss may include human injury or death; however, it also involves events such as financial, information and equipment losses [86, p. 75]. According to Leveson [85], STAMP was based on the concept of a system theory that focuses on all system components to study and examine the system's behaviours. In systems theory, safety is investigated as '*the emergent properties*' that arise from system component interactions. Therefore, safety becomes a '*control problem*' for which the main goal of control is to prevent or mitigate the occurrence of losses by enforcing the safety constraints on the design of the system.

STAMP extends traditional causality models by preventing component failures and identifying losses that result from interactions among system components. The losses are not the result of interactions among components or events in a linear fashion, but are instead viewed as occurring from '*the inadequate enforcement of constraints on system behaviours*' [86]. The major components of a general control structure in STAMP are shown in Figure 3.1. The application of STAMP is divided into three concepts, as follows:

1. **Safety constraints:** In systems theory, the controller sends certain control actions based on its operational plans (control algorithms). In addition, the controller may receive feedback about its actions. The main aim of STAMP is to investigate the occurrence of losses (accidents) due to the safety constraints having not been successfully enforced by the controller. However, to identify and enforce the safety constraints in the design and operation of a complex system, technical knowledge of design concepts and implementation of hardware and software may be required [86, p. 80].
2. **Hierarchical safety control structure:** In systems theory, systems are organised into levels, each of which involves safety constraints in its activities. At each level a controller is responsible for sending actions and receiving feedback. The

controlled processes perform the following two main tasks: 1) *providing a communication channel among different levels (feedback)* ; and 2) *enforcing safety constraints for which a specific controller is responsible*. Accidents may occur when these processes establish inadequate control, resulting in the violation of safety constraints in lower-level components. To understand and prevent the causes of accidents, the adaptive feedback loops play an important role in identifying the three following faults:

- Missing constraint (unspecified responsibility for safety).
- Inadequate control command (commands are not applied correctly at a lower level).
- Inadequate communicated feedback (feedback is missing regarding constraint enforcement among different levels).

Therefore, it can be said that the hierarchical safety control structure captures relationships and interactions by analysing the system as a set of feedback control loops.

3. **Process Models:** The process model can be defined as '*an adaptive feedback function*' that aims to examine the processes leading to the loss of a system. It also demonstrates the beliefs of a controller on how to handle the current state of the process and the information coming from feedback [86, p. 65].

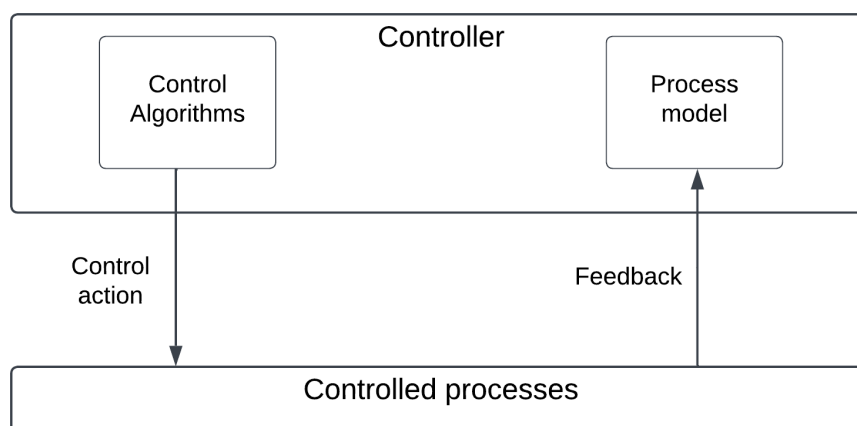


FIGURE 3.1: Major components of an abstract control structure based on STAMP [86, p. 75]

3.1.4.2 Overview of the STPA Method

Based on STAMP, the STPA method was developed to control the potential cause of losses (hazards) during the development of a target system, allowing the hazards to be controlled or eliminated. In contrast with STAMP, the goal of analysis is not the examination of known accidents to detect inadequate enforcement constraints but, instead,

the identification and prevention of new accidents/losses. However, key concepts of STAMP, such as its safety constraints, a hierarchical safety control structure and a process model, are used to accomplish STPA [86, p. 101].

According to the STPA handbook by Leveson and Thomas [84], the steps for applying STPA are as demonstrated in Figure 3.2. They are represented in a horizontal structure of four steps, involving the identification of the following:

1. **Purpose of the analysis:** This step aims to demonstrate high-level system losses (accidents) and their relevant hazards and safety requirements. A hazard is a system state illustrated by system loss and is defined as '*environmental conditions leading to unwanted events*' [84]. The safety requirements emphasise the enforcement of safety constraints and mitigate or prevent a system transition into an identified hazard state.
2. **Inadequate safety control structure:** This step considers events and interactions leading to the violation of a behavioural constraint, then redesigns the control structure as a more effective shield against known violations. This allows more system hazards, such as *component failure, inadequate component interactions and software failure*, to be examined [121].
3. **Unsafe control actions in inadequate safety control structure:** This step aims to elucidate the dynamics of system behaviours to reveal the Unsafe Control Actions (UCAs) of the system under examination. It is an exploration technique for which the type of UCAs can be organised into the following categories:
 - (a) Not providing Control Action (CA) leads to hazard.
 - (b) Providing CA leads to hazard.
 - (c) Time of applying CA.
 - (d) Time of stopping CA.
4. **Missing loss scenarios:** This step identifies the reasons why UCAs may occur in the system under examination. The scenarios are created to explain the following:
 - (a) How inadequate CA could cause UCAs and eventually lead to losses.
 - (b) How safe CA might be designed but not be executed properly.

Finally, once missing scenarios are identified, they can be considered to create additional safety requirements.

Comparing STPA with other hazard identification methods, STPA identifies fault types that will gain importance in future, such as *inadequate component interactions* [24]. By contrast, traditional hazard analysis methods, such as FMEA and FTA, have mainly

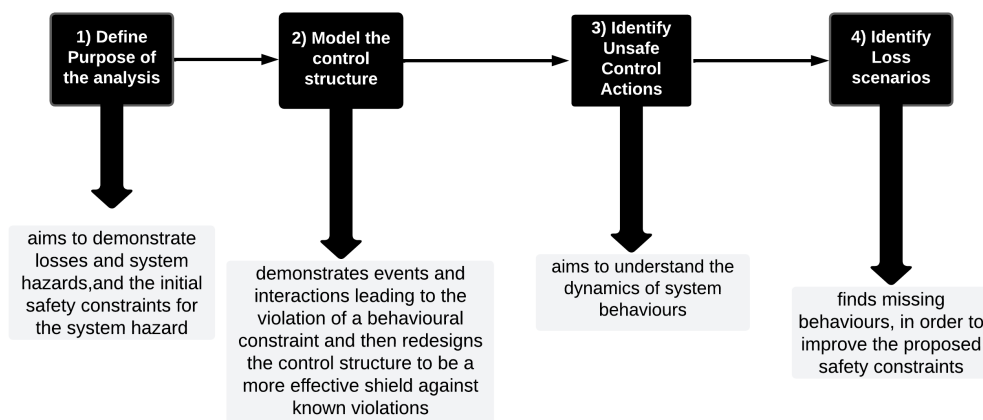


FIGURE 3.2: Overview of basic STPA steps, adapted from [84]

been based on the decomposition approach, which divides the system into components and assumes that accidents (failures) are caused by component failure. In addition, traditional methods focus on hardware errors, which are only caused by the occurrence of component failures [51].

STPA has been implemented into SDV systems to analyse other failures that may have ensued without the occurrences of component failures. Abdulkhaleq et al. [3] propose a dependable architecture for SDV systems based on STPA. The main idea of this approach is that the functional and architectural design of SDVs can be organised into the following three levels:

1. **Vehicle level:** This level considers SDV systems as a single control component that obtains data from sensors and sends control commands to actuate the vehicle.
2. **System level:** This level involves the multiple interdependent software components (e.g. planning system or control system).
3. **Component level:** This level is a low-level view of the SDV system that connects system-level components to actual devices and software functions.

Abdulkhaleq and Wagner [1] applied STPA to the Adaptive Cruise Control (ACC) to show that the result of STPA can be applied to identify potential accident scenarios, such as human decision-making errors, software flaws and component interaction accidents. Moreover, Hanneet et al. [89] applied STPA to a Lane Keeping Assist (LKA) system to derive safety constraints and requirements. However, they consider only the LKA system, and the driven requirements did not cover the interactions of LKA with the human fallback component or other autonomous functions, such as ACC.

3.2 Industrial Safety Methodologies

The safety standards for designing SDV systems are presented in Section 3.2.1. The following Sections (3.2.2 and 3.2.3) discuss the engineering methodologies of certain autonomous vehicle companies, such as Waymo and the General Motors (GM).

3.2.1 Safety Standards

Investigating the industrial safety methodologies of automotive vehicle companies is an ongoing concern. Automotive vehicle companies have developed safety methodologies on the basis of well-known safety protocols, one of which is the protocol proposed by the National Highway Traffic Safety Administration (NHTSA) [8].

The NHTSA safety protocol [8] aims to assess the safety of autonomous vehicles on a broad scale; it is a theoretical safety approach widely adopted in the industry. Specifically, it contains 12 elements that can be categorised into the following two groups: design and testing, as summarised in Table A.1 and Table A.2 in Appendix A. The main interest of the current research is to emphasise the design elements that must be considered in SDVs. Some of these design concepts, such as Operational Design Domain (ODD), Object and Event Detection and Response (OEDR), and the human fallback component, were explained in the previous chapter.

To analyse the complicated design concepts of SDVs, the NHTSA encourages autonomous companies to adopt and implement certain safety standards, such as ISO 26262 [67, 72] and ISO 21448 [73]. The International Organization for Standardization (ISO) standards are a comprehensive safety framework that includes safety management, product life-cycle, analysis methods and safety guidelines. ISO standards, as hazard identification methods, also play a major role in establishing analysis methods for meeting or producing the safety requirements of a software-intensive system [24].

- **ISO 26262:** This is an automotive safety standard that aims to address and generate the functional safety of electrical/electronic systems (E/E) in road vehicles. It defines functional safety as *'the absence of unreasonable risk due to hazards caused by malfunctioning behaviour of electrical/electronic systems'* [72]. It also tries to demonstrate the occurrence of failures at component level. To establish the hazard identification techniques, ISO 26262 identifies several steps to be taken:
 1. **The identification of a system item:** An item is considered a system component or a sequence of multiple system components.
 2. **The identification of possible hazard states:** With the help of hazard identification methods, possible hazard states are identified. A variety of hazard

methods, such as FMEA, FTA and HAZOP, are suggested to provide an efficient hazard identification strategy.

3. **The classification of failures (hazard states):** The measurement of failures is specified through a quantitative method known as an Automotive Safety Integrity Level (ASIL).

Although ISO 26262 has been designed to address the occurrence of failure at the component level, the occurrence of hazard states at the system level remains a significant concern [4]. Therefore, ISO 26262 suggests using techniques such as brainstorming, checklists, FMEA and field studies to identify potential hazard states at the system level.

However, the hazard identification methods of ISO 26262 are outdated for the protection of software-intensive systems [51, 4]. New methods or a combination of hazard analysis techniques are needed to examine the software-intensive system and its dynamic features. A technical paper presented in the Society of Automotive Engineers (SAE) [24] suggests that STPA can be integrated within ISO 26262 to identify a variety of system failures, such as component failures, system failures, inadequate component interactions, software failures and human error.

- **ISO 21448:** This is also a safety standard for driver assistance software, known as the Safety of the Intended Functionality (SOTIF) standard' [73]. Unlike ISO 26262, ISO 21448 is intended to cover the hazard states that might occur without component failure. Therefore, ISO 21448 places emphasis on calculating the operational scenarios that SDVs can perform [78]. These scenarios can be categorised as follows:
 1. Ineffective actions of the awareness system to handle unexpected conditions, namely conditions that have not been covered in the ODD.
 2. The misuse of a fallback human component to handle unexpected operating conditions, namely a fallback human component does not act appropriately.
 3. The evaluation of an ambiguous requirement between a fallback human component and an autonomous controller, namely clarifying what an autonomous controller can do.
 4. The identification of issues that can be observed within the operational environment, namely transportation infrastructure and weather.

3.2.2 Waymo: Safety Methodology

The Waymo safety report [133] involves a comprehensive safety technique for SDVs. It covers all 12 elements of the NHTSA safety protocol, organising them into five levels as follows:

1. **Behavioural level:** This includes the decisions required to obey traffic rules. These decisions rely on the OEDR system and its ODD for handling a range of scenarios. However, the autopilot software has not yet reached automation Level 5.
2. **Functional safety:** This is related to the human fallback component to minimise the severity of failures and return the SDV to a safe state. Therefore, when a failure occurs, the Waymo vehicle tries to ensure that a system has a fallback option.
3. **Crash safety:** The Waymo vehicle adopts crash safety strategies as recommended by the NHTSA. These strategies involve designing a system that ensures minimum damage to the people inside or outside an SDV in the event of a crash.
4. **Operational safety:** This is related to the Human-Machine Interfaces (HMIs) that are usable, convenient and intuitive for passengers. The focus at this level is on permitting passengers some level of control over the SDV in ways that do not affect system safety.
5. **Non-collision safety:** This refers to the design of a strategy that minimises potential hazards to people who may interact with the SDV. For instance, it may involve the physical safety of the first responders (candidates who test a vehicle), mechanics, hardware engineers, and so on.

The Waymo team also tried to identify as many hazard scenarios as possible before testing and releasing their vehicle. The initial identification of hazards relied on the hazard analysis techniques mentioned in ISO 26262. The Waymo team also designed mitigation strategies to eliminate and reduce the effects of these hazards at system level. However, the major concept of the Waymo safety approach relies heavily on an iterative '*build-test fix*'¹ approach to evaluate the correctness of the autopilot software. For instance, the behaviours of autopilot software have been recorded and tested on both private and public roads. These recorded behaviours/data were used to analyse and extract unsafe behaviours of the autopilot software. The Waymo team then developed several mitigation strategies for preventing these unsafe behaviours in the next iteration of building a system.

3.2.3 GM: Safety Methodology

The GM safety strategy [97] closely follows the NHTSA safety protocol, individually addressing each of the protocol's 12 elements. It does not try to simplify or reorganise the NHTSA guidance and focuses on its implementation strategies for accomplishing

¹The '*build-test-fix*' is an iterative software development process that is typically repeated multiple times until the software meets the expected quality standards and requirements. The goal is to detect and fix software errors/bugs in the iterative development process [36].

the required safety assessments. Similar to the Waymo safety approach, the GM safety strategy is built around an iterative *'build-test-fix'* approach to estimate the correctness of the autopilot software. The identification of its hazard techniques can be summarised into the following three types of analysis:

1. **Deductive analysis (Design and process FTA):** This uses hazard methods to identify a system component failure. The mitigation strategy handles the identified failure at component level. However, the outcome of the hazard analysis relies mainly on the development team's experience [51].
2. **Inductive analysis (Design and process FMEA):** This uses a hazard method to identify all possible hazard states, from the highest- to the lowest-level component. Thus, it ensures that the mitigation strategy handles the identified hazards in both low- and high-level components. However, to be able to proceed inductively, the design of the system must first be fully developed [24]. Therefore, the systems must be described in great detail, and the changes required to the design may be cost-intensive.
3. **Exploratory analysis (HAZOP study):** This also uses hazard analysis methods to identify potential hazards or unsafe actions, based on the analysis of a system's functionality. The mitigation strategy addresses the identified hazards at system level, based on the domain knowledge of the development team.

3.3 Formal Methodologies

An overview of formal methods is provided in Section 3.3.1. The use of various formal techniques, as applied to analyse critical properties in critical systems, is presented in Section 3.3.2.

3.3.1 Overview of Formal Methods: Definition, Advantages and Limitations

Definition: From a computer science perspective, formal methods are defined as the mathematical models used to express the properties of a system using a systematic approach. This form of representation can help to demonstrate the specification, design and verification of software/hardware systems [63]. The mathematical background for formal methods is represented in a formal specification language that contains the syntactic (notation), semantic (models' behaviours) and precise rules on how to meet the behaviours of the models.

Advantages: There are many advantages to using formal modelling. The first is the cost of fixing a software error at an early stage of a development lifecycle. For example, the

cost of correcting a software error at the implementation and testing stage can be 100 times the cost of correcting errors at the requirement and design stage [22]. In addition, misinterpreted requirements are a cause of errors in software systems; therefore, formal methods interpret the informal requirements into formal specifications, playing a vital role in reducing the costs of software errors or improving the understandability of the entire system [135].

The second advantage is to use formal reasoning to establish the model's properties via Proof Obligations (POs) [5]. This can help to reveal design errors via the designed models, rather than correct bugs at the testing phase. Consequently, these designed models can be manipulated during the design phase to avoid the need to make significant changes during the implementation phase. POs also enable designers to prove the absence of bugs, whereas testing can only show bugs' presence via the test scenarios.

Limitations: There are some limitations to using formal methods and proving the properties of a system, as demonstrated in the following points [60]:

- **Representation of system properties:** Often not all properties of a system can be modelled. The non-functional requirements, such as performance, are challenging to model and measure on a large scale.
- **Representation of a real-world environment:** Converting the entire real-world environment to a systematic approach or formal model is challenging due to the modelling tools and formal language capabilities.
- **Proving a complicated property:** Some properties are generated from the conjunction of multiple properties that cannot be demonstrated in a formal language or may be impossible to prove.
- **Proving is not always correct:** Mistakes may have happened during modelling and proving; however, the type of modelling tools and their automated theorem provers may help reduce the number of times such mistakes are made.

3.3.2 Formal Specification Approaches

Many modelling approaches are provided in the literature. A recent survey article by Luckcuck et al. [88] suggests that the formalisms for modelling robotics systems, such as SDV systems, can mainly be organised into two approaches: *system-based*; and *logic-based*.

In system-based approaches, the behaviours of a system are modelled by defining its states and operations, which represent the entire system. Examples of this category include formal languages like Z, B-Method, and Event-B. For more information on the syntax and structure of these specification languages, refer to [87].

In logic-based approaches, the behaviours of a system are modelled by defining certain properties that can be expressed through logic. A variety of logics, such as Linear-Time Temporal Logic (LTL), Computational Tree Logic (CTL), Metric Temporal Logic (MTL) and Signal Temporal Logic (STL), specify the dynamic properties of a system over time [88]. For more information on the syntax and structure of these specification languages, refer to [127, 87].

Subsection 3.3.2.1 discusses the role of the Event-B modelling language in supporting SDV systems and hazard identification through STPA. Subsection 3.3.2.2 presents an exploration of the benefits of employing logic-based approaches to SDV systems. Finally, Subsection 3.3.2.3 addresses various other methods that could potentially prove useful for SDV systems.

3.3.2.1 Event-B Modelling Method

Event-B [63] is a formal method commonly used for system development. The main advantage of using it is to introduce the system specifications gradually into the formal model through refinement techniques. A formal model in Event-B has two parts: contexts and machines.

Contexts comprise the static parts of a model, and provide axiomatic characteristics. A context involves definition of the following elements:

1. **Carrier sets:** Abstract types, and cannot be non-empty.
2. **Constants:** Logical variables whose values do not change.
3. **Axioms:** Logical predicates for constraining the properties of carrier sets and constants.
4. **Theorems:** Properties that can be proven on the basis of axioms or other theorems.

Machines describe the dynamic parts of a module. An Event-B machine can involve the following clauses:

1. **Refines:** These are used to introduce more details to a concrete machine (e.g. M_0 indicates the abstract machine, and M_1 refers to a concrete machine, which is defined as 'M1 refines M_0 ').
2. **Variables:** These describe the system states. Variables are defined on the basis of mathematical formulae such as sets, relations and functions.

3. **Invariants:** These describe the properties of a system. They also constrain variables and must always be true. For example, if $I(V)$ represents an invariant for variable (V) , this invariant (I) is held to be true for any change to the value of (V) .
4. **Events:** An event is 'an atomic transition' that changes the states of the system. The transition state of an event is constrained by the guards and the actions.
5. **Guards:** These are referred to in the 'where' clause of an event. Guards are predicates that describe the conditions that must be met for the event to occur. For instance, if an event has more than one guard, for the event to be triggered the conjunction of those guards should hold. If the guards of an event are not met/satisfied, the execution of events stops. This causes deadlock in the system model.
6. **Parameters:** An event may have parameters that cause the guard to maintain a state. They describe how the variables of a machine change simultaneously to preserve the atomic nature of the event.

For instance, for an event e with parameters t and variables v , the guard for the event can be written as $G(t, v)$, and the action for the event as $S(t, v)$, as in Formula 3.1.

$$e \cong \text{any } t \text{ where } G(t, v) \text{ than } S(t, v) \text{ end} \quad (3.1)$$

An event may include no parameters, and have only guards and actions to change its variables directly. The following Formula (3.2) shows how a non-parameter event can be written.

$$e \cong \text{where } G(v) \text{ than } S(v) \text{ end} \quad (3.2)$$

Also, an event may comprise no parameters and guards to initialise the machine variables, as shown in Formula 3.3.

$$e \cong \text{begin than } S(v) \text{ end} \quad (3.3)$$

The actions of events may involve various assignments, expressed as follows:

$$v := \text{Expression}(t, v) \quad (3.4)$$

$$v : \in \text{Expression}(t, v) \quad (3.5)$$

$$v : \parallel \textit{Predicate} (t, v) \quad (3.6)$$

An assignment can be deterministic (e.g. Formula 3.4), where it specifies the value of expression (t, v) to v . By contrast, an assignment can also be non-deterministic (e.g. Formulae 3.5 and 3.6). Specifically, Formula 3.5 assigns any value from the set of expression (t, v) to v , while Formula 3.6 assigns any value fulfilled in predicate (t, v) to v .

Moreover, as the invariants $I(v)$ are inductive they must be maintained by all events. This ensures that all events strictly obey the invariants when an event attempts to modify the state of variables. For this reason, Event-B machines use theorem-proving to verify the consistency of events. Moreover, the Event-B machines apply model-checking techniques that aim to explore all reachable states of the system while transforming the invariants as the safety properties of a system.

Some contributions of using Event-B within SDV systems and STPA are discussed in the following points

- **Constructing Event-B models for the system based on the system requirements**

The details of this approach were inspired by the cookbook [26] for the modelling and refinement of control systems, which consists of a plant (environment), controller and, in some cases, the human operators who may interact with a system. The modelling steps suggested in the cookbook are based on the four-variable model by Parnas and Madey [107]. The guidelines in the cookbook suggest that the phenomenon of a system can be divided into the following two categories: 1) variables that describe critical properties between environment and controller; and 2) variables that may be used to represent the interaction between the human operators and the environment.

Three contributions based upon the four-variable model of Parnas and Madey [107] relate to this approach to modelling the functionality of the autonomous controller: a Cruise Control System (CCS) [137]; Lane Departure Warning System (LDWS) [136]; and the Speed Control System (SCS) [90]. However, it can be said that these autonomous functions have a low level of automation (Level-1) in which advanced human fallback features, such as the awareness of the driver, are completely ignored.

- **Constructing Event-B models based on the safety constraints of an autonomous function:** The SDV must implement fail-safe mechanisms [80], often known as the ‘policing function’ [23]. The concept of fail-safe mechanisms focuses on functional requirements, and is part of the system requirements. The policing function can check the output values of autonomous modules, such as a perception module at runtime [23]. The essential steps in using a validation technique such as

a policing function are to demonstrate the safety constraints on the autonomous functions. These safety constraints are used either to validate the output of the autonomous functions or to detect failures in the runtime. Hoang et al. [65] observe that the concept of metamorphic relationships, which aims to discover an expected relationship between inputs and outputs, can be used to identify safety constraints. Therefore, to adopt this approach, the safety constraints of an autonomous function must be identified.

- **Constructing Event-B models based on hazard identification methods:** The Event-B formal method was used with the hazard identification methods to ensure that the identified hazards were prevented or mitigated at the early stage of design. Colley and Butler [35] developed a method to demonstrate and formally analyse the critical requirements (artefacts) generated by the method of STPA analysis. The goal was to use modelling techniques, such as formal verification, by the use of the Event-B formal method and its Rodin toolset. The model detects vulnerable system states within the resultant model. Similarly, Howard et al. [68] adopted STPA to generate a critical requirement, while formal models were constructed in Event-B to verify that those security requirements completely mitigate against vulnerable system states. However, Dghaym et al. [40] extended the work of [35, 68] to develop a compositional approach to elicit the critical requirements for autonomous functions and then formalise these critical requirements into Event-B models.

3.3.2.2 Logic-based Approaches

The main idea of these logics is to identify events occurring either next, globally or eventually on the basis of the nature of the properties under investigation [28]. Some contributions of using logic-based approaches with SDV systems are discussed in the following points:

- **Verifying decisions of autonomous controller:** The concept of a rational agent is commonly used to focus on the autonomous controller, who is responsible for making decisions, and to simplify the complexity of SDV systems. More specifically, a rational agent is a software that can perceive its environment via sensors and explain its intentions [113]. In order to use the concept of the rational agent, the logical requirements (rules) must be defined. Subsequently formal methods, such as LTL, can be used to verify the beliefs of rational agent software. Fisher et al. [50] proposed a methodology for autonomous systems based on the concept of a rational agent. The main idea of this methodology is to divide an autonomous system into the following three stages: 1) the environment stage, which uses sensors to receive the real-world environment; 2) the continuous/-dynamic functions, which apply Artificial Intelligence (AI) techniques to obtain

knowledge from the environment stage; and 3) the rational agent level, which involves the possible decisions of the autonomous system (rules) based on stages 2 and 3. Related to Fisher's methodology are a variety of case studies [75, 76, 49] that have used a GWENDOLEN programming language² to implement a rational agent software to establish a module checker (Agent Java Pathfinder (AJPF)) and verify the safety properties of SDV systems at runtime. For instance, the case study mentioned in [49] focused on high-level decisions in SDV systems, where the rules of a rational agent are defined through the following steps: 1) vehicle navigation; 2) obstacle avoidance; 3) obstacle selection; and 4) vehicle recovery.

- **Verifying perception tasks of autonomous controller:** The STL was used to capture the requirements of the perception module [127], for instance a *requirement R2* presented as '*Sensor S should detect its visible/target obstacle within T1 time unit*'. Then, it interoperated into an STL formula to support the control design and testing phase of the system under investigation. The driving simulation software 'Sim-ATAV'³ was used to explore and test the proposed requirements. However, in order to reason about the performance of the perception module, Time Quality Temporal Logic (TQTL) was introduced as a formal language to monitor the outputs and awareness algorithms for SDV systems [42]. TQTL attempts to specify the minimum requirements that can be used to either test or verify the final output of the perception module. For instance, a *requirement TQTL-R1* may be proposed as '*at every time step, for all the objects (id1) that were observed and detected by the perception module in the frame (vision), if the object class is pedestrian with of probability of more than 0.7, then in the next 5 frames, the object id1 should still be classified as pedestrian with a probability of more than 0.6*'. Next, the temporal logic formula was written to demonstrate a set of formal requirements.

3.3.2.3 Other Approaches

Finite-State Automata (FSA) is an approach used to specify system behaviours as a state-transition system. It involves formalisms for discrete-system events, including capturing time and probabilistic transition. For instance, Guidolini et al. [58] note that FSAs have been used to handle pedestrians at pedestrian crossings for the Intelligent Autonomous Robotic Automobile (IARA's autonomous vehicle)⁴, as shown in Figure 3.3. However, the case study by Guidolini et al. [58] handles only a single driving scenario, so Aeberhard et al. [9] proposed a hybrid deterministic model to illustrate the processes of selecting a driving action. In their model, the vehicle control variables

²GWENDOLEN is a programming language for developing intelligent agents. It is primarily used to specify the behaviour of an agent and its interactions with other agents [39].

³Sim-ATAV is a software platform for simulating SDV systems. It provides a virtual environment for testing and evaluating the performance of SDVs and their algorithms [126].

⁴The Intelligent Autonomous Robotic Automobile (IARA) is an autonomous vehicle project developed by the Federal University of Espirito Santo in Brazil [14].

are divided into two classes. The first is a finite set of lateral control variables that concern lane-keeping and lane change states. The second involves cruise control states and their corresponding longitudinal control variables. The selection of behaviours was established by evaluating the current vehicle state to specify the appropriate driving manoeuvres.

The ontology-based approach has also been used to describe the driving environment and reason for the actions taken by a decision-making module. The main idea of ontologies is that the domain of knowledge of a target system can be used to provide the “key concepts, properties, relationships and axioms of a given domain” [109]. For instance, Zhao et al. [141] have used this approach to implement a decision-making module based on knowledge obtained from traffic regulations (e.g. ‘Stop’, ‘To the right’ or ‘Give way’) and the location of a vehicle in its environment. Their focus was on the traffic scenarios that appear at intersections and on narrow roads; therefore, the rules were implemented into a decision-making module on the basis of the regulation signs of a specific location in the environment. The main drawback of this approach is that it is time-consuming to design an optimal world model composed of every traffic item, such as the lanes at every locality or city [88].

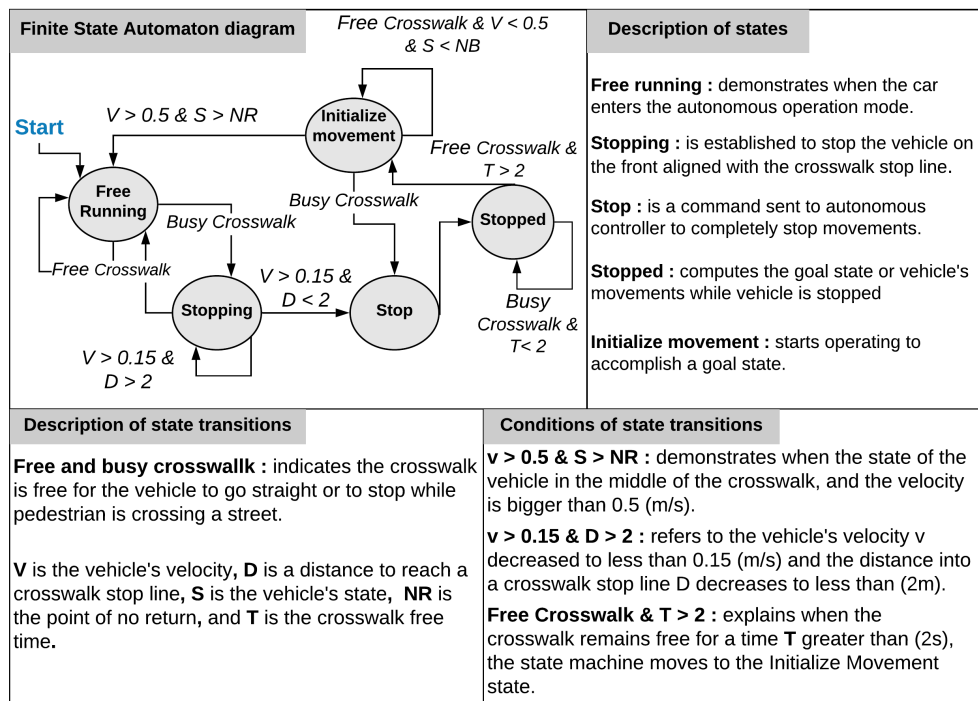


FIGURE 3.3: Finite-state automata of IARA's vehicle for handling pedestrians at pedestrian crossings

3.4 Discussion

The primary aim of this chapter is to review literature sources to identify both informal and formal methods used for analysing safety-critical systems, including SDV systems. This objective directly contributes to addressing [RQ1.2](#).

Traditional informal hazard methods, such as FMEA and FTA, mainly rely on a probabilistic approach to assess the severity and probability of risk (hazard) at the component level, while the STPA method focuses on the dynamic control of a system and on causes of hazards in the absence of component failure. STPA can be adopted at an early stage of design to emphasise the safety requirements of a system [84]. By contrast, traditional hazard identification methods can be adopted after the design of a system has been completed, which makes any changes to the design expensive [51].

Automotive vehicle companies rely on the NHTSA safety protocol for developing the autopilot software. The NHTSA protocol is a comprehensive safety framework that involves 12 elements for assessing SDVs. This protocol also encourages automotive companies to implement certain safety standards, such as ISO 26262 [67] and ISO 21448 [73], which involve a variety of hazard identification methods, such as FTA, FMEA and HA-ZOP. However, these methods are outdated and were developed mainly to handle component failure. There is a need for a new hazard method or a combined method of multiple hazard identification techniques to consider the dynamics of system behaviours. According to the Society of Automotive Engineers (SAE) [24], STPA could be integrated within ISO 26262 to identify other failures, such as inadequate component interactions, which may occur either with or without component failure.

Formal methods have been used in many ways to emphasise and support a critical property of the software under investigation. The system-based approach aims to specify the entire states and operations of a target system. However, gathering the system requirements of a software-intensive system, such as SDV systems, may be challenging. Therefore, based on the safety constraints of the functional requirements, a runtime monitoring function called "*policing functions*" has been developed to emphasise critical properties of a target system at the runtime [65]. The logic-based approach attempts to identify the critical properties of the system under investigation over time. However, this means that the logical requirements (rules) must be identified to examine or verify the selected properties at runtime. Other approaches, such as FSAs and ontologies, aim to identify the behaviours of a software-intensive system on the basis of modelling the target environment of the system under investigation. These approaches attempt to specify the actions that may be taken by the decision-making module. This way of modelling, which involves creating and designing the optimal world model of a target system, can be time-consuming.

Finally, the existing literature highlights a clear gap concerning the need for new rigorous and systematic methods or a combined approach of multiple techniques to account for the dynamics of system behaviours. These methods should accurately identify safety requirements and formally represent them with associated formal representations to ensure consistency and safety properties.

3.5 Conclusion

This chapter presents a review of both informal and formal methods for system safety analysis in critical systems such as SDV systems. Informal hazard identification methods, such as FMEA, FTA, and STPA, are discussed, followed by an exploration of rigorous analysis techniques. In the next chapter we investigate a case study conducted for this research.

Chapter 4

Case Study

This chapter provides a discussion of a case study that is relevant to the context of this research. The selection of the case study is based on three main criteria:

1. **Complexity:** It must involve both the human driver and the autopilot software to execute the Dynamic Driving Tasks (DDTs).
2. **Control speed and steering features:** It must explore how the autopilot software autonomously sets the steering and speed of Self-Driving Vehicles (SDVs).
3. **Capability for human intervention:** It must consider the ability of a human driver to assume control of an SDV when an intervention request is sent.

Considering these parameters, the Automated Lane Centring (ALC) case study was chosen. The ALC system is a semi-automated system that collaborates with the human driver to accomplish DDTs. It is a sophisticated system encapsulating multiple functionalities, including Lane Keeping Assist (LKA), Adaptive Cruise Control (ACC) and Driver Monitoring System (DMS). Additionally, the human driver is required to assume a fallback role during critical driving scenarios.

Section 4.1 investigates the characteristics and functionalities of the ALC system. Section 4.2 summarises the ALC system as four main features, highlighting the complexity of performing DDTs within the functionalities of LKA, ACC and DMS. Section 4.3 gives a conclusion of this chapter.

4.1 Automated Lane Centring

The ALC system is a well-known feature of the Advanced Driver Assistance Systems (ADAS) and is categorised as being at automation Level 2 and Level 3 [99, 104]. It uses

sensors, cameras and other onboard technologies to detect and monitor the lane lines on the road surface. If the Self-Driving Vehicle (SDV) begins to drift or deviate from the centre of the lane, the ALC system autonomously makes minor steering corrections to reposition the SDV in its target lane. The ALC system is designed to function at various speeds and in various driving conditions, thereby enhancing comfort and reducing the workload on the human driver of the SDV.

It is critical to note that the ALC system does not eliminate the need for a human driver. Despite the assistance with lane centring, the human driver must remain engaged and attentive, prepared to resume manual control when necessary. For this reason, the ALC is often paired with a DMS to ensure driver attentiveness for safe driving operation [104].

Subsection 4.1.1 explores the features of an ALC system that help to keep the SDV within its target lane. Subsection 4.1.2 examines the human monitoring features that the DMS uses to ensure driver attentiveness, especially when the driver might need to take control of the SDV. Subsection 4.1.3 provides further analysis of the ALC, specifically examining the real-world autopilot software known as OpenPilot¹ which seeks to achieve the ALC functionalities.

4.1.1 ALC Features for Lane Stability in SDV

The primary function of the ALC system is to maintain the SDV in the centre of its desired/target lane, with the human driver responsible for performing lane change manoeuvres [18, p. 3]. The ALC system operates in conjunction with other SDV components, importantly alerting the human driver to its driving actions. Additionally, the ALC system takes advantage of the lower-level physical components of an SDV. It uses the braking and steering systems to keep control and ensure that the SDV stays within the target lane even when navigating bends in the road [10].

The ALC system follows the same architecture that is outlined in Section 2.3 of Chapter 2. In practical terms, the decision-making module is also known as the planning module. For example, the open-source driving assistance software, OpenPilot, structures the ALC system into the following three main modules: *perception*, *planning*, and *control*. According to Jiao et al. [74], OpenPilot's architectural design is similar to that of Tesla's autopilot software.

Both academic researchers and industry practitioners highlight the variety of features in an ALC system. Some researchers emphasise that the objective of an ALC system is to identify the desired path and adjust the steering of the SDV to remain centred within a target lane [10, 71], while others identify additional ALC features, such as issuing

¹OpenPilot, is an open-source driver assistance system with advanced Level 2 autonomous driving features. It includes steering and speed control to keep an SDV inside its target lane [10].

warnings to the human driver to undertake corrective action [41]. However, a technical report by Mei et al. [41] and feedback from an engineering expert from General Motors (GM) suggest that the ALC includes the following three main functionalities that describe its high-level system requirements:

1. **Lane Departure Warning (LDW):** This issues a warning to the human driver when the SDV is about to depart its current lane. The human driver is then responsible for applying corrective action.
2. **LKA:** By making a slight adjustment to the lateral control feature (steering), this prevents the SDV from drifting into another lane. However, due to the limitations of the LKA in managing complex driving scenarios such as abrupt lane changes, the human driver's input continues to be essential for implementing required corrections.
3. **ALC:** This function is integrated with several automated features, including LKA, ACC, and DMS. This technology allows the system to qualify as a Level 2 or Level 3 automated driving system [18, p. 2]. First, the LKA maintains the SDV within its target lane by continually adjusting its steering. Second, the ACC ensures a safe distance from the vehicle ahead by regulating the SDV's speed. Last, when the LKA or ACC features are active, the DMS monitors the driver's awareness.

4.1.2 DMS Features for Safeguarding Driver Alertness in SDV

To compute the awareness of a human driver, the DMS can be considered to have either active or sensitive features. For instance, an active human feature might be the touch of hands on the steering wheel, while a sensitive feature might be the detection of eye gaze by a camera inside the vehicle. The concept of active and sensitive features has been adopted by the autopilot software currently on the market. According to Volvo's autopilot software [131], to activate the ALC function the human driver must be hands-on at the steering wheel. In contrast, Tesla's autopilot software [102] and the OpenPilot software [10] police the human driver by a combination of both active and sensitive monitoring features. For instance, Tesla's autopilot software models 3 and Y [102] employs a sensitive monitoring feature that uses an in-car camera to detect and monitor the human driver when the advanced driver assistance system is in operation.

In OpenPilot [10], the DMS operates by tracking precisely various indicators to ensure the driver's engagement and attentiveness during autonomous driving operation. These indicators often involve parameters such as the driver's hand position on the steering wheel and their eye and head movements. If the DMS determines that the driver is paying insufficient attention to the road, it immediately initiates an alert. This can be manifested as either auditory signals or visual cues on the vehicle's dashboard.

If the driver fails to respond to these alerts or their level of attentiveness does not improve, OpenPilot implements a risk mitigation plan. This plan involves slowing the vehicle down to prevent an accident when there is no human driver ready to take control; if necessary, this brings it to a complete halt.

There are several notable advantages to incorporating a DMS within an ALC system [18, 46]. These benefits include:

1. **Enhanced safety:** The DMS continuously monitors the driver's awareness level. This ensures that the driver is ready to intervene and take control when either a system signals the need for intervention or there is a system malfunction.
2. **Supervisory control:** Even with semi-automated systems such as ALC it is crucial that the driver can intervene promptly during unexpected situations. The DMS ensures that the driver's attention remains on task, ready to take control of the vehicle.
3. **Supporting gradual transition to full autonomy:** By allowing a combination of human and automated control, the DMS facilitates a smoother and more gradual transition towards fully autonomous vehicles (Levels 4 and 5).

4.1.3 Further Analysis of the ALC on the OpenPilot Platform

This subsection explains the exploration of the ALC on the open-source platform, *OpenPilot*. The OpenPilot platform [10] is well-known driver assistance software used in automotive field research. It provides the same functionality as the existing driver assistance systems of most new vehicles on the road. It employs a development methodology similar to Tesla's autopilot software, known as 'end-to-end' learning methodology [94, 71]. This end-to-end learning methodology simplifies the complexity of performing the Dynamic Driving Task (DDT) into a single Artificial Intelligence (AI) framework, which involves perception, planning, control and DMS, as discussed in Section 2.3 in Chapter 2.

In this exploration, two main objectives guide our study of OpenPilot. The first is to discover which features of the ALC system keep the SDV in its target lane. The second is to study how the software behaves in the CARLA driving simulator² [43] to identify those situations where a driver may need to take control of the SDV. Details on setting up the CARLA simulator are in Appendix B. The summary of the exploration process in the ALC system provided by OpenPilot is detailed in the following subsections.

²CARLA (Car Learning to Act) [43] is an open-source autonomous driving simulator. It is a flexible, versatile tool created to facilitate the development, training and validation of autonomous driving systems.

4.1.3.1 Overview of OpenPilot Software with LKA, ACC and DMS Features

Figure 4.1 illustrates the interactions among OpenPilot's internal modules. A summary of these interactions is provided below:

- Human driver interactions:** The in-car display system acts as a communication channel between the ALC system and the human driver. This interface integrates the DMS feature to evaluate the driver's awareness. Furthermore, the ALC system uses this display to alert and inform the driver about the operation of the ACC and LKA features.
- Perception and functionality:** The LKA and ACC features begin by estimating the driving environment using sensors such as cameras and radar. Within the perception module the ALC system recognises the lane position and calculates the distance to the leading vehicle on the basis of sensor data. The planning module then specifies the necessary changes to steering and speed to keep the SDV within its target lane and at a safe distance from the vehicle ahead. The control module then implements these changes, adjusting the SDV's position accordingly.

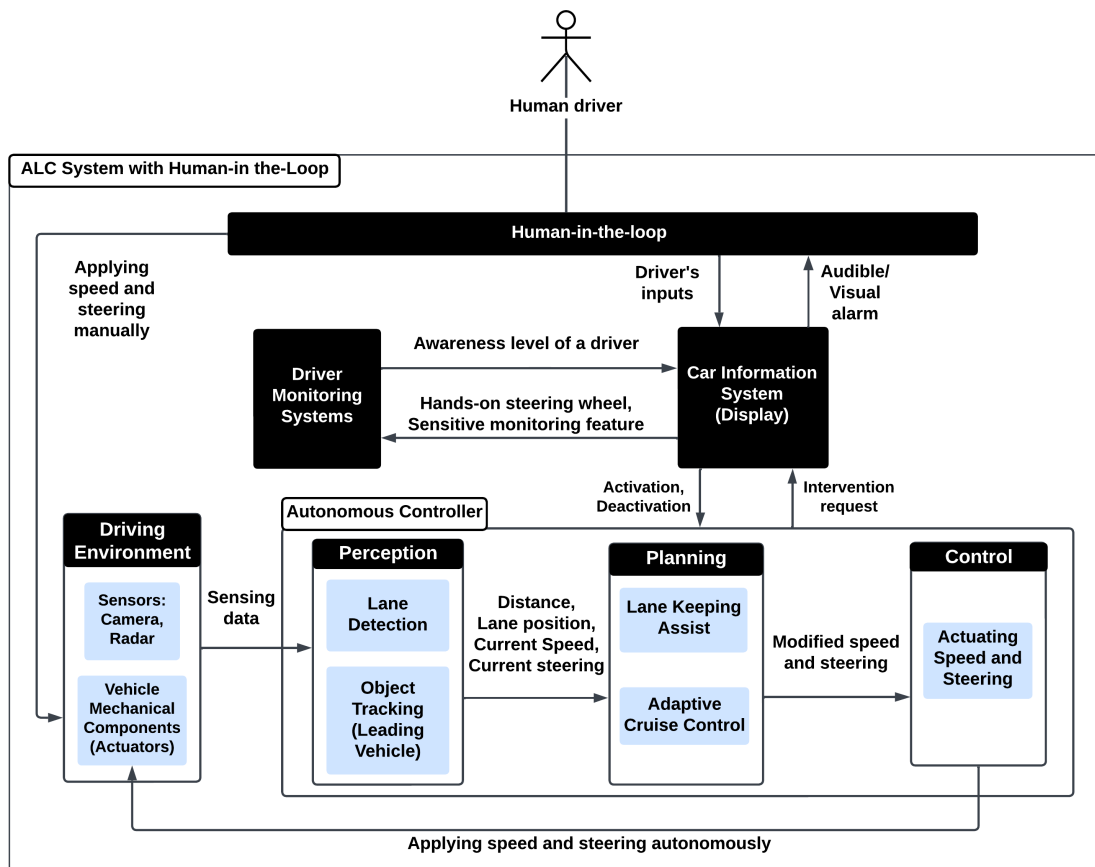


FIGURE 4.1: Relationships between OpenPilot's internal modules

4.1.3.2 LKA and ACC Control in OpenPilot

In controlling steering, OpenPilot uses computer vision-based algorithms to analyse surrounding environmental features and make necessary adjustments. The functionalities of both the LKA and the Lane Departure Warning (LDW) are enabled by an end-to-end deep learning model. This model takes as its inputs both the SDV 's current state and the latest images from the SDV 's front-facing camera [30].

Speed control in OpenPilot is managed through the ACC function. The ACC uses a radar sensor to maintain a safe and consistent distance from the vehicle ahead. This feature not only aids in maintaining a constant speed but contributes to system safety by minimising the risk of rear-end collisions.

OpenPilot incorporates three main modules to control the speed and steering of an SDV. First, the perception module helps the system to comprehend its driving environment by using incoming images and radar readings to detect vital features such as the position of a leading vehicle or lane lines. OpenPilot assesses the success of this detection process by a detection score. This score, ranging from 0 to 100%, indicates OpenPilot's confidence in its perception capabilities. Based on this interpretation of the incoming data, OpenPilot uses the incoming data to calculate a desired path. This path is crucial to maintaining the SDV within its target lane. Additionally, it guarantees a safe distance from any leading vehicle [10].

Second, the planning module uses the calculated desired path to decide on a target position, with the necessary adjustments to speed and steering. The target position is a location within the target lane that the SDV aims to reach, while the required speed and steering adjustments aid the low-level physical components of the SDV in reaching that target position.

Last, the control module implements changes in speed and steering to support the SDV in reaching its target position. It uses Model Predictive Control (MPC) to determine the most suitable steering angle and speed. This is done using three factors [48]:

1. **System dynamics:** This involves a continuous adjustment of speed and steering. The adjustment aims to navigate the SDV to its target position.
2. **Leading vehicle constraints:** These help prevent the SDV from colliding with the vehicle ahead.
3. **Speed and steering angle constraints:** These constraints prevent the SDV from changing its steering or speed too quickly, ensuring that the human driver has sufficient time to respond safely if the ALC system requests an intervention.

4.1.3.3 DMS in OpenPilot

As OpenPilot software is classified as a system with Level 2 capabilities [30], the DMS in OpenPilot is created to increase safety by ensuring that the driver stays alert and engaged while driving, even when the ALC features are enabled. Thus, the human driver is still responsible for the overall control of the SDV and must be ready to intervene when necessary.

The DMS in OpenPilot uses a mix of active and sensitive monitoring features [10]. Similar to Tesla autopilot software [102], these features make sure that the driver is aware and can react if the ALC system issues a request to intervene. For instance, it may send the driver one of two intervention/warning messages:

1. **'Always keep hands on the steering wheel and eyes on the road'**: This determines the awareness level of the driver required to activate the OpenPilot software.
2. **'Take control. Turn exceeds steering limit'**: This indicates that the system is to release control of the SDV and that the driver is to be responsible for navigating the vehicle in its driving environment if the ALC system exceeds the range for the steering angle.

Due to the unpredictable behaviour of human drivers, such warnings and intervention requests may be ignored. Thus, it is crucial to define a limited time for an SDV to wait for the human reaction. According to a study of the Crash Warning Interface Metrics program (CWIM) about LDW notifications [82], after an auditory notification a human driver may take approximately 700 milliseconds to override the system and provide a steering response. Other studies suggest a longer interval time, with the human driver taking approximately 10 seconds to refocus and pay attention to the road [93]. For this reason automotive driving applications such as Tesla and OpenPilot have implemented notification systems to maintain human drivers' attentiveness. For instance, when an intervention request is sent, OpenPilot [10] allows a driver 4 seconds to react. If the driver disregards this request, OpenPilot issues an auditory notification and, after 6 seconds, reduces the speed of the vehicle until the SDV is brought to a complete halt.

4.2 DDT Features Derived from the ALC System

The ALC system can be broken down into several high-level features, as follows:

- **Feature 1:** This initial interpretation aims to understand how the ALC system interacts with the driver.

- **Feature 2:** This extension contains two crucial functionalities: the LKA and the DMS. The LKA is responsible for controlling the SDV 's steering angle to ensure that the vehicle remains within its target lane. Concurrently, the DMS monitors the driver's attentiveness inside the SDV to ensure their awareness to take control when necessary.
- **Feature 3:** This extension considers both LKA and ACC functions to control steering and speed to keep the SDV inside the target lane. The ACC manages the speed of the vehicle, adapting it as needed to maintain a safe distance from a leading vehicle on the basis of the driving conditions.
- **Feature 4:** This extension involves modelling the driver's potential responses when the ALC system prompts the driver to intervene. Various factors are taken into consideration, such as the driver's reactions, responsiveness and ability to take corrective action.

Each of these aspects of the ALC system is summarised in its own subsection.

4.2.1 Feature 1: High-Level Interactions Between ALC System and Human Drivers

This subsection details the key high-level interactions between the ALC system and the human driver. These interactions illustrate how the ALC and the human driver engage together to accomplish the DDTs, as follows:

1. The ALC system can control the SDV to reach a new position within the target lane.
2. During the occurrence of critical driving scenarios, the ALC system relies upon the human driver's intervention to control the SDV.
3. The human driver is required to be ready to take control of the SDV, with the purpose of performing the entire driving task.

4.2.2 Feature 2: The LKA and DMS Functions in the ALC System

This subsection offers an overview of the high-level operations of the ALC system for the LKA. These operations are as follows:

1. The perception component of an ALC interprets sensing data, such as images, to identify perceived environmental features such as lane markings and their detection score.

2. The desired path to keep an SDV within its target lane is estimated on the basis of how the perception component interprets the perceived environmental features.
3. The decision component uses the identified/desired path to specify the target position and the adjustments to steering necessary to reach that position.
4. The control component is tasked with actuating the new steering to reach the target position.
5. The ALC system can depend on the human driver of the SDV when a critical driving scenario arises.

Furthermore, the ALC ensures the driver's awareness through the DMS component. The high-level operations of the DMS are as follows:

1. The DMS computes the driver's awareness on the basis of both active and sensitive monitoring features.
2. If the driver fails to provide the necessary level of human monitoring features, the DMS deactivates the ALC system.

4.2.3 Feature 3: The LKA and ACC Functions in the ALC System

This subsection expands upon the previous ALC feature by incorporating both the LKA and ACC for controlling the speed and steering of an SDV. The ALC's operations within the DMS remain consistent; however, adjustments are made to the autonomous determination of speed and steering by the ALC system, as follows:

1. The perception component of an ALC processes various sensing data, such as images and radar points. This data is used to detect and understand key environmental aspects, such as lane markings and the position of any potential leading vehicle, along with a score that indicates confidence in these detections.
2. The perception component then uses this understanding to estimate the optimal path to keep the SDV within its target lane. This estimate includes considerations for potential leading vehicles.
3. Once the desired path is established, the decision component then specifies the target position and determines the necessary adjustments to speed and steering required to reach this target position.
4. Subsequently, the control component is responsible for actuating the modified steering and speed parameters to reach the target position.
5. In a critical driving scenario the ALC system continues to rely on the human driver of the SDV.

4.2.4 Feature 4: Modelling the Driver Reactions in the ALC System

This subsection observes the temporal dynamics of the driver's response when the ALC system initiates a request for manual intervention:

1. The ALC system issues a request for intervention and initiates a pre-specified time (countdown) for the driver's response.
2. If the driver does not respond within the specified time, the ALC system triggers an auditory alert to attract the driver's attention.
3. From the moment of the auditory alert, the driver is given a further specific time window in which to react.
4. If the driver still does not react within the specified time, the ALC system proactively reduces the SDV's speed.

4.3 Conclusion

This chapter discussed a case study relating to the context of this thesis. The ALC case study was selected because it represents the complexity of controlling various functions in SDV systems along with the critical participation of human drivers in performing DDTs. In the following chapter we present the initial version of our methodology, focusing on the development of a rigorous analysis template.

Chapter 5

Rigorous Analysis Template

This chapter introduces our first contribution, entitled the *Rigorous Analysis Template (RAT)*. Parts of this chapter were published in COMPSAC 2022 [12]. Specifically, that material corresponds to the content in Section 5.2.

The Rigorous Analysis Template (RAT) highlights the interplay of automation by an Self-Driving Vehicle (SDV) and a human driver, especially when the SDV considers the human driver as a fallback mechanism to handle hazardous events. The development of RAT represents a significant advancement in identifying automation aspects and analysis processes for SDV systems. Its design aims to establish an analytical framework that specifically addresses the main Research Question (RQ2) of this thesis.

RQ2: *How can an analytical framework be designed to systematically highlight the roles of both a human driver and an SDV system during autonomous operation?*

RQ2 aims to design an analysis framework that emphasises the roles of a human driver and an SDV system during autonomous operation. This includes identifying key automation features and explaining how the general scope of analysis might be performed. Section 5.1 justifies the creation of a template for SDV systems and selection of informal and formal methods. Section 5.2 provides an overview of our approach and the processes involved. Section 5.3 explains how the RAT framework tackles RQ2. Section 5.4 presents related work and Section 5.5 gives a conclusion to this chapter.

5.1 Justifications

This research develops an analytical framework to capture and analyse safety properties in SDV systems. Our aim is to identify unsafe or inadequate high-level system

component interactions between drivers and SDV systems. The main idea is to provide a universal methodology applicable to any semi-automated system (automation levels 1 to 3). To achieve this, we have divided our approach into three main aspects:

1. A template that identifies generic aspects of automation between drivers and SDVs by explaining the system component interactions involved in SDV systems.
2. Selection of a suitable hazard identification method to identify inadequate or unsafe system component interactions between drivers and SDV systems.
3. Integration of formal and hazard identification methods to enhance the quality of the analysis by proving that the occurrence of hazardous events has been mitigated at the design level.

Each aspect is detailed in its own subsection below.

5.1.1 Motivation Behind Automation Aspects in Template

The Society of Automotive Engineers (SAE) (J3016) [101] affords SDVs' autonomy various levels, as mentioned in Table 2.1. These levels refer to Dynamic Driving Task (DDT) aspects such as lateral and longitudinal control, Object and Event Detection and Response (OEDR) and fallback on a human driver by autopilot software. According to this standard, most manufacturers' systems remain in semi-automation (Levels 1 to 3) due to the ambiguity of the collaborative requirements between the human driver and the autopilot software. For instance, automation Levels 1 and 2 rely on the human driver to handle any hazardous events that may arise in the driving environment, while at automation Level 3 the human driver is merely the *'fallback-ready user'*. According to the SAE [101], a *'fallback-ready user'* refers to the assumption that the user (human driver) is receptive to any requests to intervene and immediately become the driver. Therefore, a system with Level 3 capabilities must first detect a hazardous event and then inform the human driver to take control of the vehicle.

Due to the limited capabilities of systems to perform OEDR features, the key concern about automation levels is the transition point from Level 2 to Level 3 [20]. The main reason for this concern is that a system with Level 2 capabilities is limited in its ability to observe and respond to all the relevant objects in its Operational Design Domain (ODD). Therefore, the human driver is expected to supervise the driving environment and immediately take control of the vehicle when a hazard arises. By contrast, the SAE assumes that a system with Level 3 capabilities is able to detect and respond to the most highly relevant objects; therefore, if a critical driving scenario unfolds, the system must instruct the human driver to take control of the vehicle within an adequate timeframe.

To perform their DDTs, SDVs involve a complex system of multiple components/modules. As seen in Figure 2.4 in Chapter 2, the complexity of a system can be simplified into four components. However, there is a need to link DDT features such as OEDR, ODD and fallback to the internal components that are involved in the SDV systems. Therefore, our template aims to accomplish two main goals as follows:

1. Clarify and identify the automation aspects between human drivers and SDVs during the performance of DDTs, especially if a system requires a human driver to play a fallback role to ensure the safety of a system.
2. Link the automation aspects of DDTs with the internal components of SDV systems in order to specify the responsibility of either human drivers and SDV systems during the performance of DDTs.

5.1.2 Hazard Identification Methods

The choice of a hazard identification method depends on how it can be applied practically to analyse complex system component interactions, such as those between SDVs and human fallback drivers. Based on our exploration of various methods in the cyber-physical domain in Section 3.1, Systems Theoretic Process Analysis (STPA) was chosen for the following reasons:

1. STPA focuses on the dynamic control of a system and identifies causes of hazards even in the absence of component failure. Additionally, studies such as [19, 24] suggest that STPA can investigate any inadequate or unsafe component interactions that may occur, with or without component failures.
2. STPA can be applied in the early stages of design to emphasise the Safety Requirements (SRs) of a system [84]. This helps to guide the development of the system and avoid costly changes after the architectural design of the target system has been built.
3. STPA is an iterative analysis methodology that allows for gradual development of the target system by iterating the design and re-performing the analysis.

Furthermore, STPA has been involved in various ongoing research projects at the home institution. For example, Colley and Butler [35] developed a method to formally analyse the critical requirements (artefacts) generated by the STPA analysis method. Similarly, Howard et al. [68] adopted STPA to develop critical requirements for safety and security, which formally mitigate against vulnerable system states. Dghaym et al. [40] extended the work of [35, 68] to develop a compositional approach for eliciting critical requirements for autonomous functions based on the STPA method. Therefore, we aim

to extend these works to SDV systems. Moreover, the expertise of the research community at the home institution can significantly enhance the quality of the research.

5.1.3 Formal Methods

The choice of a rigorous specification approach rests on how well it can capture the driven-STPA requirements. Based on the exploration of various methods in the cyber-physical domain in Section 3.3, the Event-B modelling approach was chosen for several reasons:

1. Event-B supports modelling of design aspects at system level, rather than just at software level.
2. Event-B introduces system specifications gradually into the formal model through refinement techniques that enable traceability of critical requirements with associated formal representation.
3. Event-B is supported by the Rodin toolset [7], which includes both theorem-proving and model checking (ProB) [83].

The University of Southampton has a strong research track record in formal methods and, as for the STPA method, has applied Event-B to develop several safety-critical systems. Consequently, existing expertise within the research community at the home institution facilitates the progress of this study to a superior level of quality.

5.2 Rigorous Analysis Template

This section presents the process of constructing and employing RAT framework. The approach, as illustrated in Figure 5.1¹, consists of two primary phases. The *aspect identification* phase aims to identify the various considerations to be taken into account when designing SDV systems. The second phase of *analysis process* applies the STPA and Event-B to identify and ensure the mitigation of unwanted interactions between an SDV and human drivers at system level. To validate our approach, we show how an Automated Lane Centring (ALC) system encapsulates high-level system component interactions between the autonomous and the human controllers in a process initially summarised in Section 4.2.1 of Chapter 4.

Subsection 5.2.1 outlines the creation of a template for SDV systems, with a specific focus on identifying automation aspects. Subsection 5.2.2 explains how we apply the template to the ALC system. Subsection 5.2.3 discusses the application of STPA and Event-B modelling techniques to the instantiated ALC template.

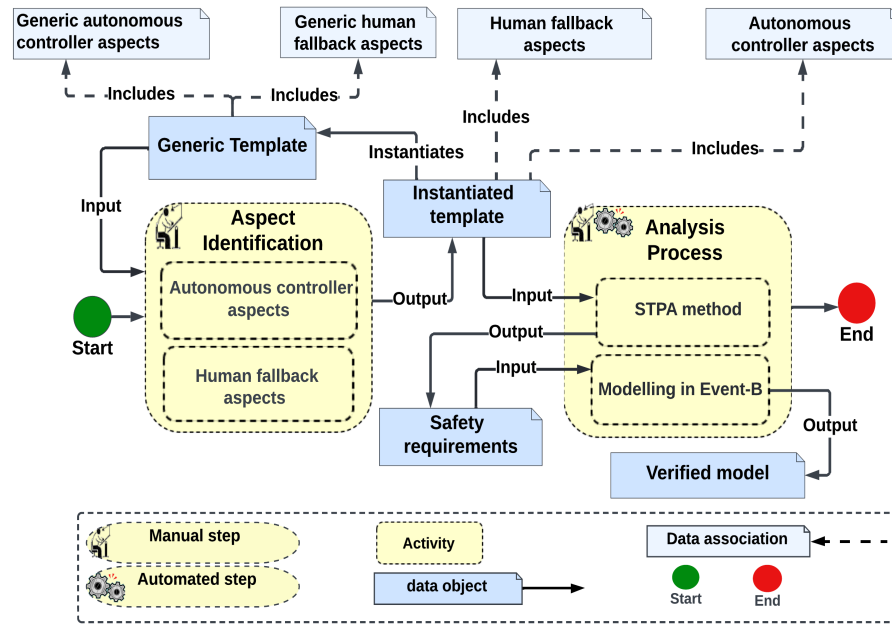


FIGURE 5.1: Overview of processes in RAT

5.2.1 Development of a Template

Due to the complexity of component interactions of semi-automation systems, we clarify the aspects of the SDV systems into two categories:

1. **Autonomous controller aspects**, determining the identification of a driving decision to actuate the SDV.
2. **Human fallback aspects**, establishing the engagement channel between the human fallback component and the autonomous controller.

In the following we present generic autonomous aspects of the SDVs then organise them into the internal components of an SDV.

Generic autonomous controller aspects: Table 5.1 shows the generic autonomous controller aspects of SDV systems. Various studies, such as the DARPA Challenges [128], Badue et al. [14] and OpenPilot [10], have shown that SDVs require three modules to perform the DDT: the perception, decision-making and control modules. As a result, our generic aspects of an autonomous controller are *sensing data, the perceived environmental features, driving decision* and *driving action* (see Figure 5.2).

The perception module is intended to perform the OEDR task by observing the driving environment via sensors. The perceived environmental features are considered as the

¹The notation in Figure 5.1 is the standard notation to describe work processes known as 'solution-patterns', (see <https://vvpatterns.ait.ac.at/about-vv-patterns/>).

output values of the perception module. These features aim to demonstrate how the autonomous controller achieves the OEDR task.

The decision-making module aims to accomplish the planning task, which involves a system’s goal such as keeping a vehicle between the lane lines. These kinds of goals cannot be accomplished without the help of a perception module; therefore the input values of a decision-making module mainly rely on the output values of a perception module. The driving decision can be considered as the output value of a decision-making module in order to show how an autonomous controller interprets the perceived environmental features and achieves a system’s goal.

Finally, the control module is responsible for accomplishing the lateral or longitudinal tasks that actuate the SDV in its environment. It receives the outputs of the decision-making module to specify modifications to low-level physical/electronic SDV components. For instance, the SDV ’s velocity become a signal to the accelerator pedal, while the SDV ’s orientation guides changes to steering wheel control.

TABLE 5.1: Generic autonomous controller aspects

Generic autonomous controller aspects	Description
Sensing data	Sensing data refers to information that is collected through sensors or other monitoring devices. Sensors are devices that detect critical properties of the environment.
Perceived environmental features	These features explain the ability of a system to detect and respond to any environmental events in a specific ODD.
Driving decision	This shows how an autonomous controller interprets the perceived environmental features and achieves a system’s goal.
Driving action	This indicates the required manipulation of the physical components of a vehicle. The vehicle control features can be represented as lateral and longitudinal features.

Generic human fallback aspects: Table 5.2 shows the generic human fallback aspects of SDV systems. Various automotive companies, such as Tesla [102, 56], Comma.ai [10] and Volvo [131], have shown that Human-In-The-Loop System (HITLS) integrates with the internal components of SDV to become a fallback option when a system fails to operate as expected. According to these companies, their autopilot software uses a Driver Monitoring System (DMS) to ensure the awareness level of a human driver in order either to activate the SDV systems or to warn the human driver about the SDV ’s current state. As a result, we find that the generic features of a human fallback driver are *the awareness level* and *the intervention request*. The awareness level aims to measure the awareness of the human driver and ensure a secure transition of the vehicle’s control,

while the intervention request denotes the need for the human driver to take control of the vehicle if the SDV system issues a request to intervene.

TABLE 5.2: Generic human fallback aspects

Generic human fallback aspects	Description
Awareness level	This feature indicates the status of a human driver when the SDV system is operating. The status of a human driver could be either aware or unaware.
Intervention request	This indicates that a system needs the human driver to intervene immediately.

Template for SDV systems: As a result of this phase the generic aspects of an autonomous controller and a human fallback driver were organised into the generic template prototype shown in Figure 5.2. This diagram contains the generic aspects of SDVs and how these aspects interact inside the internal components to accomplish the DDTs.

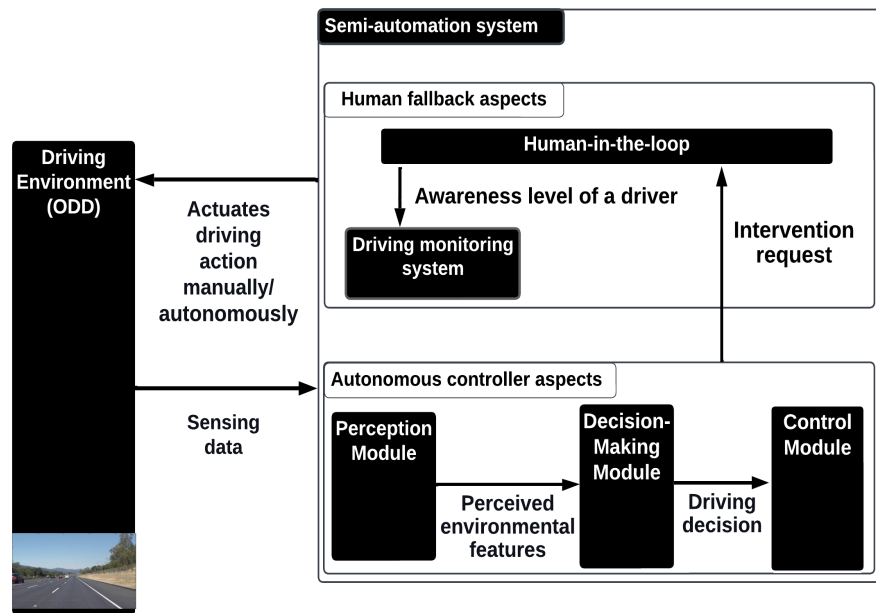


FIGURE 5.2: Template for SDV systems

5.2.2 Template Instantiation Process for ALC system

This subsection presents how the generic automation aspects of the SDV system can be used to instantiate the automation aspects of the ALC system discussed in Chapter 4.

Based on the general automation aspects of the SDV system, we categorised the instantiation process into two main categories: 1) *autonomous controller aspects* and 2) *human fallback aspects*. Figure 5.3 instantiates the generic autonomous aspects of an autonomous controller and a human fallback component into the ALC system.

First, the autonomous controller aspects focus on how the ALC system autonomously determines the speed and steering angle of an SDV to meet either Lane Keeping Assist (LKA) or Adaptive Cruise Control (ACC) functions. These autonomous aspects contain three primary components: perception; decision; and control. Table 5.3 illustrates the instantiation of these autonomous controller aspects, highlighting the advanced features of the LKA and ACC functions.

The sensing data is categorised into *camera images* and *radar reading points*, obtained respectively from the camera and radar sensors. The perception component receives the camera images and radar reading points to detect *the desired path*. In more detail, it estimates the identification of a desired path based on the interpretation of the images and radar reading points to detect lane markings and any potential leading vehicle. These interpretations are quantified by a confidence score (probabilities) that explicitly estimates the detection confidence of the ALC system in maintaining an SDV in its target lane. Therefore, the desired path, confidence score and any potential leading vehicle are treated as the perceived environmental features.

The decision component uses the desired path to define the required speed and steering angle modification to adjust an SDV onto a new (target) position between the lane lines. Hence, *target position* and *required modification of speed and steering angle* can be considered the driving decision that an SDV aims to achieve. Last, the control component outlines the driving action that manipulates the physical vehicle components to apply the modified speed and steering angle to reach the target position.

TABLE 5.3: Autonomous controller aspects.

Generic Aspects	ALC Specific Aspects
Sensing data	- Camera images - Radar reading points
Perceived environmental features	- Desired path - Confidence score - Leading vehicle
Driving decision	- Target position - Target speed - Target steering
Driving action	- Actuate the steering and speed autonomously

Second, the human fallback aspects aim to understand how the DMS component ensures the responsiveness of a human driver when the ALC system may request intervention. Table 5.4 extends the generic human fallback aspects to include the advanced features of the DMS.

The DMS can ensure the awareness level of a human driver via two monitoring features, active and sensitive. For instance, an active monitoring feature could require a driver to keep their hand on the steering wheel of the SDV while the ALC system is in operation. Additionally, a sensitive monitoring feature employs the in-car camera

to detect and monitor changes in the facial, eye and head movements of the human drivers of the SDV. Therefore, we extended the awareness level of a human driver to be either *hands on the steering wheel* or a *sensitive-monitoring feature*.

Furthermore, the ALC might alert a driver to assume control of an SDV. Therefore, we expanded the intervention request to include *warning messages*, *auditory notifications* and *corrective actions*. These messages might be sent when the ALC system requires driver correction. If the driver does not respond to these warning messages, the ALC may issue an auditory notification to inform the human driver that the ALC is to release control of the SDV.

TABLE 5.4: Human fallback aspects

Generic Aspects	DMS Specific Aspects
Awareness level	- Hands on steering wheel - Sensitive monitoring feature
Intervention request	- Warning message - Auditory notification - Corrective action

5.2.3 Analysis Process

The analysis process was carried out on the basis of the instantiated template of the ALC system. This phase was divided into two primary steps: the STPA method and modelling in Event-B. These steps are detailed in the subsequent subsections.

5.2.3.1 Systems Theoretic Process Analysis

The main reason for using STPA was to identify the potential hazards of a semi-automated system and to emphasise the safety requirements concerning the roles of a human driver and the ALC system. We employed the STPA guidelines provided by Leveson and Thomas [84]. These guidelines can be summarised into four steps as discussed in Subsection 3.1.4.2 of Chapter 3.

STEP 1: Define high-level losses, hazards and requirements

This step aims to identify the System Losses (SLs), System Hazards (SHs) and their corresponding requirements. With the help of the advanced of autonomous aspects between the autonomous controller and the human fallback component mentioned earlier in Figure 5.3, we defined the system as a semi-automation system that integrates with the human driver and driving environment. There are two main sub-systems: human fallback and autonomous controllers. The autonomous controller observes the

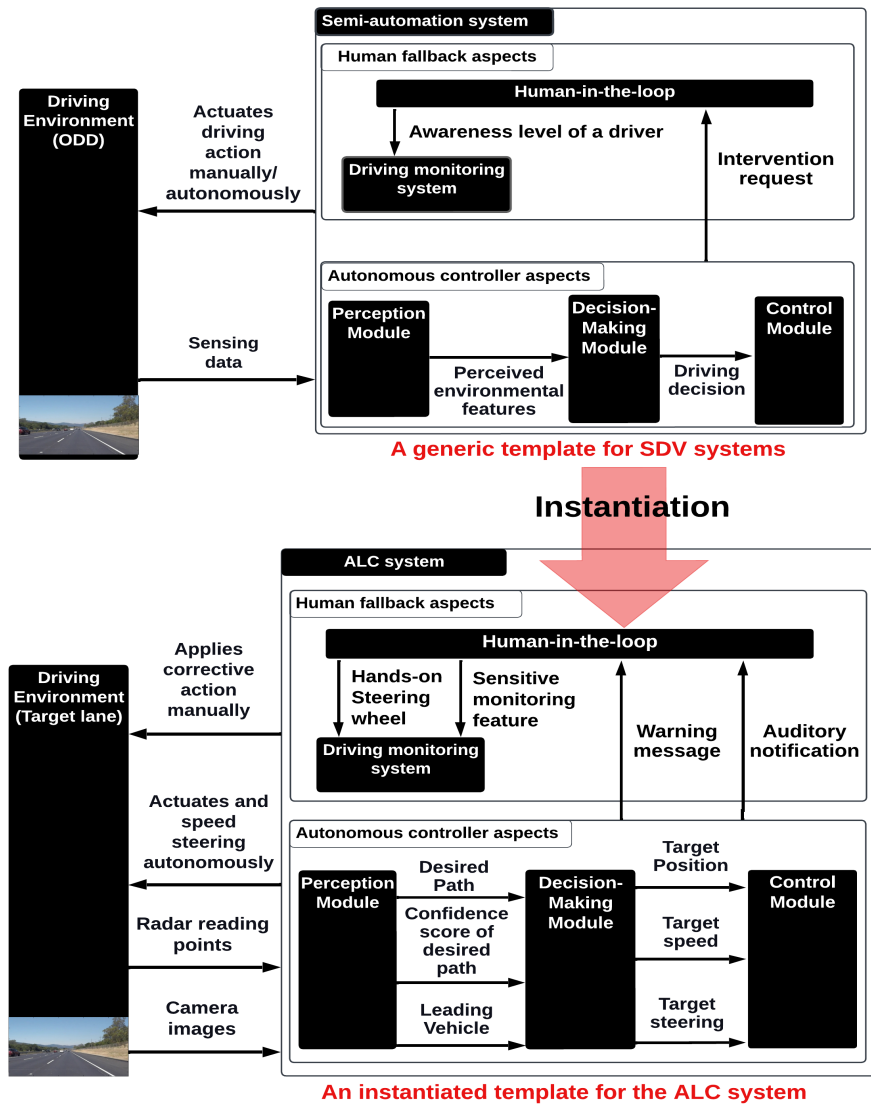


FIGURE 5.3: Instantiating process, from generic template to ALC concrete template

autonomous aspects outside the SDV, while the human fallback controller polices the changes inside the SDV. According to SAE [101], semi-automation systems assume that the driver provides a corrective action if the autonomous controller fails to operate as expected; therefore, a high-level system loss is SL1:

- **SL1:** *'The autonomous controller drives the SDV to collide with other obstacles outside its target lane'.*

As the human driver plays a fallback option in dealing with hazardous events, a high-level system hazard can be identified as SH1:

- **SH1:** *'The human driver is unaware of autonomous operations'.*

The initial Safety Requirements (SRs) could be identified as follows:

- **SR1:** The human fallback controller must compute/ensure the awareness level of the driver to activate the ALC system.
- **SR2:** If the driver does not provide hands on steering wheel, the human fallback warns and alerts the driver.
- **SR3:** If the system raises the alarm, the driver is to be responsible for performing the entire driving task.

STEP 2: Construct the hierarchical control structure

After identifying the system losses, hazards and initial requirements, the next step was to create the control structure. This step captures the functional relationships and interactions of the main system components as a set of Control Actions (CAs) and Feedback loops (Fs).

Our hierarchical control structure is shown in Figure 5.4. It comprises four layers: human driver (inside the SDV); human fallback controller; autonomous controller; and driving environment (outside the SDV). **CA1** and **F1** indicate the driver's responsibility to monitor the changes in the driving environment or intervene by taking control of the vehicle. **CA2** indicates the driver's responsibility to provide the hands on steering wheel to activate the system, while **F2** shows the possible system-level interactions between the human fallback controller and the human driver when the system sends a feedback loop to alert the driver. **CA3** and **F3** demonstrate the responsibility of the autonomous controller to perceive the driving environment and compute a target steering into a new position. **CA4** indicates a system interaction between the human fallback controller and the autonomous controller when the awareness level is computed and sent to the autonomous controller.

STEP3: Determine how new hazardous states may occur

This step aims to understand the dynamics of system behaviours in order to identify new hazardous events. We organised the Unsafe Control Actions (UCAs) analysis into three groups corresponding to **CA1** to **CA4**, as discussed earlier in Figure 5.4. The first group involves actuating the new steering variable (**CA3**). Table 5.5 illustrates UCAs associated with the actuating steering variable in the ALC. Specifically, the autonomous controller may propose a target steering angle in a manner that requires the physical wheel to exceed its steering angle limit. For instance, assuming the maximum range of the steering angle is 70 and the minimum range is -70, the autonomous controller

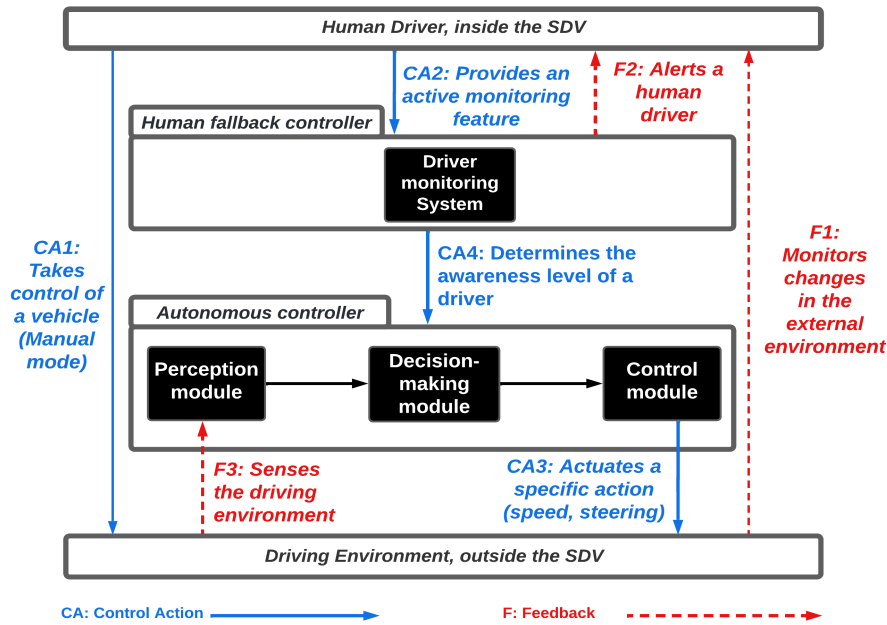


FIGURE 5.4: Control structure of high-level system interactions for an ALC system

must consider these specifications when proposing its target steering angle. Therefore, a new system hazard can be defined as SH2:

- **SH2:** ‘The autonomous controller may propose its target steering angle outside the steering angle range of the power steering system of the vehicle’.

TABLE 5.5: Unsafe control actions in steering variable of ALC system

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA3	UCA1: Autonomous controller does not consider a steering limit of the vehicle’s power steering system.	UCA2: Autonomous controller may propose a target steering angle that exceeds the physical wheel’s steering angle range.	N/A	N/A
Causal Factor				
Wrong adjustment of the actuating variables (steering).				

The second group is associated with human drivers to provide such an active monitoring feature (CA2) to activate the operation of the ALC system (CA3). Table 5.6 illustrates UCAs to the DMS operations. For example, the driver might be aware but not react when the ALC issues an intervention request. Therefore, a new system hazard can be defined as SH3:

- **SH3:** *'The human driver does not pay attention to the autonomous operations of an ALC system'.*

TABLE 5.6: Unsafe control actions in awareness level of driver

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2 and CA3	UCA3: DMS verifies(CA2) and shares(CA3) the awareness level of a driver with the ALC system, but a driver does not react to a request to intervene.	N/A	UCA4: Driver is aware, but does not react.	N/A
Causal Factor				
A poor reaction of a driver when the ALC issued a request to intervene.				

The third group aims to investigate the correction action from the human driver (CA1). In fact, the human driver assumes the responsibility to provide the correction action when the ALC system issues a request to intervene. Table 5.7 determines one UCA when the ALC system may release control of an SDV where the driver correction action is unknown. Therefore, a new system hazard can be defined as SH4:

- **SH4:** *'The autonomous controller may release control of an SDV when driver correction action is unknown'.*

TABLE 5.7: Unsafe control actions in driver correction action

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA1	N/A	N/A	N/A	UCA5: ALC system may release a control of an SDV, but the driver does not react.
Causal Factor				
Unknown reactions of a driver when an ALC system releases a control of an SDV				

STEP 4: Develop new requirements for the hazardous states identified in Step 3

This step aims to improve the proposed safety requirements based on the hazardous events identified in Step 3. For instance, the possible safety requirements to prevent system hazards (SH2 to SH4) are as follows:

- **SR4:** If the autonomous controller attempts to exceed the maximum or minimum steering angle, the human fallback controller warns and alerts the driver to take control of a vehicle [SH2].
- **SR5:** If the target steering angle exceeds the minimum steering angle, then the vehicle's power steering system modifies the target steering angle to the minimum steering angle [SH2].
- **SR6:** If the target steering angle exceeds the maximum steering angle, then the vehicle's power steering system modifies the target steering angle to the maximum steering angle [SH2].
- **SR7:** If the driver does not provide the hands-on steering wheel and a sensitive monitoring feature, the ALC system will be immediately deactivated [SH3].
- **SR8:** If the autonomous controller system issues a request to intervene (warning messages), the driver will be responsible for performing the entire driving task (manual driving) [SH3].
- **SR9:** When the ALC system issues an auditory notification to release control of an SDV, the driver is expected to assume control [SH4].

5.2.3.2 Modelling in Event-B

The driven safety requirements of the STPA (**SR1** to **SR9**) can be used as inputs of this stage. The Event-B models are created to prove that the occurrence of losses and their relevant hazards is eliminated or mitigated at system level. The Event-B elements (actions, guards and invariants) are discussed to show how the formal models in Event-B capture the safety requirements of the STPA. A full script of the formal models can be seen in Appendix C. The refinement strategy of our Event-B model can be divided into two categories: 1) *Autonomous controller aspects of the system*; and 2) *Human fallback aspects of the system*.

In the subsequent points we summarise some aspects of our formal models.

1) Formalisation of the autonomous controller features

The main goal of an autonomous controller is to move an SDV to be within its target lane, which corresponds to **CA3** and **F3**, as shown earlier in Figure 5.4. We used a constant, **Lane**, to specify all positions of vehicles in the lane, namely $\text{Lane} \subseteq \text{POSITION}$. The position of the vehicle is modelled as a variable **position**, with a safety invariant $\text{position} \in \text{Lane}$, namely the SDV most always be within the lane.

Three events model the various stages of the system (for autonomous controller aspects), namely, `perception`, `decision`, `control` and `environment` as in Figure 5.3. The relationship between the input and output of each stage is abstractly captured by constant functions. For example, the `decision` event is modelled as follows.

```

event decision
when
  @grd1 : stage = Decision
then
  @act1 : stage := Control
  @act2 : target_position := compute_target_position(desire_path)
  @act3 : target_steering_angle := compute_target_steering_angle(desire_path)
end

```

Here, `desire_path` is the output of the `perception` stage and is modelled as a set of positions, i.e., $\text{desire_path} \subseteq \text{POSITION}$. The constant functions `compute_target_position` and `compute_target_steering_angle` represent the computation made by the autonomous controller, and are defined accordingly

```

compute_target_position  $\in \mathbb{P}(\text{POSITION}) \rightarrow \text{POSITION}$ 
compute_target_steering_angle  $\in \mathbb{P}(\text{POSITION}) \rightarrow \text{STEERING\_ANGLE}$ 

```

The position of the vehicle is updated in `environment` event according to the current `position` and the `steering_angle`.

```

event environment
when
  @grd1 : stage = Environment
then
  @act1 : stage := Perception
  @act3 : position := move(position  $\mapsto$  steering_angle)
  @act2 : image := camera(move(position  $\mapsto$  steering_angle))
end

```

To ensure that `environment` maintains the safety of the system, i.e., $\text{position} \in \text{Lane}$, we added the following invariant.

$$\text{stage} = \text{Environment} \Rightarrow \text{move}(\text{position} \mapsto \text{steering_angle}) \in \text{Lane}$$

We introduced an additional stage between `control` and `environment` to allow possible interventions to correct the autonomous controller output. From the `Intervention` either the `steering_angle` is accepted or it needs to be corrected.

```

event accept
when
  @grd1 : stage = Intervention
  @grd2 : move(position  $\mapsto$ 
    steering_angle)  $\in \text{Lane}$ 
then
  @act1 : stage := Environment
end

event correct any angle when
  @grd1 : stage = Intervention
  @grd2 : angle  $\in \text{STEERING\_ANGLE}$ 
  @grd3 : move(position  $\mapsto$  angle)  $\in \text{Lane}$ 
then
  @act1 : stage := Environment
  @act2 : steering_angle := angle
end

```

2) Formalisation of the human fallback aspects

We modelled the human fallback aspects, namely `warning_message`, `auditory_notification`, `hands_on_steering_wheel`, `sensitive_monitoring_features_detected` using Booleans (**SR1**). In particular, we required that the `auditory_notification` is raised when the system is missing either awareness aspects (**SR2**).

$$\begin{aligned} \text{hands_on_steering_wheel} = \text{FALSE} &\Rightarrow \text{auditory_notification} = \text{TRUE} \\ \text{sensitive_monitoring_features_detected} = \text{FALSE} &\Rightarrow \text{auditory_notification} = \text{TRUE} \end{aligned}$$

An additional Boolean `manual_drive` was introduced to indicate that the human driver must be responsible (**SR9**) when the `auditory_notification` is on (**SR3**).

$$\text{auditory_notification} = \text{TRUE} \Rightarrow \text{manual_drive} = \text{TRUE}$$

Several events model the awareness detection (**SR1**), for example `hands_on_steering_wheel` is detected as followed.

<pre> event hands_on_wheel when @grd1 : hands_on_steering_wheel = FALSE then @act1 : hands_on_steering_wheel := TRUE @act2 : auditory_notification := bool(sensitive_monitoring_features_detected = FALSE) end </pre>	<pre> event hands_off_wheel when @grd1 : hands_on_steering_wheel = TRUE then @act1 : hands_on_steering_wheel := FALSE @act2 : auditory_notification := TRUE @act3 : manual_drive := TRUE end </pre>
--	---

The original event `accept` was refined as follows:

```

event accept refines accept
when
@grd1 : stage = Intervention
@grd2 : warning_message = FALSE
@grd3 : manual_drive = FALSE
then
@act1 : stage := Environment
end

```

That is, acceptance only occurs when there are no warning messages or when the system is in autonomous mode. This requires an additional invariant to ensure the consistency of the refinement.

$$\begin{aligned} \text{stage} = \text{Intervention} \wedge \text{warning_message} = \text{FALSE} \\ \Rightarrow \text{move}(\text{position} \mapsto \text{steering_angle}) \in \text{Lane} \end{aligned}$$

The event `correct` is extended to assume that a driver has responded when the ALC system may issue such intervention (**SR8**) as follows:

```

event correct extends correct
when
@grd4: manual_drive = TRUE  $\vee$  warning_message = TRUE
end

```

Table 5.8 details how the safety requirements (**SR1** to **SR9**) has been satisfied in a formal model. The majority of the proof obligations in these formalisations were verified either automatically, using Rodin provers, or with the assistance of additional external prover plug-ins such as SMT solvers.

TABLE 5.8: Safety requirements (**SR1** to **SR9**) in formal model.

Safety requirements	Event-B elements
SR1	Guards and actions in the detection of human monitoring features, e.g, action in the event <code>hands_on_wheel</code> : <code>@act1: hands_on_steering_wheel := TRUE</code>
SR2	An invariant, <code>hands_on_steering_wheel = FALSE \Rightarrow auditory_notification = TRUE</code>
SR3	An invariant, <code>auditory_notification = TRUE \Rightarrow manual_drive = TRUE</code>
SR4	Guard in the event <code>correct</code> , <code>@grd4: manual_drive = TRUE \vee warning_message = TRUE</code>
SR5 & SR6	Action in the event <code>correct</code> , <code>@act2: steering_angle := angle</code>
SR7	Actions in the detection of human monitoring features, e.g, action in the event <code>hands_off_wheel</code> : <code>@act3: manual_drive := TRUE</code>
SR8 & SR9	An invariant, <code>warning_message = TRUE \wedge auditory_notification = TRUE \Rightarrow manual_drive = TRUE</code>

5.3 Discussion

This chapter introduces the RAT framework as a response to **RQ2**. RAT comprises two main stages. The first stage, *aspect identification*, identifies the autonomous aspects of an SDV system and how it collaborates with human drivers. These aspects are then categorised into autonomous controller and human fallback aspects, which are organised into a template. This template elucidates how the SDV system and human driver cooperate to ensure safe autonomous operations. Specifically, the template shapes automation features by:

- Clearly defining and identifying the automation aspects between human drivers and SDVs during DDTs, particularly when a system relies on a human driver to guarantee safety.
- Establishing links between the automation aspects of DDTs and the internal components of SDV systems, specifying the responsibilities of human drivers or SDV systems during their performance.

In the subsequent *analysis stage*, STPA and Event-B methodologies are applied. STPA systematically identifies critical requirements that arise from interactions between the SDV system and human drivers. Event-B then rigorously captures these driven STPA requirements, ensuring consistency and enabling traceability with associated formal representations.

To validate the RAT framework, the template instantiated into a concrete example, such as the ALC system, illustrates how RAT is applied in practice. This demonstration provides a substantial understanding of how RAT effectively emphasises the responsibilities of the ALC system and human drivers during SDV autonomous operations.

5.4 Related Work

The safety properties of SDVs were derived from the concept that the autonomous controller is responsible for performing DDTs. The SAE [101] defines DDTs as containing various aspects, including lateral and longitudinal control, OEDR, human fallback driver and the ODD. In our approach, these features are incorporated into the template to illustrate how the human driver and SDV work together to achieve safe DDTs. Several studies, such as the DARPA Challenges [128], Zong et al. [143] and Lex Fridman [52], have highlighted that SDV systems rely on four main modules: perception; decision-making; control; and DMS. These modules are included in the template in order to establish the links between them and the SAE 's features

Traditionally, automotive designers have placed primary responsibility for safety on the human fallback driver [81]. International Organization for Standardization (ISO) 26262 (Road Vehicles – Functional Safety) [67, 72] supports this concept by designating the human fallback component as responsible for safety at system level. In addition, ISO has introduced another safety standard known as ISO 21448 (Road Vehicles – Safety of the Intended Functionality) [73], which aims to address safety hazards in a system even without component failures, namely a human fallback component does not act appropriately.

The ISO Standards are incorporated into the safety assessment approach, known as the National Highway Traffic Safety Administration (NHTSA) – Safety framework [8].

This framework is primarily designed for autonomous companies and has been used by Google [133] to prioritise safety in SDVs. According to the SAE technical paper presented by Rajiv [24] and the NHTSA [8], STPA can be integrated with ISO 26262 to identify other failures, such as inadequate component interactions, that may occur either with or without component failures. Therefore, implementing STPA in SDVs enables the analysis of failures that may arise even in the absence of component failure. Moreover, Colley and Butler introduced an innovative integration of STPA and Event-B [35]. This methodology leverages the Event-B formal method and its corresponding tool, Rodin [7], to employ modelling techniques. The driven STPA requirements are used as a framework to develop a rigorous model aiming proficiently to mitigate potential hazards and vulnerable system states. In our approach, the emphasis is on driven STPA requirements that guide modelling of the automation aspects of both autonomous and human fallback controllers.

5.5 Conclusion

In this chapter we presented an RAT approach for SDV systems. The generic template was organised into two main categories, autonomous controller and human fallback component, to define precisely the automation aspects of the internal components of SDV systems. The automation aspects of ALC system were presented to show how a generic template was instantiated in a concrete case study. The STPA was chosen to capture interactions in the high-level system components. The outcome of the STPA was the safety requirements that aim to prevent unwanted interactions between a human fallback and an autonomous controller. The driven STPA requirements were used to design formal models in Event-B. The creation of Event-B models helped to prove that unwanted interactions obtained from the STPA were eliminated or mitigated at system level.

In the following chapter a novel process embedded within the RAT approach is discussed. This process uses the proposed template to identify modelling patterns for SDV systems.

Chapter 6

Rigorous Analysis Template Process

This chapter describes our second contribution, entitled the *Rigorous Analysis Template Process (RATP)*. Parts of this chapter were published in COMPSAC 2023 [13]. Specifically, that material corresponds to the content in Section 6.1.

The Rigorous Analysis Template Process (RATP) extends the discussion on automation aspects for Self-Driving Vehicle (SDV) systems presented in Chapter 5. Specifically, It is crucial to recognise the limitations outlined in Chapter 5, including its non-iterative nature and inability to scale with the complexity of the system. These identified limitations are the inspirations behind the development of the RATP method.

RATP is an iterative approach that consists of five systematic steps. These steps are inspired by the Systems Theoretic Process Analysis (STPA) and Event-B methodologies. The key advantage of the RATP approach is to allow the system's behaviour to be analysed across varying levels of abstraction. The output of RATP is a set of safety requirements that guide the development of a rigorous model to maintain the system's safety against identified hazardous states at various levels of refinement.

In line with the previous chapter, the RATP approach is developed to address the main Research Question (RQ3.1) of this thesis.

RQ3.1 *How can a methodology be developed to systematically analyse the complexity of system safety from high-abstraction behaviours down to more detailed behaviours?*

RQ3.1 aims to address the complexity of the SDV system by decomposing the analysis into multiple analysis iterations. Each iteration examines the system's behaviours, starting from a high-level abstraction layer and progressively moving to a more detailed, concrete layer. The analysis layers, referred to as *modelling patterns*, are thoroughly explained in this chapter.

Section 6.1 provides an overview of the processes involved in the RATP approach. Section 6.2 establishes the modelling patterns for SDV systems. Section 6.3 explains how the RATP approach tackles RQ3.1. Section 6.4 discusses the advantages, limitations and related work associated with the RATP approach. Section 6.5 provides a conclusion to this chapter.

6.1 Systematic Analysis Steps of the RATP

The systematic process of the RATP approach is divided into five steps, as shown in Figure 6.1¹. These steps will be used to develop modelling patterns in Section 6.2. However, the focus of this section is dedicated to providing a more in-depth explanation of each step.

Moreover, the RATP approach involves an iterative process for re-performing the analysis (from Step 1 to Step 5). There are several benefits of such an iterative analysis methodology, as follows:

1. It allows the abstraction behaviours of a system to be analysed and modelled.
2. It supports a robust traceability of the system losses, hazards and their corresponding safety requirements into associated formal representations.
3. It gives the safety requirements a precise syntax where the consistency of the safety invariants can be formally verified.

Each step of the RATP approach is then further detailed in its own subsection.

6.1.1 Instantiating System Boundary Diagrams

This step aims to develop a series of hierarchical system boundary diagrams that support the development process at different levels of abstraction. The generic template of SDV systems is employed as an input for this step. The creation of this generic template was previously discussed in Section 5.2.1 of Chapter 5.

The idea for constructing such a boundary diagram is inspired by Step 1 in STPA. Leveson and Thomas [84, p. 17] recommend viewing the system as an abstraction representation based on the interactions involved in the main system components, which includes making decisions about what the system's components and boundaries are. For

¹The notation used in Figures 6.1, 6.5, 6.9, and 6.10 is the standard notation to describe work processes known as 'solution-patterns', (see <https://vvpatterns.ait.ac.at/about-vv-patterns/>).

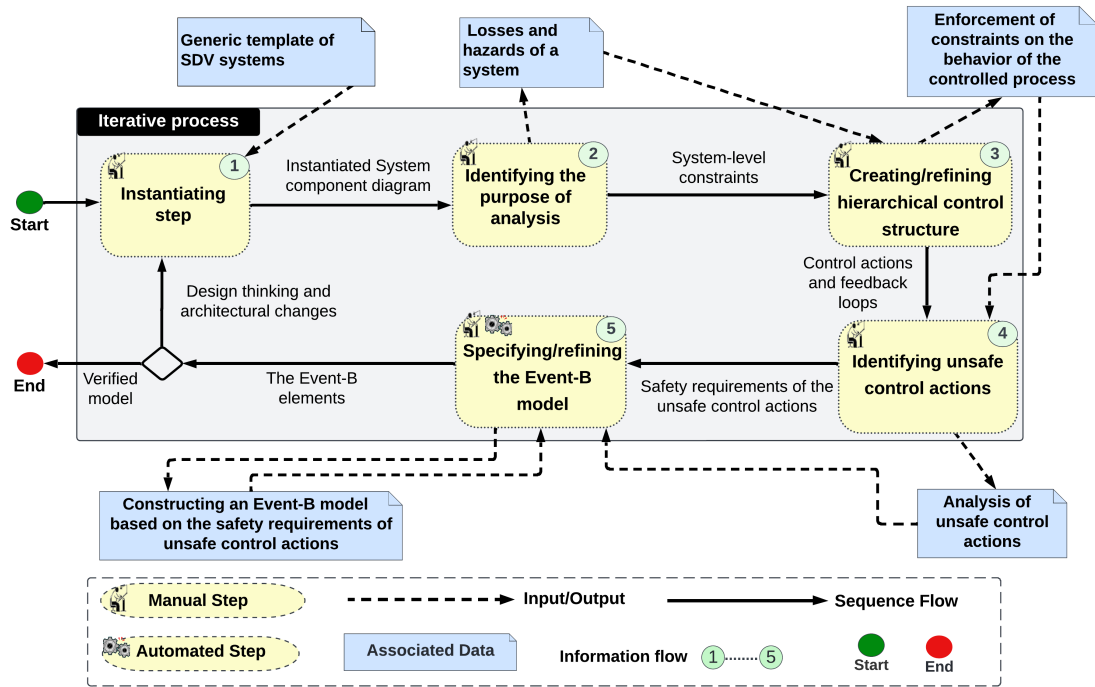


FIGURE 6.1: Systematic process of RATP approach

example, they suggested a diagram in Figure 6.2 to separate a system from its environment. Therefore, the system under analysis is defined based on its system boundaries, subsystems/components, and the interactions between those components.

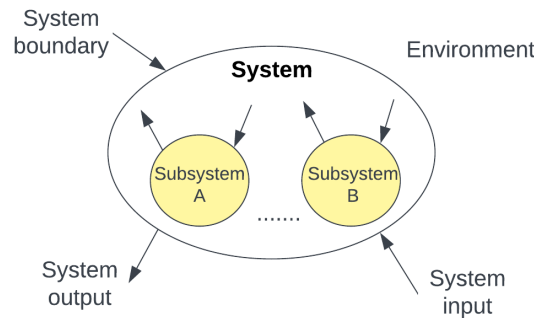


FIGURE 6.2: System boundary diagram, adapted from [84, p. 17].

However, in contrast to Leveson and Thomas, the system boundary diagram in the RATP approach is developed gradually through several iterations of the process, as shown in Figure 6.3. At the most abstract level, the system boundary diagram is constructed in Figure 6.3a based on how a system interacts with its environment. In further iterative processes, the system boundary diagram is refined in Figures 6.3b and 6.3c to add further details about the subcomponents involved in the main system and how those subcomponents explain the system component interactions. Therefore, the primary advantage of a system boundary diagram is defining a system under examination and guiding the instantiating process during the iterative analysis processes.

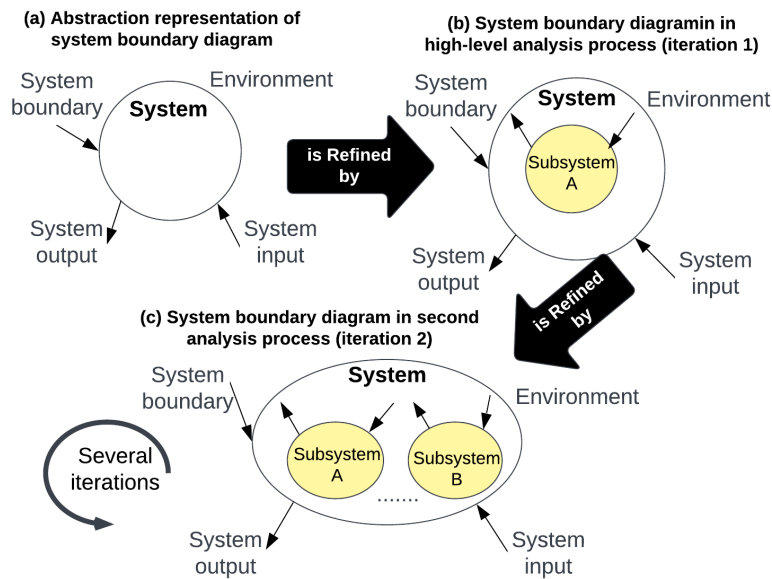


FIGURE 6.3: System boundary diagrams in RATP approach

6.1.2 Identifying the Purpose of Analysis

This step aims to identify the purpose of analysis in each iterative analysis process. Based on Step 1 in STPA, the goal of the safety engineering process is to mitigate or eliminate hazards in a system under investigation by specifying system conditions that must be satisfied to prevent hazards (system constraints).

The identification of system hazards and constraints is associated with System Losses (SLs), as illustrated in Figure 6.4. The SL may include human injury or death, but it also involves other unwanted events, such as financial, information, and equipment losses [86, p. 75]. The System Hazard (SH) is a system state illustrated by a system loss and is defined as ‘*environmental conditions leading to unwanted events*’. The Safety Constraint (SC) is a specific system condition considered to prevent or mitigate the SH. To identify SLs and their associated SHs and SCs, Leveson and Thomas [84, p. 17] suggest defining relationships based on the following three factors: *system*, *system boundary* and *environment*, as shown earlier in Figure 6.2.

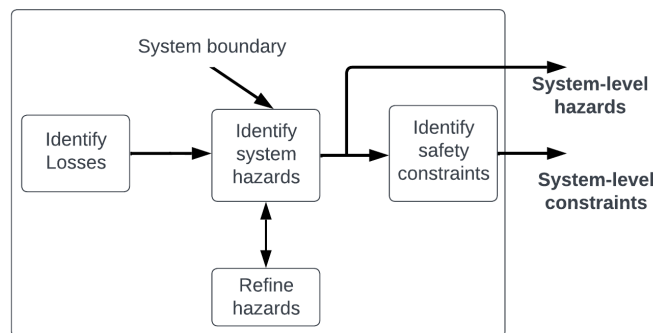


FIGURE 6.4: Overview of defining the purpose of analysis, from [84, p. 16]

In the RATP approach, this Step is similar to Step 1 in STPA. However, we introduce the system losses and hazards gradually through several iterations of the process, as shown in Figure 6.5. The identification of SLs and SHs can be illustrated based on how the system boundary diagram is constructed in the previous Step. Therefore, the SLs and their corresponding SHs and SCs are gradually identified based on how the system boundary and system components interact with their environment at different levels of abstraction.

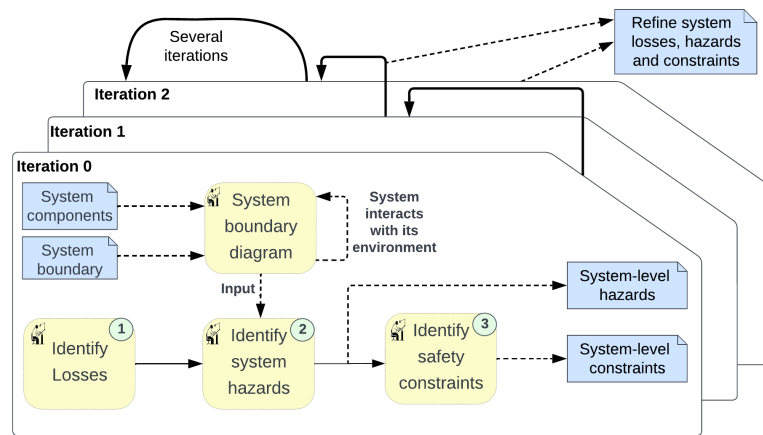


FIGURE 6.5: Overview of defining the purpose of analysis in RATP approach

6.1.3 Creating/Refining Hierarchical Control Structure

After identifying the SHs and their SCs, the control structure is constructed. A control structure is a system model that captures the functional relationships and interactions of the main system components as a set of Control Actions (CAs) and Feedback Loops (FLs). An effective control structure will enforce SCs on the behaviour of the overall system [84]. In Step 2 of STPA, the generic control structure is shown in Figure 6.6. In a broad sense, a control structure involves the following elements:

1. **Controllers:** Each controller in the control structure is responsible for ensuring that the system operates safely and effectively. In general, a controller is an entity that manages the behaviour of the system or its components. If the controller fails, it can lead to hazardous states where the SCs might be violated.
2. **Control Actions:** The CAs in STPA are essential to ensure the safety and reliability of complex systems. These actions are designed to mitigate, detect, or correct hazardous conditions. The actuators inside the controller are responsible for implementing these actions in order to enforce the SCs in the behaviour of the system or its component.

3. **Feedback:** In STPA, FLs ensure that the system output is constantly monitored and adjusted to keep it within safe states. These feedback loops detect and respond to the SHs by monitoring system outputs and adjusting system inputs.
4. **Inputs and outputs from the system components:** In STPA, the inputs and outputs of system components are analysed to identify potential hazards and determine appropriate CAs and FLs.
5. **Controlled processes:** In STPA, controlled processes refer to a process that has some form of control mechanism. This control mechanism can be a physical control, such as a safety interlock.

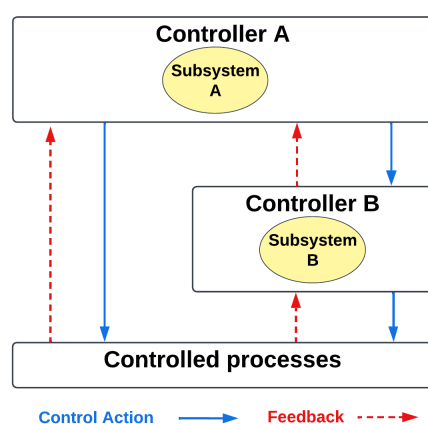


FIGURE 6.6: Simple form of control structure, adapted from [84, p. 24]

The RATP approach develops a control structure similar to Step 2 in STPA. However, it is constructed incrementally, taking into account the system components that were included in the system boundary diagram instantiated earlier (see Figure 6.3). Hence, the control structure is designed to encapsulate the interactions between system components across various levels of abstraction, as demonstrated in Figure 6.7.

6.1.4 Identifying Unsafe Control Actions

After constructing the control structure, the analysis of Unsafe Control Actions (UCAs) is performed in order to reveal the unsafe operations of the system under examination. According to Step 3 in STPA, the identification of UCAs could be summarised, as shown in Figure 6.8. To perform the analysis of UCAs, the control structure and SHs are inputs to this step. The UCAs are actions that could lead to SHs [84, p. 35]. New safety constraints/requirements would be proposed to mitigate or eliminate the identified UCAs.

According to [84], using the UCAs is an exploration technique that can be organised into the following categories:

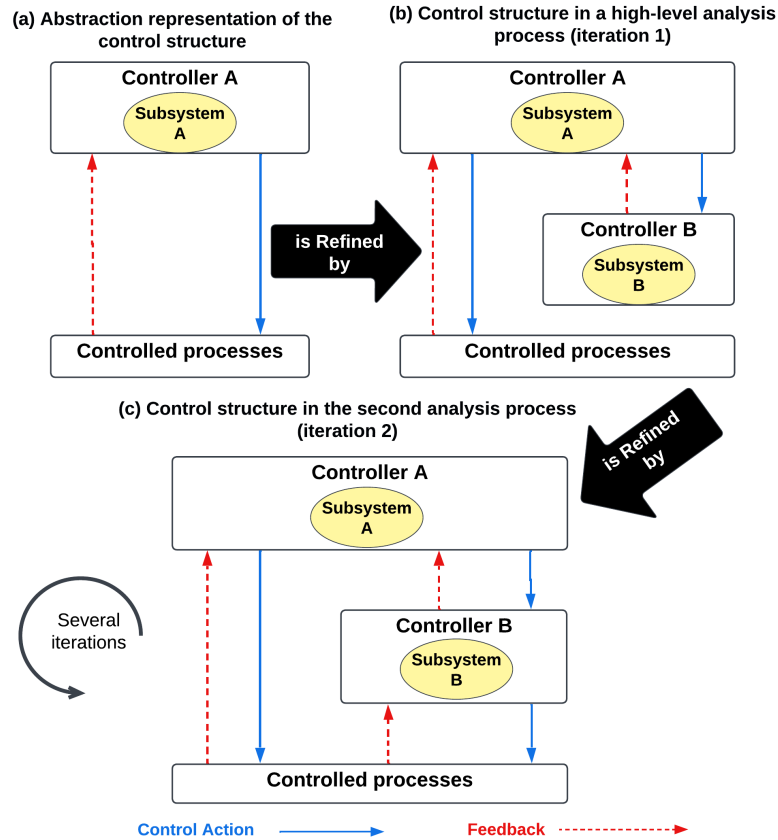


FIGURE 6.7: Overview of constructing the control structure in RATP approach

1. **Not providing CA leads to SH:** This indicates that a CA required for system safety is not provided in the control structure.
2. **Providing CA leads to SH:** This denotes that an UCA is provided in the control structure.
3. **Time of applying CA:** This investigates whether a safe CA required for system safety is provided at the wrong time or in the wrong order.
4. **Time of stopping CA:** This examines whether a safe CA required for system safety is applied too long or stopped too soon.

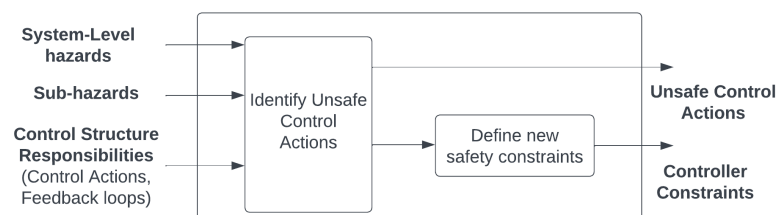


FIGURE 6.8: Overview of defining unsafe control actions, adapted from [84, p. 42].

Similar to Step 3 in STPA, we aim to identify the UCAs gradually through several iterations of the process, as shown in Figure 6.9. The identification of UCAs and their

Safety Requirements (SRs) is gradually driven on the basis of how the control structure is constructed in the previous step. The SRs can be defined as the functional requirements that mitigate the occurrence of SHs, where the causal factors or system actions that contribute to the SHs are demonstrated.

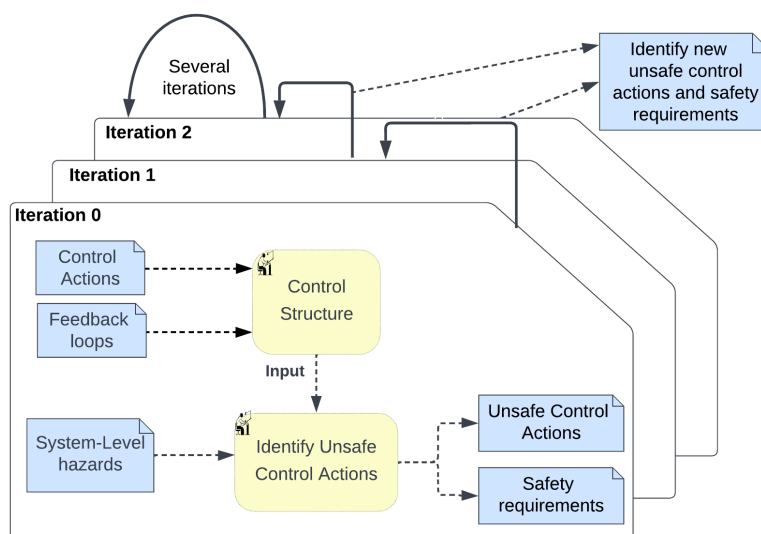


FIGURE 6.9: Overview of defining unsafe control actions in RATP approach

6.1.5 Specifying/Refining the Event-B Model

The goal of this step is to model the behaviours of a system, with a primary focus on ensuring the enforcement of SRs based on the UCAs identified in Step 4. In the analysis of iterative processes, formal models are gradually built, as shown in Figure 6.10. In order to specify a formal model, the control structure and the SRs are inputs to this step. In addition, the formal representations of SRs, as well as explanations of the assumptions involved in a formal model, are the outputs of this step.

A formal model is built using the Event-B language and its support tool, *Rodin platform* [7]. The guidance for developing a rigorous model in Event-B is summarised in the following steps:

1. Each iterative analysis process corresponds to one machine in Event-B.
2. Constraints within the design are to be modelled in a static part of a formal model through contexts.
3. The safety property of a system (the absence of hazards) is captured using invariants.
4. Each control action in the control structure corresponds to one or more variables.
5. The restrictions on control actions can be modelled on guards of events.

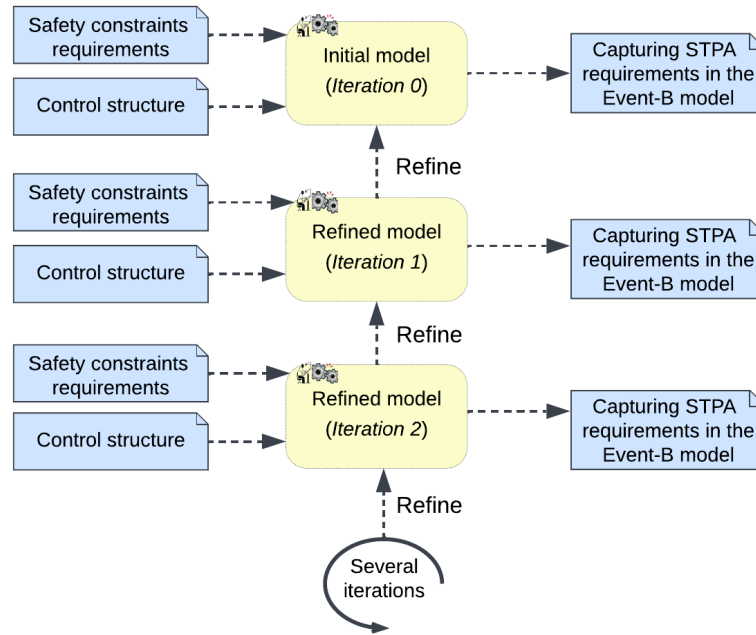


FIGURE 6.10: Overview of specifying formal models in RATP approach

6. The SRs are captured by invariants or guards within events.
7. The feedback loops can be modelled as combinations of invariants, constants, variables, actions and guards.

6.2 Modelling Patterns of SDV Systems

This section presents the development of modelling patterns for SDV systems. In a broad sense, a modelling pattern is a reusable solution that describes how to solve a commonly occurring problem in various situations [6]. By following the systematic steps of the RATP approach, the focus here is on developing a series of hierarchical patterns gradually, starting from a high-level analysis layer and moving towards a more detailed concrete layer. The aim of the analysis in each layer is as follows:

- **Layer 0:** Presents a high-level description of the interaction between the SDV system and its driving environment. Specifically, it covers how the constraints imposed by the driving environment can affect the movement of an SDV. This includes situations where the SDV needs to change its position to accommodate these restrictions.
- **Layer 1:** Shows in further detail how SDV systems modify their lateral and longitudinal variables to comply with environmental constraints and reach a new position within the driving environment.

- **Layer 2:** Models the autonomous operations of a system, including perception, decision-making, and control, to demonstrate how the semi-automated system autonomously identifies its lateral and longitudinal variables.
- **Layer 3:** Adds further details on how the awareness level of a driver can impact autonomous operations. It specifically discusses how SDV systems ensure that human drivers remain responsive when the system issues a request to intervene.

Each layer is then further detailed in its own subsection.

6.2.1 Abstraction Level (Layer 0)

Step 1. Instantiation: We start by defining the boundaries of what a semi-automated system is designed to do and what it is not. The Operational Design Domain (ODD) is an essential concept for automation levels because it specifies the conditions under which the SDV system must function [124, 101]. For example, the ODD for semi-automation levels (level 1 to level 3) is generally limited to specific driving scenarios, such as highway driving or parking. Therefore, we define the SDV system as a semi-automated system designed to move a physical vehicle (SDV) within a specific ODD. The system's boundary is determined by the limited ODD where the SDV system is expected to operate. Based on these definitions, the system boundary diagram is constructed in Figure 6.11.

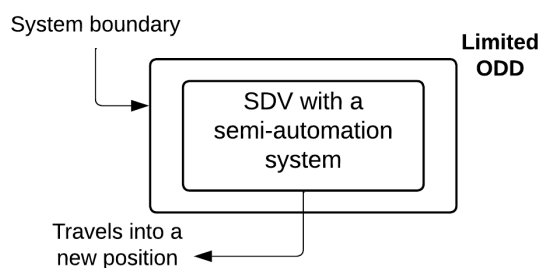


FIGURE 6.11: System boundary diagram in RATP Layer 0

Step 2. The purpose of analysis: Based on the previous system boundary diagram in Step 1, the System Loss (SL) is identified to ensure that the SDV only operates in a limited ODD. The main loss of a system (focusing on its purpose of moving the SDV in its limited ODD), **SL0**:

- **SL0:** The SDV collides with an object outside its ODD.

A high-level SH associated with **SL0** is **SH0**:

- **SH0**: The SDV travels into a position outside its ODD.

A high-level SC that satisfies the system conditions to prevent **SH0** is **SC0**:

- **SC0**: The SDV must always be located inside its limited ODD.

Step 3. Control structure: An abstract control structure is demonstrated in Figure 6.12. It involves two main components: 1) *the SDV with a semi-automation system* and 2) *the driving environment with a limited ODD*. The SDV with a semi-automation system presents a high-level view of a system designed to move the SDV within its limited ODD. The limited ODD is a driving environment that indicates a controlled process where the SDV can interact within its defined boundaries.

To capture a high-level interaction between an SDV and its limited ODD, **SC0** is translated as **CA1** and **F1**. For instance, **CA1** implies that an SDV may change its position into a new position, while **F1** indicates the monitoring of a change in an SDV's position.

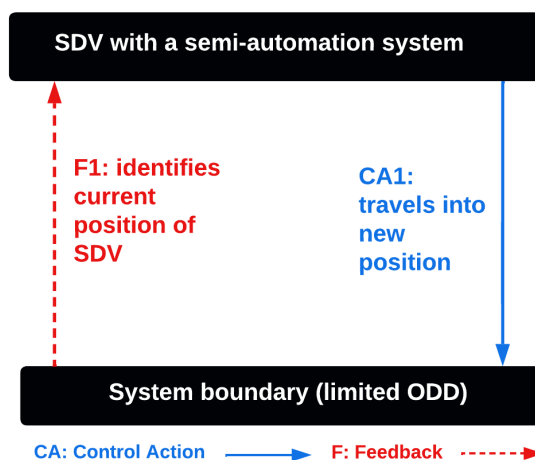


FIGURE 6.12: Control structure in RATP Layer 0

Step 4. Identifying unsafe control actions: The UCAs of **CA1** is demonstrated in Table 6.1. The *UCA1.1* represents the possibility of the system traveling to a new position outside the limited ODD. A contributing causal factor to **SH0** is the unsafe movement of a system. The identification of a safety requirement to mitigate its occurrence is:

- **SR0.1**: The SDV must travel into a new position inside the ODD [**SH0**].

TABLE 6.1: Unsafe control actions in RATP Layer 0

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA1	N/A	<i>UCA1.1</i> : The SDV may travel from a position inside the limited ODD to a new position outside that ODD.	N/A	N/A
Causal Factor				
Unsafe movement of an SDV				

TABLE 6.2: Control actions and feedback loops in formal model for RATP Layer 0

Control Action & Feedback loop	Event-B elements
CA1	Variable, <code>SDV_POSITION_env</code>
F1	Action, <code>SDV_POSITION_env := init_position</code>

Step 5. Event-B model: A full script of the initial machine (`m0`) can be seen in Section D.1 in Appendix D. Mapping is provided in Table 6.2 to demonstrate how **CA1** and **F1** link with the representation of an abstract model (`m0`).

At the abstraction level (`m0` machine), we use a constant `ODD` to specify the all positions of SDVs in the ODD, i.e., $ODD \subseteq POSITION$. The physical position of the SDV (**CA1**) is modelled as a variable `SDV_POSITION_env`, with a safety invariant $SDV_POSITION_env \in ODD$, i.e., the SDV must always be within the limited ODD (**SC0**). In addition, the initial position of an SDV (**F1**) is modelled as a constant `init_position`, where the initialisation position of SDV must be inside the limited ODD, i.e., $init_position \in ODD$.

The initial machine (`m0`) involves four events. Firstly, the `initialisation` event sets a valid default state of a physical position of the SDV as `SDV_POSITION_env := init_position`. Secondly, the `move` event abstractly captures a movement of an SDV into a new position as follows:

```

event move
any new_position
where
  //SR0.1: travel into a position inside
  ODD
  @grd1: new_position ∈ ODD
then
  //moved into a position inside ODD
  @act1: SDV_POSITION_env :=
    new_position
end

```

Along with the `move` event, we added two additional events: `system_on` and `system_off`. These events demonstrate the activation and deactivation of the autonomous functions of the system. Specifically, the `system_on` event indicates the initiation of autonomous functions, while the `system_off` event represents their deactivation.


```

event system_on
where
@grd1 : Sys_status = OFF
then
@act1 : Sys_status := ON
end

event system_off
where
@grd1 : Sys_status = ON
then
@act1 : Sys_status := OFF
@reset_pos : SDV.POSITION_env :=
    init_position
end

```

Table 6.3 shows how a safety requirement (**SR0.1**) has been captured in a formal model. At this level (machine **m0**), there is only one assumption (A) involved in a formal model, as **A0.1**:

- **A0.1**: The current position of an SDV starts inside the limited ODD.

TABLE 6.3: Safety requirement (**SR0.1**) in formal model for RATP layer 0

Safety requirements	Event-B elements
SR0.1	Guard (@grd1) in the <code>move</code> event, @grd1: <code>new_position</code> \in ODD

6.2.2 High-Level Analysis Process (Layer 1)

Step 1. Instantiation: A semi-automated driving system is a technology that can take control of some driving tasks, such as managing lateral and longitudinal variables. However, a human driver must be present and prepared to provide corrective action at any time. The lateral variables in a semi-automated vehicle may include the vehicle’s position within a lane and the necessary steering input to maintain the desired path. Meanwhile, longitudinal variables could include the vehicle’s speed, acceleration, and braking. These two variables together give a comprehensive view of the system’s behaviour, enabling it to move a SDV into a new position within its ODD.

To capture these high-level aspects, the system boundary diagram is refined in Figure 6.13. Specifically, the SDV with a semi-automated system is divided into two main system components: 1) *human fallback driver* and 2) *autonomous controller*. The human fallback driver can provide lateral and longitudinal variables manually, while the autonomous controller can apply them autonomously.

Step 2. The purpose of analysis: This step aims to refine system losses, hazards, and constraints based on the instantiated system boundary diagram in the previous Step. Our main focus is on investigating how the lateral and longitudinal variables can be modified to move an SDV into a new position within its limited ODD. Therefore, a

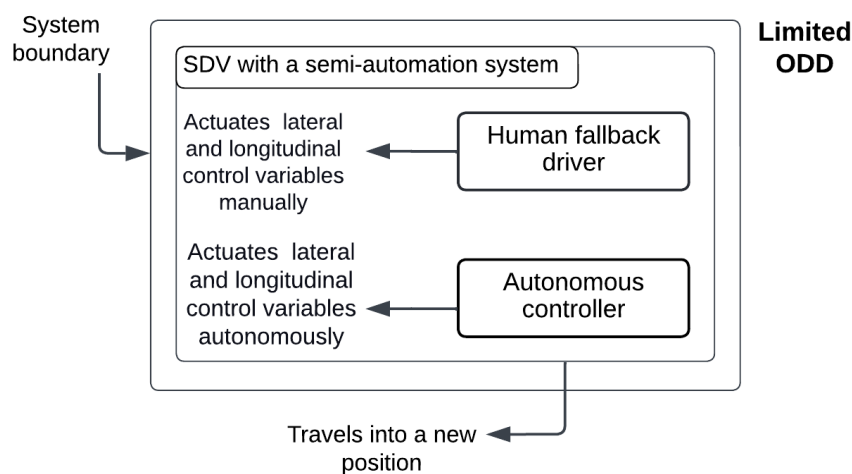


FIGURE 6.13: System boundary diagram in RATP Layer 1

refined System Loss (**SL1**) is driven from the previously identified safety requirement (**SR0.1**) as follows:

- **SL1**: A new lateral and longitudinal variable enables an SDV to reach a new position outside its ODD.

As the lateral and longitudinal variables may be modified by either a human driver or a semi-automated system, a new system hazard of **SL1** is identified as **SH1**:

- **SH1**: The lateral and longitudinal variables of an SDV are actuated by either a human driver or a semi-automated system to reach a new position outside the limited ODD.

A high-level safety constraint associated with **SH1** is defined as **SC1**:

- **SC1**: The lateral and longitudinal variables of an SDV must be modified to reach a new position inside the ODD.

Step 3. Refining control structure: We have refined the control structure to demonstrate how both the human driver and the semi-automated system can potentially modify the lateral and longitudinal variables of an SDV, as illustrated in Figure 6.14. To identify the lateral and longitudinal variables responsibly, the SDV with a semi-automated system splits into two main controllers: *the human fallback driver* and *the autonomous controller*. New Control Actions (**CAs**) and Feedback Loops (**Fs**) are added based on the **SC1**. For instance, **CA2** sets the lateral and longitudinal variables of an

SDV autonomously, while **F2** monitors any changes in the current lateral and longitudinal variables. As a human driver might be responsible for setting new lateral and longitudinal variables of an SDV, **CA3** indicates a high-level interaction between the human driver and the limited ODD. The driver may provide corrective action if the semi-automated system fails to keep the SDV within its limited ODD.

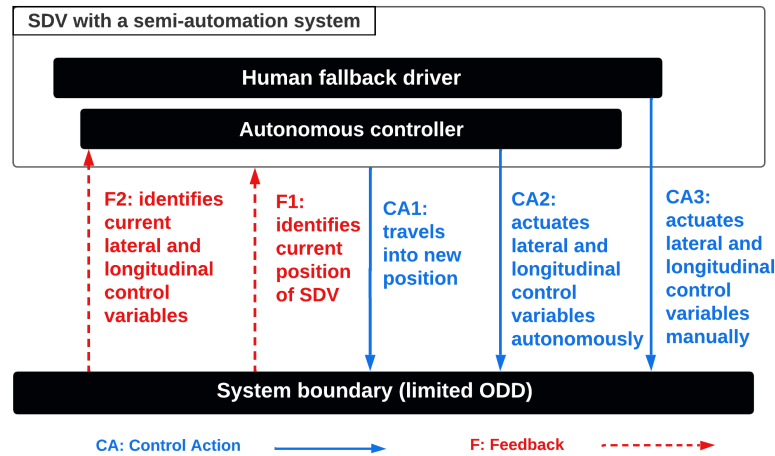


FIGURE 6.14: Control structure in RATP Layer 1

Step 4. Identifying unsafe control actions: The identification of Safety Requirements (SRs) is associated with the Unsafe Control Actions (UCAs) of **CA2** and **CA3**, as shown in Table 6.4. These UCAs explain a causal factor that leads to the **SH1**. For instance, *UCA2.2* and *UCA2.3* indicate that the lateral and longitudinal variables of an SDV can be modified to reach a position outside the limited ODD while the driver’s corrective action is missing. Therefore, the new safety requirements associated with **SH1** are as follows:

- **SR1.1:** the semi-automated system must actuate a lateral and longitudinal variable autonomously to reach a new position inside the limited ODD [**SH1**].
- **SR1.2:** the human driver can provide corrective action that actuates a lateral and longitudinal variable manually to reach a new position inside the limited ODD [**SH1**].

Step 5. Refining Event-B model: A full script of the refined machine (**m1**) can be seen in Section D.2 in Appendix D. Mapping is provided in Table 6.5 to detail how **CA2**, **CA3** and **F2** are linked to the representation of a refined model (**m1**).

At this refinement level (**m1** machine), we modelled the physical lateral variable of an SDV as a variable called **LATERAL_VAR**. This variable has two constants, **max_lat** and **min_lat**, that specify the defined range of lateral variables. Similarly, we modelled the

TABLE 6.4: Unsafe control actions in RATP Layer 1

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2 and CA3	UCA 2.1: Semi-automation system moves a vehicle into a new position inside the ODD (CA1), but the specifications of lateral and longitudinal control variables are missing.	UCA 2.2: actuates lateral and longitudinal control variables autonomously (CA2) to reach a position outside the ODD.	UCA 2.3: driver's corrective action is missing (CA3) when an SDV moves outside the ODD.	N/A
Causal Factor				
Wrong adjustment of the control variables (lateral and longitudinal)				

TABLE 6.5: Control actions and feedback loops in formal model for RATP Layer 1

Control Action & Feedback loop	Event-Block elements
CA2	Variables, LATERAL_VAR and LONGITUDINAL_VAR
CA3	Variables, LATERAL_VAR and LONGITUDINAL_VAR
F2	Actions in initialisation event, LATERAL_VAR :∈ LATERAL and LONGITUDINAL_VAR :∈ LONGITUDINAL

physical longitudinal variable of an SDV as a variable called `LONGITUDINAL_VAR`, which also has two constants, `max_lon` and `min_lon`, that specify the defined range of longitudinal variables. The movement of an SDV into a new position is modelled as a constant `move`, where modifications of the lateral and longitudinal variables enable the SDV to reach new positions. The definition of the `move` function is shown below.

$$\text{move} \in \text{POSITION} \times \text{LONGITUDINAL} \times \text{LATERAL} \rightarrow \mathbb{P}_1(\text{POSITION})$$

Three events model the high-level cases of the system at this layer, i.e., `auto_actuating` (CA2), `manual_actuating` (CA3), `move` (CA1) as shown earlier in Figure 6.14. The `auto_actuating` event abstractly specifies an autonomous identification of a new lateral and longitudinal variable to reach a new position inside the limited ODD (SR1.1).

```

event auto_actuating
any auto_lat auto_lon
where
/* System status is ON*/
@grd1 : Sys_status = ON
/* lateral definition */
@grd2 : auto_lat ∈ LATERAL
/* longitudinal definition */
@grd3 : auto_lon ∈ LONGITUDINAL
/*(SR1.1) movement leads to ODD*/
@grd4 : move(SDV_POSITION.env ↦
auto_lon ↦ auto_lat) ⊆ ODD
then
@act1 : LATERAL_VAR := auto_lat
@act2 : LONGITUDINAL_VAR :=
auto_lon
end

```

In the same way, the `manual_actuating` event describes how a human driver may provide a new lateral and longitudinal variable to reach a new position inside the limited ODD (SR1.2).

```

event manual_actuating
any
manual_lat manual_lon
where
@grd1 : manual_lat ∈ LATERAL
@grd2 : manual_lon ∈ LONGITUDINAL
//SR1.2: modification leads to ODD
@grd3 : move(SDV_POSITION_env ↦
manual_lon ↦ manual_lat) ⊆ ODD
then
@act1 : LATERAL_VAR := manual_lat
@act2 : LONGITUDINAL_VAR :=
manual_lon
end

```

Finally, the `move` event triggers either a manual or an autonomous adjustment of the SDV's lateral and longitudinal position, enabling it to relocate to a new position within the limited ODD (SC1).

```

event move extends move
where
/* new (target) position must be within set of position inside the ODD*/
@grd2 : new_position ∈ move(SDV_POSITION_env ↦ LONGITUDINAL_VAR ↦ LATERAL_VAR)
end

```

Table 6.6 shows how the safety requirements (SR1.1 and SR1.2) have been captured in a formal model. At this level (machine `m1`), there is a new assumption (A) involved in a formal model, as A1.1:

- **A1.1:** The human driver provides lateral and longitudinal variables manually to keep an SDV inside the limited ODD.

TABLE 6.6: Safety requirements(SR1.1 and SR1.2) in formal model for RATP Layer 1

Safety requirements	Event-B elements
SR1.1	Guard (@grd4) in the <code>auto_actuating</code> event, @grd4: <code>move(SDV_POSITION_env ↦ auto_lon ↦ auto_lat) ⊆ ODD</code>
SR1.2	Guard (@grd3) in the <code>manual_actuating</code> event, @grd3: <code>move(SDV_POSITION_env ↦ manual_lon ↦ manual_lat) ⊆ ODD</code>

6.2.3 Second Analysis Process (Layer 2)

Step 1. Instantiation: Based on the generic system component diagram (mentioned earlier in Figure 5.2), the SDVs include three main components of perception, decision-making and control for performing the autonomous operations. Therefore, we assume that these operations are accomplished by the autonomous controller inside the SDV with a semi-automated system. In Figure 6.15, we refine a previous system boundary

diagram (mentioned earlier in Figure 6.13) to capture the autonomous operations of the SDV with a semi-automated system.

SDVs use advanced sensor tools for the critical Object and Event Detection and Response (OEDR) task, allowing the vehicle to perceive and navigate its limited ODD safely. The perception component analyses the sensing data to identify the environmental features and predict their detection scores. The decision-making component uses the detected environmental features to determine appropriate driving actions, while the control component transforms these decisions into physical vehicle movements, using actuators connected to the SDV's control systems.

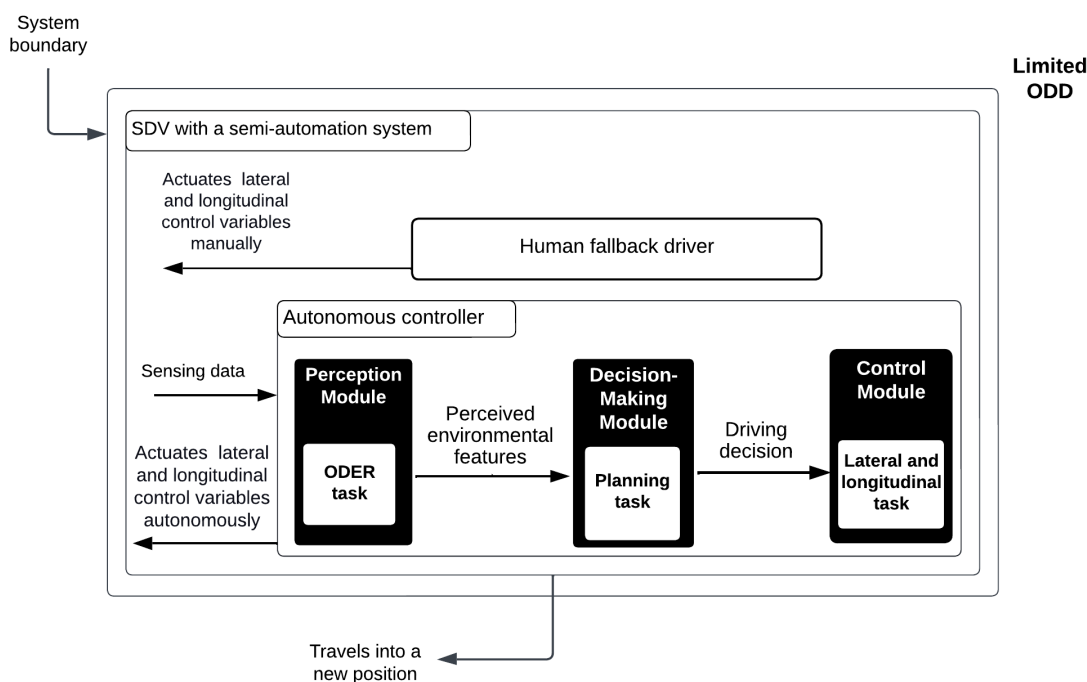


FIGURE 6.15: System boundary diagram in RATP Layer 2

Step 2. The purpose of analysis: The purpose of the analysis is to study how the semi-automated system accomplishes its autonomous operations. Based on the system boundary diagram (mentioned earlier in Figure 6.15), a refined System Loss (SL2) can be derived from the previously identified safety requirement **SR1.1** as follows:

- **SL2:** The autonomous controller autonomously actuates lateral and longitudinal variables to reach a new position outside the limited ODD.

As the autonomous controller of a semi-automated system involves three main components (perception, decision-making module, and control), various system hazards can be identified to ensure the safe identification of lateral and longitudinal variables. These System Hazards (SHs) are as follows:

- **SH2.1:** The perception component identifies the perceived environmental features that may locate the SDV outside the limited ODD.
- **SH2.2:** The decision-making component identifies a target lateral and longitudinal variable to reach a new position inside the limited ODD.
- **SH2.3:** The control component may actuate a target lateral and longitudinal variable that violates the physical actuator component involved in the SDV systems.

A high-level safety constraint associated with these system hazards is **SC2**:

- **SC2:** The semi-automated system must perform its autonomous operations to keep an SDV inside its limited ODD.

Step 3. Control structure: A refined control structure is constructed through a semi-automated system that accomplishes its autonomous operations, as shown in Figure 6.16. The autonomous controller is divided into three components of perception, decision-making and control. The interactions (input/output) between these components are captured by using a set of Control Actions (**CAs**) and Feedback loops (**Fs**).

In the perception component, **F3** indicates that the semi-automated system must obtain incoming sensing data to identify perceived environmental features (**CA4**). **F4** covers a feedback loop between the perception and decision components when the semi-automated system may reason about identifying perceived environmental features.

In the decision component, **CA5** implies the responsibility of a semi-automated system to identify a target (new) position of an SDV. Additionally, **CA6** indicates the responsibility of a semi-automated system to identify the required change of lateral and longitudinal variables to reach that new position with the limited ODD. **F5** covers a feedback loop between the decision and control components when the control component may accept/reject, actuating a new lateral and longitudinal variable (**CA2**) leading to that new position (**CA1**).

Step 4. Identifying unsafe control actions: At this layer, UCAs of autonomous operations are demonstrated in three categories. The first category involves unsafe operations in the perception component (**SH2.1**). These UCAs are presented in Table 6.7 and explained as follows:

- The autonomous controller may identify environmental features without sensing the limited ODD.
- The identified perceived environmental features might not match the current position of an SDV.

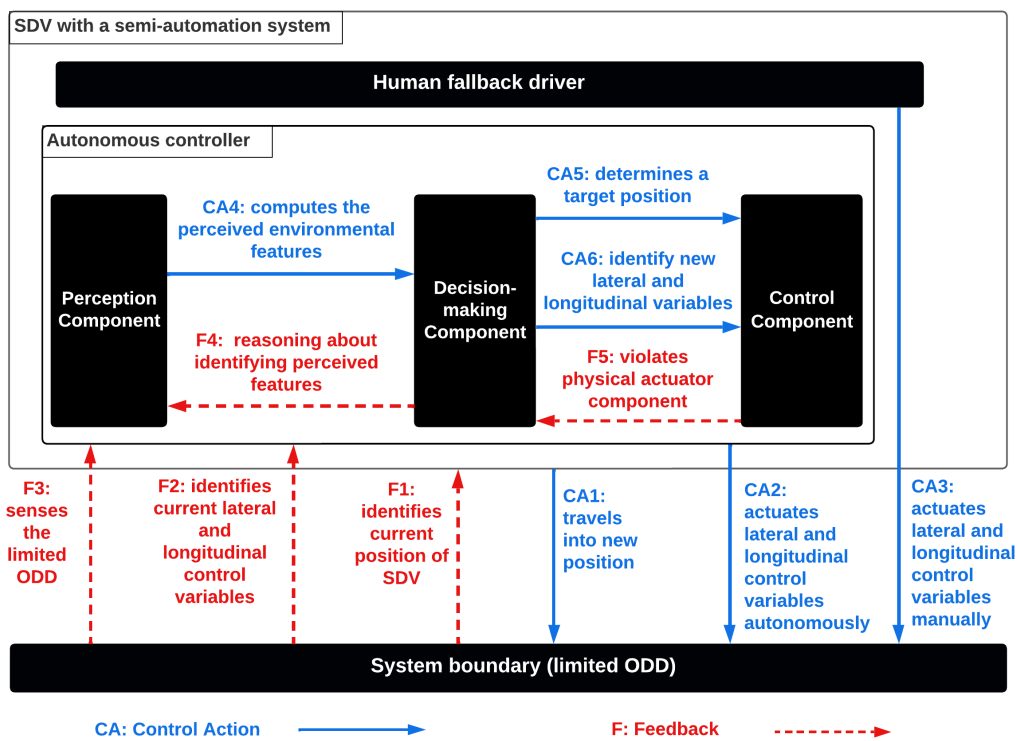


FIGURE 6.16: Control structure in RATP Layer 2

- The sensing data used to identify the perceived environmental features may be incorrect.

TABLE 6.7: Unsafe control actions of perception component in RATP Layer 2

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA4	UCA3.1: autonomous controller may identify the perceived environmental features without sensing the limited ODD (F3).	UCA3.2: Perceived environmental features do not update according to the SDV's position (F1).	UCA3.3: Perceived environmental features identified according to wrong/late sensing data (F4).	N/A
Causal Factor				
Failure in the identification of the perceived environmental features.				

Therefore, the potential safety requirements associated with the perception component can be identified as follows:

- **SR2.1:** The perception component must update the sensing data according to the current position of an SDV [SH2.1].

- **SR2.2:** The perception component must use sensing data to identify the perceived environmental features for keeping an SDV inside the ODD [SH2.1].
- **SR2.3:** The perception component can rely on the driver correction if the perceived environmental features do not keep an SDV inside the ODD [SH2.1].

The second category involves the unsafe operations associated with the decision component (SH2.2). These UCAs are presented in Table 6.8 and are explained below:

- The autonomous controller may identify a target position based on incorrectly perceived environmental features, which require the SDV to move to a new position outside the limited ODD.
- The target lateral and longitudinal variables lead to a new position outside the limited ODD.

TABLE 6.8: Unsafe control actions of decision component in RATP Layer 2

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA5 and CA6	UCA3.4: autonomous controller may identify the perceived environmental features (CA4) that required an SDV to move into a new position (CA5) outside the limited ODD.	UCA3.5: autonomous controller may identify a new lateral and longitudinal (CA6) that proposes for a new position (CA5) outside the limited ODD.	N/A	N/A
Causal Factor				
Unsafe identification of target position and lateral/longitudinal variables.				

Therefore, the potential safety requirements associated with the decision component can be identified as follows:

- **SR2.4:** The decision-making component must identify a target (new) position based on the accurately identified environmental features [SH2.2].
- **SR2.5:** The decision-making component must determine the required change in lateral and longitudinal variables based on the identified (new) position [SH2.2].
- **SR2.6:** The decision-making component can rely on the driver correction if a target position cannot keep an SDV inside the limited ODD [SH2.2].

The third category concerns the unsafe operations associated with the control component (SH2.3). These UCAs are presented in Table 6.9 and explained below.

- The lateral and longitudinal variables cannot be physically applied to the actuator component involved in the SDV systems.
- The target lateral and longitudinal variables exceed the actuator component limits involved in the SDV systems.

TABLE 6.9: Unsafe control actions of control component in RATP Layer 2

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2	UCA3.6: autonomous controller cannot actuate a new lateral and longitudinal variable (CA6).	UCA3.5: autonomous controller proposes new lateral and longitudinal variables (CA2) that may violate a physical actuator component involved in the SDV systems (F5).	N/A	N/A
Causal Factor				
Exceeding the actuator component limits for applying lateral and longitudinal variables.				

As a result, the potential safety requirements related to the control component can be identified as follows:

- **SR2.7:** The control component must only activate the lateral and longitudinal variables within the limits specified by the actuator component, such as the maximum and minimum lateral and longitudinal adjustments [SH2.3].
- **SR2.8:** The control component can rely on the driver correction if the target lateral and longitudinal variables violate the actuator component limits in the SDV system [SH2.3].

Step 5. Event-B model: Full details of the refined machine (m2) can be seen in Section D.3 in Appendix D. Mapping is provided in Table 6.10 to demonstrate how new control actions (CA4, CA5, CA6) and feedback loops (F3, F4, F5) link with the representation of a refined machine (m2).

A constant function `sensor` is defined to capture the sensing data in multiple positions, i.e., `sensor ∈ POSITION → SENSING.DATA`. An important specification of an SDV is demonstrated in a consistency invariant `sensing_data_env = sensor (SDV_POSITION_env)`, i.e., the SDV system must receive incoming sensing observations (F3) based on the physical position of the SDV (SR2.1).

TABLE 6.10: Control actions and feedback loops in formal model for RATP Layer 2

Control Action & Feedback loop	Event-B elements
CA4	Variables, <code>generic_features</code> , <code>generic_path</code> and <code>accuracy</code> .
CA5	Variable, <code>selectPos</code> .
CA6	Variables, <code>selectLat</code> and <code>selectLon</code> .
F3	Variable, <code>sensing_data_env</code> .
F4	Guard <code>grd3</code> in <code>lowAccuracy</code> event, <code>accuracy</code> < 80.
F5	Guard <code>grd3</code> in <code>violationLimit</code> event, <code>selectLat</code> \notin LATERAL \vee <code>selectLon</code> \notin LONGITUDINAL.

The perception component is a crucial part of the semi-automated system that uses sensing data from external sensor systems to identify environmental features. Therefore, we represent the functionality of the perception component by specifying two constant functions: `Identify_features` and `OEDR_task`, which are defined below.

```
//function shows the expected results of a received sensing data
@typeof-features: Identify_features  $\in$  (SENSING_DATA  $\times$  PERCEIVED_FEATURES)  $\rightarrow$ 
Accuracy
//function recognises set of positions (path) based on the perceived features
@typeof-path_recognition: OEDR_task  $\in$  (SENSING_DATA  $\times$  PERCEIVED_FEATURES  $\times$ 
Accuracy)  $\rightarrow$   $\mathbb{P}$ (POSITION)
```

The decision-making component is responsible for analysing the information gathered by the perception component and making decisions based on that information. Therefore, we specify three castanet functions: `specify_target_position`, `compute_lateral` and `compute_longitudinal`, which are defined below.

```
//function specifies a target position based on a path obtained from OEDR
@typeof-compute_target_position: specify_target_position  $\in$   $\mathbb{P}$ (POSITION)  $\rightarrow$  POSITION
//function computes the lateral variable to reach a specific (target) position
@typeof-compute_lateral: compute_lateral  $\in$  POSITION  $\times$  LATERAL  $\rightarrow$  LATERAL
//function computes the longitudinal variable to reach a specific (target) position
@typeof-compute_LONGITUDINAL: compute_longitudinal  $\in$  POSITION  $\times$  LONGITUDINAL
 $\rightarrow$  LONGITUDINAL
```

The control component is responsible for translating the driving decisions generated by the decision component into low-level control signals that can effectively actuate the vehicle's actuators to achieve a new position. Therefore, we have specified a constant function `move` to represent the actuation task of the SDV, which is defined as follows:

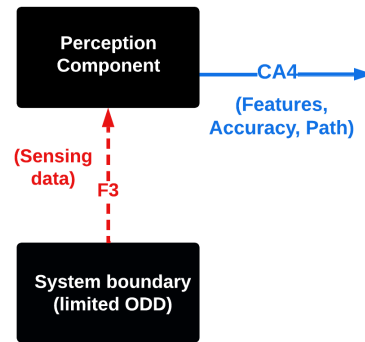
```
//function actuates lateral/longitudinal variables into new positions
@typeof-m: move  $\in$  POSITION  $\times$  LONGITUDINAL  $\times$  LATERAL  $\rightarrow$   $\mathbb{P}_1$ (POSITION)
```

Furthermore, the high-level operations of a semi-automated system are organised into five stages: `Perception`, `Decision`, `Control`, `Intervention` and `AutonomousDriving`. These stages are defined within the variable `stage`, and the gluing invariant `glu_inv1` is added to ensure consistency between the semi-automated system's status and these stages. The invariant is defined as follows:

```
//stages when a system is performing its autonomous operations
@gluing_inv1 : Sys_status = ON ⇒ stage ∈ {Perception, Decision, Control, Intervention,
AutonomousDriving}
```

The stages of a semi-automated system are modelled into five sequential categories. The first category is the perception stage, which abstractly identifies the environmental perceived features (CA4) based on the sensing data (F3). The event `perception` is modelled as follows:

```
event perception
any gf when
@grd1 : gf ∈ PERCEIVED_FEATURES
@grd2 : Sys_status = ON
@grd4 : stage = Perception
then
//sensing data based on SDV's position
@act1 : sensing_data_env := sensor (SDV_POSITION_env)
//features to be detected in the sensing data
@act2 : generic_features := gf
/* accuracy of the detection results */
@act4 : accuracy := Identify_features(sensing_data_env
↳ generic_features)
/* recognise the path (set of position)*/
@act5 : generic_path := OEDR_task(sensing_data_env ↳
features ↳ accuracy)
/* change a stage of system to be in Decision */
@act6 : stage := Decision
end
```



Based on event perception, we have added two invariants: `detection_task` and `recognition_task`. These invariants ensure that the autonomous controller obtains sensing data from a sensor during the `Perception` stage. Furthermore, the perceived environmental features, such as features, path, and accuracy, are computed based on the interpretation of this sensing data (SR2.2). Subsequently, the stage of the autonomous controller is transitioned to the `Decision` stage. The definitions of these invariants are as follows:

```
//identify the perceived environmental features based on the sensing data
@detection_task: stage = Perception ⇒ sensing_data_env ∈ ran(sensor) ∧ (
sensing_data_env ↳ generic_features ↳ accuracy) ∈ dom(OEDR_task)
//recognise the set of positions (path) according to the interpretation of sensing data
@recognition_task: stage = Decision ⇒ generic_path ∈ ran(OEDR_task)
```

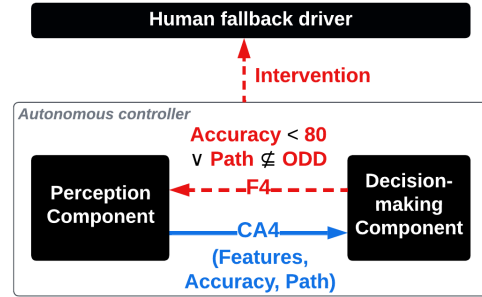
When an autonomous controller enters the `Decision` stage, it goes through two sub-stages: `Intervention` and `Control`. In the `Intervention` stage, a semi-automated system may prompt a human driver to take control of the vehicle (SR2.3) if the autonomous controller detects potential issues such as going outside its ODD or if there are inaccuracies in identifying the path or environmental features. This is done in a feedback loop

(F4), where the autonomous controller evaluates the identification of environmental features. To model these reasoning aspects, we developed the event `violationExpectedFeatures`, which is explained as follows:

```

event violationExpectedFeatures when
  @grd1 : Sys_status = ON
  @grd2 : stage = Decision
  //F4: feature checking
  @grd3 : accuracy < 80  $\vee$  generic_path  $\not\subseteq$ 
    ODD
then
  @act1 : stage := Intervention
end

```

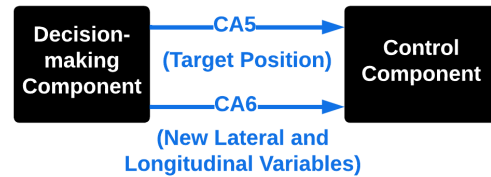


On the other hand, during the preparation of the `Control` stage, the decision-making component of a semi-automated system calculates a target position (CA5) and determines necessary adjustments to the vehicle's lateral and longitudinal variables (CA6). This calculation is based on the identified path and features, as well as the current lateral and longitudinal variables of the vehicle. Therefore, the event `decision` is modelled as follows:

```

event decision when
  @grd1 : Sys_status = ON
  @grd2 : stage = Decision
  @grd3 : accuracy  $\geq$  80  $\wedge$  generic_path  $\subseteq$  ODD
then
  //position obtained from path
  @act1 : selectPos := specify_target_position(
    generic_path)
  //lat/long based on the selected position
  @act2 : selectLat := compute_lateral(
    selectPos  $\mapsto$  LATERAL_VAR)
  @act3 : selectLon := compute_longitudinal(
    selectPos  $\mapsto$  LONGITUDINAL_VAR)
  @act4 : stage := Control
end

```



Based on event `decision`, we have added two invariants: `selPosition` and `selLatLon`. An invariant `selPosition` ensures that a target (new) position will compute from the set of positions located inside the identified path (SR2.4). In addition, an invariant `selLatLon` is also added to ensure that a required change of lateral and longitudinal variables are computed according to the target position and the current lateral and longitudinal variables of an SDV (SR2.5). The definitions of these invariants are as follows:

```

//target position is based on path where detection accuracy is high
@selPosition : stage = Decision  $\wedge$  accuracy  $\geq$  80  $\wedge$  path  $\subseteq$  ODD  $\Rightarrow$  path  $\in$  dom(
  specify_target_position)

```

```

/*target lateral and longitudinal identified based on a new (target) position*/
@selLatLon : stage=Control ∧ accuracy ≥ 80 ∧ path ⊆ ODD ⇒ (selectPos ↦ LATERAL_VAR) ∈ dom
  (compute_lateral) ∧ (selectPos ↦ LONGITUDINAL_VAR) ∈ dom (compute_longitudinal)

```

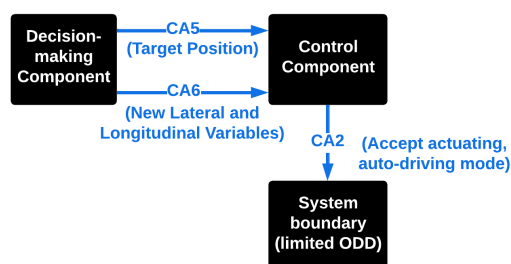
In the monitoring process of implementing new lateral and longitudinal variables (CA6), the semi-automated system must ensure that any changes to these variables do not exceed the maximum or minimum range of the actuators involved in the SDV. To achieve this, we have developed a model that breaks down the monitoring process into two distinct events.

1. The first event involves the acceptance of the actuating process by the event **control** (SR2.7). For instance, if the new lateral and longitudinal variables (CA2) fall within the acceptable range of the actuators to move a SDV within its limited ODD, the autonomous controller's stage is changed to **AutonomousDriving**. The event **control** is modelled as follows:

```

event control when
@grd1 : Sys_status = ON
@grd2 : stage = Control
//long/lat in the defined range
@grd3 : selectLat ∈ LATERAL ∧
  selectLon ∈ LONGITUDINAL
//new long/lat lead to the ODD
@grd4 : move(SDV_POSITION_env ↦
  selectLon ↦ selectLat) ⊆ ODD
then
@act1 : stage:=AutoDriving
end

```

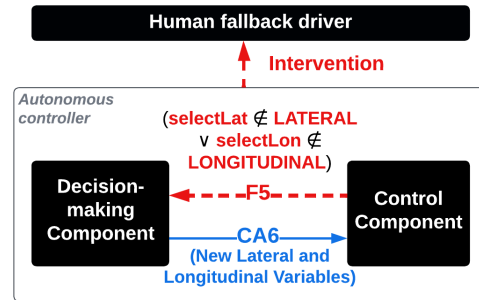


2. The second event, **violationLimit**, represents a scenario where the autonomous controller attempts to exceed the acceptable limits of the lateral and longitudinal variables of the actuators involved in the SDV (F5), or where modifications to the lateral and longitudinal variables will result in a new position outside the limited ODD (F5). In such cases, the semi-automated system may require human intervention, and the control of the SDV is transferred to the driver (SR2.6 and SR2.8). Therefore, the autonomous controller's stage is changed to **Intervention**. The event **violationLimit** is modelled as follows:

```

event violationLimit where
  @grd1: Sys_status = ON
  @grd2: stage = Control
  //F5:violates actuator limits
  @grd3: selectLat  $\notin$  LATERAL  $\vee$ 
    selectLon  $\notin$  LONGITUDINAL  $\vee$ 
    move(SDV_POSITION_env  $\mapsto$ 
      selectLon  $\mapsto$  selectLat)  $\not\subseteq$  ODD
then
  @act1: stage := Intervention
end

```



In order to translate digital signals into mechanical actions or movements in SDVs, a Boolean flag `signal` is used to indicate the actuation of lateral and longitudinal variables. When the `signal` is `TRUE`, the semi-automated system moves to a new position. However, since these variables can be actuated either autonomously or manually, two events are developed to model these scenarios.

1. The first event, `auto_driving`, is derived from the abstract event `auto_actuating` and involves autonomously setting the lateral and longitudinal variables (`CA2`). To accomplish this, an Event-B witness is utilised to map between the abstract parameter and the new variables identified by the autonomous controller's operations. During this process, the controller's stage is in `AutonomousDriving`. The event `auto_driving` is modelled as follows:

```

event auto_driving refines
  auto_actuating
where
  @grd1: Sys_status = ON
  @grd2: selectLat  $\in$  LATERAL
  @grd3: selectLon  $\in$  LONGITUDINAL
  @grd4: move(SDV_POSITION_env  $\mapsto$ 
    selectLon  $\mapsto$  selectLat)  $\subseteq$  ODD
  @grd5: stage = AutonomousDriving
  @grd6: signal_flag = FALSE
then
  /* setting autonomously */
  @act1: LATERAL_VAR := selectLat
  @act2: LONGITUDINAL_VAR :=
    selectLon
  /* ready to move */
  @act3: signal_flag := TRUE
with
  /* replace autonomously */
  @auto_lat: auto_lat = selectLat
  @auto_lon: auto_lon = selectLon
end

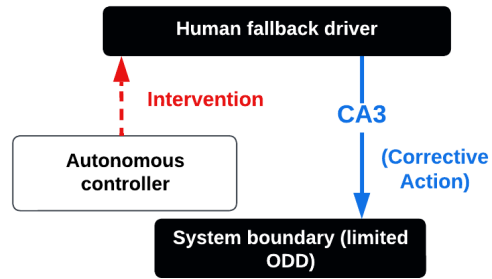
```

2. The second event, `manual_driving`, is extended from an abstract event `manual_actuating` and is designed to simulate the corrective action taken by a human driver when the autonomous controller is in `Intervention` stage. The event `manual_driving` is modeled as follows:

```

event manual_driving extends
manual_actuating where
@grd4: stage = Intervention
@grd5: signal_flag = FALSE
then
/* ready to move */
@act3: signal_flag := TRUE
end

```

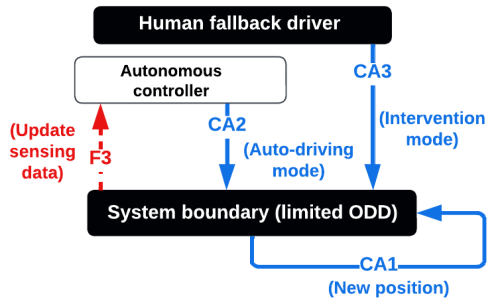


The event `move` is extended from an abstract event `move` to demonstrate how the SDV can move into a position inside the limited ODD, regardless of whether new lateral and longitudinal variables are manually actuated in `Intervention` mode or autonomously actuated in `AutonomousDriving` mode. Additionally, the sensor will update incoming sensing data based on the new position of the SDV (`SR2.1`). The event `move` is modelled as follows:

```

event move extends move
where @grd3: signal_flag = TRUE
@grd4: stage ∈ {AutonomousDriving ,
Intervention}
then
@act2: stage := Perception
@act3: sensing_data_env := sensor(
new_position)
@act4: signal_flag := FALSE
end

```



In order to ensure that the actuation task of an autonomous controller maintains the safety of the system, specifically that $SDV_POSITION_env \in ODD$, we added an environmental invariant `Environment-consistency` which is formulated as follows:

```

@Environment-consistency: stage = AutonomousDriving
⇒ move(SDV_POSITION_env ↦ selectLon ↦ selectLat) ⊆ ODD

```

Table 6.11 presents how the formal model satisfies safety requirements (`SR2.1` to `SR2.8`). The refined model (machine `m2`) involves several assumptions (A) in a formal model:

- **A2.1:** The sensor system will always provide sensing data for the current state of a limited ODD.
- **A2.2:** The autonomous controller can obtain sensing data from the sensor attached to an SDV.
- **A2.3:** Perceived environmental features, such as features, path, and accuracy, are computed based on the current position of an SDV where the sensing data are visible.

- **A2.4:** The autonomous controller can make a judgment on the detection process of the perceived environmental features based on the accuracy score. We assume that if the accuracy score is less than 80%, the autonomous controller will ask a human driver to take over the control of an SDV (Intervention stage).
- **A2.5:** We assume that a human driver is always be receptive when the semi-automated system requests the driver's intervention.

TABLE 6.11: Safety requirements (SR2.1 to SR2.8) in formal model for RATP Layer 2

Safety requirements	Event-B elements
SR2.1	Invariant, @consistency: $\text{sensing_data_env} = \text{sensor}(\text{SDV_POSITION_env})$
SR2.2	Two invariants developed based on the event <code>perception</code> , @ <code>detection_task</code> : $\text{stage} = \text{Perception} \Rightarrow \text{sensing_data_env} \in \text{ran}(\text{sensor})$ $\wedge (\text{sensing_data_env} \mapsto \text{generic_features} \mapsto \text{accuracy}) \in \text{dom}(\text{OEDR_task})$ @ <code>recognition_task</code> : $\text{stage} = \text{Decision} \Rightarrow \text{generic_path} \in \text{ran}(\text{OEDR_task})$
SR2.3	Guard <code>grd3</code> and action <code>act1</code> in <code>violationExpectedFeatures</code> event, @ <code>grd3</code> : $\text{accuracy} < 80 \vee \text{generic_path} \not\subseteq \text{ODD}$ @ <code>act1</code> : $\text{stage} := \text{Intervention}$
SR2.4	An invariant developed based on the event <code>decision</code> , @ <code>selPosition</code> : $\text{stage} = \text{Decision} \wedge \text{accuracy} \geq 80 \wedge \text{generic_path} \subseteq \text{ODD}$ $\Rightarrow \text{generic_path} \in \text{dom}(\text{specify_target_position})$
SR2.5	An invariant developed based on the event <code>decision</code> , @ <code>selLatLon</code> : $\text{stage} = \text{Control} \wedge \text{accuracy} \geq 80 \wedge \text{generic_path} \subseteq \text{ODD}$ $\Rightarrow (\text{selectPos} \mapsto \text{LATERAL_VAR}) \in \text{dom}(\text{compute_lateral}) \wedge$ $(\text{selectPos} \mapsto \text{LONGITUDINAL_VAR}) \in \text{dom}(\text{compute_longitudinal})$
SR2.6 & SR2.8	Guard <code>grd3</code> and action <code>act1</code> in <code>violationLimit</code> event, @ <code>grd3</code> : $\text{selectLat} \notin \text{LATERAL} \vee \text{selectLon} \notin \text{LONGITUDINAL} \vee$ $\text{move}(\text{SDV_POSITION_env} \mapsto \text{selectLon} \mapsto \text{selectLat}) \not\subseteq \text{ODD}$ @ <code>act1</code> : $\text{stage} := \text{Intervention}$
SR2.7	Guards (<code>grd3</code> , <code>grd4</code>) and action <code>act1</code> in <code>control</code> event, @ <code>grd3</code> : $\text{selectLat} \in \text{LATERAL} \wedge \text{selectLon} \in \text{LONGITUDINAL}$ @ <code>grd4</code> : $\text{move}(\text{SDV_POSITION_env} \mapsto \text{selectLon} \mapsto \text{selectLat}) \subseteq \text{ODD}$ @ <code>act1</code> : $\text{stage} := \text{AutonomousDriving}$

6.2.4 Third Analysis Process (Layer 3)

Step 1. Instantiation: The semi-automated systems are responsible for ensuring the driver's awareness level, especially when the SDV prompts the driver to take control. To accomplish this, we assume that the Driver Monitoring System (DMS) inside the human fallback controller computes the driver's awareness level [52]. In Figure 6.17, we refined the previous system boundary diagram to show how the DMS interprets the driver's input, such as *eyes*, *head*, and *hands of the human driver*, in order to compute the driver's awareness level when the semi-automated system may require intervention. This refinement allows for a better understanding of how the SDV systems function and how it ensures the responsiveness of the fallback driver in the intervention scenarios.

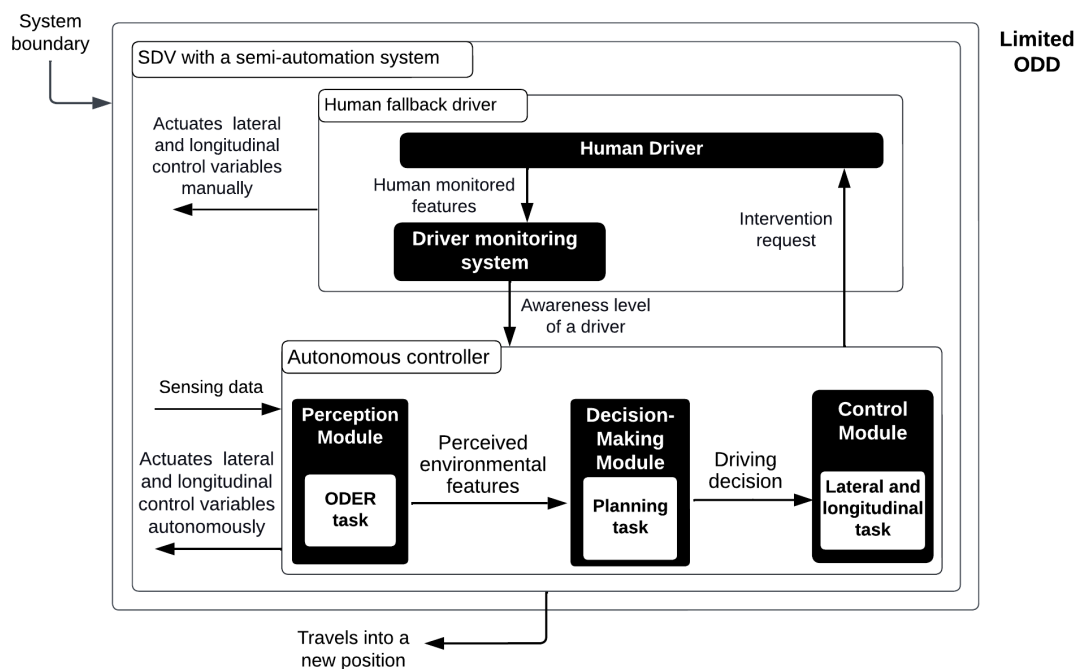


FIGURE 6.17: System boundary diagram in RATP Layer 3

Step 2. The purpose of analysis: In this layer, the purpose of the analysis is to study how the awareness level of a driver is involved in the autonomous operations of a system when the semi-automated system may issue a request to intervene. A refined System Loss (**SL3**) can be derived from the previous safety requirements (**SR2.3**, **SR2.6**, and **SR2.8**):

- **SL3:** The human fallback driver does not take control of the vehicle when the semi-automated system requests intervention.

Since the human fallback controller is responsible for ensuring the driver's responses, the System Hazards (SHs) associated with the system loss (**SL3**) are identified as follows:

- **SH3.1:** The human driver is not attentive to the autonomous operations of a semi-automated system.
- **SH3.2:** The autonomous controller of a semi-automated system issues a request to intervene while the awareness level of a human fallback driver is unknown.

To prevent these hazards, a high-level safety constraint (SC) is defined as follows:

- **SC3:** The human fallback driver must pay attention to the autonomous operations of the system when the autonomous controller moves the SDV autonomously.

Step 3. Control structure: A refined control structure is developed to ensure the awareness of the human fallback driver during the autonomous operations of a semi-automated system, as shown in Figure 6.18. To explain how the DMS verifies the awareness level of the driver, the human fallback controller is added to the semi-automated system. For example, CA8 represents a driver input that enables the DMS to ensure the driver’s awareness level. Furthermore, to capture high-level interactions between the autonomous and human fallback controllers, CA7 and F6 are added. CA7 highlights the responsibility of the human fallback controller to share the driver’s awareness level with the autonomous controller, whereas F6 simulates a potential feedback loop where the autonomous controller may request intervention.

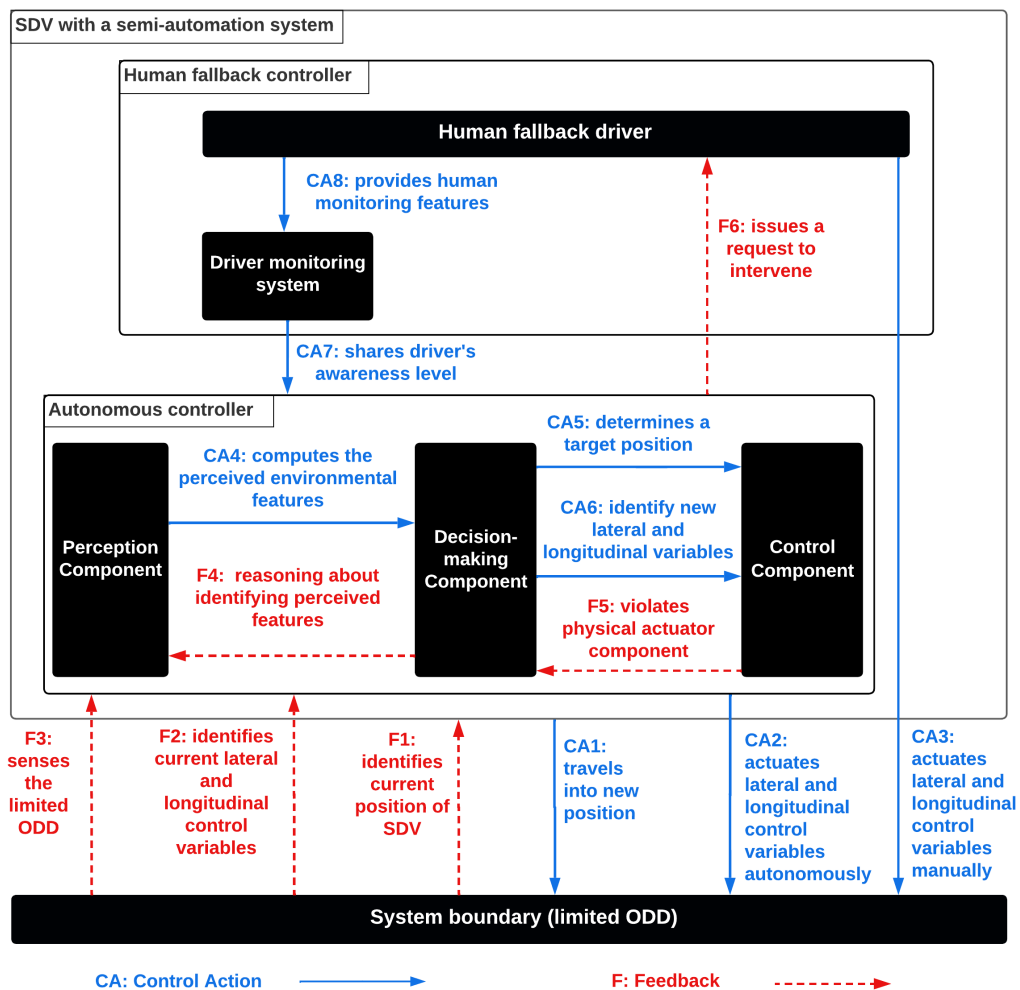


FIGURE 6.18: Control structure in RATP Layer 3

Step 4. Identifying unsafe control actions: At this layer, the UCAs are demonstrated in two categories. The first category comprises the unsafe operations of the DMS (SH3.1), which are presented in Table 6.12 and explained below.

- The autonomous controller performs its operations, such as actuating lateral and longitudinal variables autonomously, while the awareness level of the human fallback driver is unverified by the DMS.
- The semi-automated system may function before the DMS verifies the awareness level of the human fallback driver.

TABLE 6.12: Unsafe control actions of DMS component in RATP Layer 3

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA8	UCA4.1: Autonomous controller performs its operations while the awareness level of a driver is unknown.	UCA4.2: Autonomous controller performs its operations while the awareness level of a driver is unverified by the DMS (CA8).	UCA4.3: Autonomous controller performs its operations before the DMS verifies the driver's awareness level (CA8).	N/A
Causal Factor				
A human fallback driver is unaware of autonomous operations.				

As a result, the potential safety requirements related to the DMS component can be identified as follows:

- **SR3.1:** To activate the semi-automated system, the DMS must ensure the awareness level of the human fallback driver [SH3.1].
- **SR3.2:** The DMS must compute the awareness level of a driver based on the human monitoring features, such as eyes, head, and hands of the human driver [SH3.1].
- **SR3.3:** If the human fallback driver does not provide the human monitoring feature, the semi-automated system will be immediately deactivated [SH3.1].

The second category includes the unsafe operations of a system when a request to intervene is sent (SH3.2). These unsafe operations are demonstrated in Table 6.13 and explained as follows:

- The intervention request might be sent when the awareness level of the human fallback driver is unknown.
- The autonomous controller relies on the correction action of the human fallback driver while their awareness level is unknown.

Therefore, the potential safety requirements associated with SH3.2 can be identified as follows:

TABLE 6.13: Unsafe control actions of driver intervention in RATP Layer 3

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA7	UCA4.4: Autonomous controller performs its operations while the awareness level of a driver is unaware.	UCA4.5: Autonomous controller issues a request to intervene (F6) when the status of the human fallback driver is unaware (CA7).	UCA4.6: Autonomous controller relies on the correction action of the human fallback driver (CA3) while their awareness level is unknown (CA7).	N/A
Causal Factor				
Unknown reactions of a fallback driver when a semi-automated system issues a request to intervene.				

- **SR3.4:** The human fallback controller must share the awareness level of a driver with the autonomous controller [SH3.2].
- **SR3.5:** If the autonomous controller system issues a request to intervene, the driver will be responsible for performing the entire driving task (manual driving) [SH3.2].

Step 5. Event-B model: A full script of the refined machine (m3) can be seen in Section D.4 in Appendix D. Mapping is provided in Table 6.14 to detail how CA7, CA8 and F6 linked to the representation of a refined model (m3).

TABLE 6.14: Control actions and feedback loops in formal model for RATP Layer 3

Control Action & Feedback loop	Event-B elements
CA7	Variable, awarenessLevel
CA8	Variable, driver_input
F6	Variable, intervRequest

We modelled the awareness level of a driver and the intervention request using Booleans as follows:

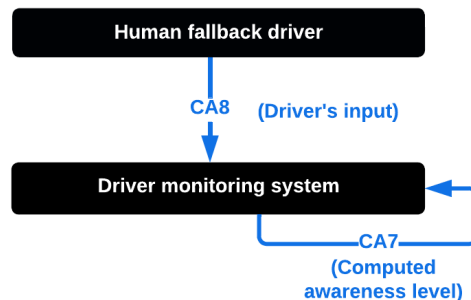
```

/* driver either aware (TRUE) or unaware (FALSE)*/
@typeof-awarenessLevel: awarenessLevel ∈ BOOL
/* DMS uses the driver's input to compute the awareness level of a driver */
/*i.e, driver's input is provided (TRUE) or not provided (FALSE)*/
@type-driverInput: driver_input ∈ BOOL
/* semi-system may issue a request to intervene*/
/*i.e, request is sent (TRUE), unsent (FALSE)*/
@typeof-intervention: intervRequest ∈ BOOL

```

A Boolean variable `awarenessLevel` (CA7) is introduced to compute the awareness level of a driver (SR3.2) based on how the DMS detects the `driver_input` (CA8); therefore, a new invariant is written, as follows:

```
@awarenesslevel: awarenessLevel
=TRUE⇒ driver_input=TRUE
```



In the operations of DMS, two events are used to detect human monitoring features and calculate the awareness level of the human fallback driver. The first event, `DMS_driver_input_detect`, models the detection of the awareness level based on the human monitoring feature. When this event is triggered, it sets the driver input to `TRUE` and the awareness level to `TRUE`. This event is modelled as follows:

```

event DMS_driver_input_detect
when
  @grd1: driver_input = FALSE
  @grd3: awarenessLevel = FALSE
then
  @act1: driver_input := TRUE
  @act2: awarenessLevel := TRUE
end

```

The second event, `DMS_driver_input_remove`, models the possible change in awareness level when the human fallback driver does not provide the human monitoring feature. This event is modelled as follows:

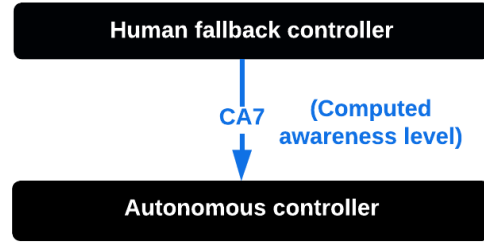
```

event DMS_driver_input_remove
when
  @grd2: driver_input = TRUE
  @grd3: awarenessLevel = TRUE
then
  @act1: driver_input := FALSE
  @act2: awarenessLevel := FALSE
end

```

In order to ensure that the human fallback driver is aware of autonomous operations, we added a new invariant `driverAware` to verify the awareness level of a driver when the semi-automated system performs its operations.

```
@driverAware: awarenessLevel =
  TRUE ⇒ stage ∈ {Perception ,
  Decision, Control,
  Intervention,
  AutonomousDriving}
```



According to an invariant `driverAware`, a new guard is added to the autonomous events, such as perception, to ensure that the semi-automated system is only operating (SR3.4) when the awareness level of a driver is verified by the DMS (SR3.1).

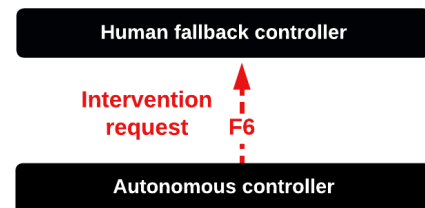
```
event perception extends perception
when
  @grd5: awarenessLevel = TRUE
end
```

Based on the intervention cases in the previous machine (m2), the semi-automated system may issue a request to intervene in two driving scenarios:

1. If the perception module of an autonomous controller identifies the path or environmental features with a low confidence score (F4), modelled earlier in event `violationExpectedFeatures`.
2. If the control module of an autonomous controller attempts to exceed the acceptable limits of the lateral and longitudinal variables of the actuators involved in the SDV (F5), or where modifications to the lateral and longitudinal variables will result in a new position outside the limited ODD (F5), modelled earlier in event `violationLimit`.

Therefore, we added a new invariant `intervCases` to ensure that the intervention request (F6) sent if the stage of semi-automated system is in `Intervention` (SR3.5).

```
@intervCases: intervRequest = TRUE
⇒
  accuracy < 80 ∨
  generic_path ∉ ODD ∨
  selectLat ∉ LATERAL ∨
  selectLon ∉ LONGITUDINAL ∨
  move(SDV_POSITION_env ↦
  selectLon ↦ selectLat) ∉ ODD
```



Based on this invariant `intervCases`, the intervention events such as `violationLimit`, modelled in the previous machine (m2), are extended as follows:

```

event violationExpectedFeatures
extends violationExpectedFeatures
where
@grd4: awarenessLevel = TRUE
@grd5: intervRequest = FALSE
then
@act2: intervRequest := TRUE
end

```

```

event violationLimit
extends violationLimit
where
@grd4: awarenessLevel = TRUE
@grd5: intervRequest = FALSE
then
@act2: intervRequest := TRUE
end

```

The `move` event is divided into two events: `auto_move` and `manual_move`. These events represent how the SDV moves to a new position (CA1). Specifically, the `auto_move` event is activated when there is no intervention request, and it assumes that the SDV applies lateral and longitudinal variables autonomously (CA2). This event is modelled as follows:

```

event auto_move refines move any
new_position where
@grd1: new_position ∈ ODD
@grd2: new_position ∈ move(
  SDV_POSITION_env ↦
  LONGITUDINAL_VAR ↦
  LATERAL_VAR)
@grd3: signal_flag = TRUE
/* new guards*/
@grd4: stage = AutoDriving
@grd5: awarenessLevel = TRUE
/* no intervention request */
@grd6: intervRequest = FALSE
then
@act1: SDV_POSITION_env :=
  new_position
@act2: stage := Perception
@act3: sensing_data_env := sensor(
  new_position)
@act4: signal_flag := FALSE
end

```

The `manual_move` event is used when the semi-automated system is in the `Intervention` stage. In this case, the SDV applies lateral and longitudinal variables manually (CA3), meaning that a human driver is involved in the movement of the SDV. This event is modelled as follows:

```

event manual_move refines move
any new_position where
@grd1: new_position ∈ ODD
@grd2: new_position ∈ move(
  SDV_POSITION_env ↦
  LONGITUDINAL_VAR ↦
  LATERAL_VAR)
@grd3: signal_flag = TRUE
@grd4: stage = Intervention
@grd5: awarenessLevel = TRUE
@grd6: intervRequest = TRUE
then
@act1: SDV_POSITION_env :=
  new_position
@act2: stage := Perception
@act3: sensing_data_env := sensor(
  new_position)
@act4: signal_flag := FALSE
@act5: intervRequest := FALSE
end

```

Table 6.15 shows how the safety requirement (SR3.1 to SR3.5) has been satisfied in a formal model. In the refined model (machine `m3`), there are two assumptions (A) involved in a formal model:

- **A3.1:** A human driver is receptive to any intervention requests when the semi-automated system issues a request to intervene.
- **A3.2:** To determine the awareness level of a driver, we assume the use of a sensor that is attached to the DMS events. This sensor monitors changes in the driver's inputs, such as eye and hand movements. By analysing these inputs, we can calculate the driver's level of awareness.

TABLE 6.15: Safety requirements (SR3.1 to SR3.5) in formal model for RATP Layer 3

Safety requirements	Event-B elements
SR3.1 & SR3.3	Invariant, $@driverAware: awarenessLevel = TRUE \Rightarrow stage \in \{Perception, Decision, Control, Intervention, AutonomousDriving\}$
SR3.2	Invariant, $@awarenesslevel: awarenessLevel = TRUE \Rightarrow driver_input = TRUE$
SR3.4	New guards in autonomous events, such as $@grd5$ in <code>perception</code> , $@grd5: awarenessLevel = TRUE$
SR3.5	Invariant for issuing intervention request, $@intervCases: intervRequest = TRUE \Rightarrow accuracy < 80 \vee generic_path \not\subseteq ODD \vee selectLat \not\subseteq LATERAL \vee selectLon \not\subseteq LONGITUDINAL \vee move(SDV_POSITION_env \mapsto selectLon \mapsto selectLat) \not\subseteq ODD$ Where new guard and action in the event <code>manual.move</code> to simulate the driver's reaction $@grd6: intervRequest = TRUE$ $@act5: intervRequest := FALSE$

6.3 Discussion

The RATP approach addresses [RQ3.1](#) through multifaceted aspects. Initially, it offers systematic and rigorous steps for analysing SDV systems, breaking down the analysis into multiple iterative layers. Each layer of analysis is carefully structured to ensure comprehensive coverage of system behaviours and interactions.

Subsequently, the RATP approach provides precise requirements and assumptions for each analysis layer, which are then assigned to specific formal representations. This ensures that all aspects of system safety are thoroughly considered and represented in a structured manner.

Moreover, the RATP approach introduces analysis layers as modelling patterns, which serve to investigate the complexity of safety in SDV systems. These patterns are designed to provide a blueprint structure for understanding the dynamics of the system. They allow for the exploration of system behaviours from a high abstract level down to a concrete level, where a human driver may need to intervene in hazardous events.

Finally, the RATP approach includes a formal verification process, where the generic automation features undergo iterative steps to ensure their consistency and traceability into the formal models. This step further strengthens the robustness of the RATP

approach in addressing safety concerns in SDV systems, as the system specifications are gradually introduced through refinement techniques. In addition, this formal verification process adds an additional layer of verification, ensuring that the identified automation features are accurately represented and aligned with the safety requirements of the SDV system.

6.4 Advantages, Concerns and Related Work

The RATP approach has been proposed as a viable model for reuse in specific case studies. The next chapter presents the application and reuse of patterns in a concrete case study. Before embarking on the case study, this section seeks to outline the advantages of using the RATP approach (Section 6.4.1), as well as its limitations and concerns regarding its application (Section 6.4.2). We compare the RATP approach to studies that have successfully used methodologies such as STPA and Event-B to analyse safety-critical systems, including SDV systems (Section 6.4.3).

6.4.1 Advantages

The RATP approach has several advantages. First, it involves a template that identifies the automation aspects of semi-automated systems for performing Dynamic Driving Tasks (DDTs). These aspects are organised into the high-level modules involved in the semi-automated systems, where the autonomous controller and the human fallback driver engage together to achieve safe driving tasks. By breaking down the system into modules and analysing each one, the RATP approach helps to ensure that the system is safe and operates as intended.

Second, the RATP employs an iterative process that involves repeating a series of systematic steps to refine and improve the quality of the entire design. By using iterative analysis, the requirements or constraints of a target system are gradually identified, and the behaviours of the system are investigated from a high-level abstract view to a more detailed, concrete view.

Third, the RATP provides robust traceability of system losses, hazards and their corresponding safety requirements or constraints into associated formal representations. For example, if a safety requirement (SR2.1) states that the SDV must use a sensor to obtain sensing data based on multiple positions, an Event-B invariant can be written as `sensing_data_env = sensor (SDV_POSITION_env)` to ensure that the sensing data are taken on the basis of the SDV 's current position.

Fourth, the RATP verifies a safety property of a system in an Event-B invariant at the early stage of analysis. The behaviours of a system and their corresponding Event-B elements are gradually added in refinement steps. For instance, if the SDV must always be within the limited ODD (**SC0**), an abstract safety invariant can be written as $SDV_POSITION_env \in ODD$. Further refinements are made by adding an environmental invariant to ensure that the movement of a semi-automated system maintains an abstract safety invariant when the physical lateral and longitudinal variables are autonomously manipulated. Therefore, a new invariant is identified as $stage = AutonomousDriving \Rightarrow move(SDV_POSITION_env \mapsto selectLon \mapsto selectLat) \subseteq ODD$. This means that when the SDV is in autonomous driving mode, the movement of the vehicle must be within the limited ODD.

6.4.2 Limitations

The RATP approach promises benefits, but it also has limitations and concerns that may impact its effectiveness in certain contexts. It is important to acknowledge these limitations.

First, while the automation aspects involved in the template are simple, they may require domain expertise to instantiate them accurately. For example, the perceived environmental features are intended to capture how the perception module interprets incoming sensing data to keep the semi-automated system within its ODD. In this case, the perceived environmental features would be identified on the basis of the specific ODD definition, where the semi-automated system is supposed to operate. Therefore, it is crucial to have a deep understanding of what the semi-automated system is intended to achieve in a specific ODD.

Second, the RATP approach relies on several assumptions during the development of its modelling patterns, for instance that the semi-automated system may depend on the human fallback driver to take control of the vehicle in a hazardous situation. Therefore, it is assumed that a driver responds when the semi-automated system requests their intervention (**A2.5**).

Third, the RATP approach has strengths and weaknesses that are included in the use of hazard identification method of *STPA*, and formal verification approach of *Event-B*. The *STPA* method focuses on the dynamic control of a system and on the causes of hazards in the absence of component failure. Therefore, hazards could still remain that might be the result of component failure. On the other hand, formal verification methods, such as *Event-B*, can verify only what they are explicitly designed to verify. As a result, knowledge might be necessary regarding the domain in which the suggested formalisms are being developed.

6.4.3 Related Work

This section presents the RATP approach in comparison to other methodologies in the field of safety-critical systems. The related work is divided into two main categories:

1. **Adoption of STPA steps in the RATP approach:** here, we evaluate how the RATP approach incorporates the STPA steps and compare it to how other methodologies use these steps.
2. **Integration of formal methods with STPA:** in this subsection, we explore the integration of formal methods with STPA.

Each of these categories is detailed in its own subsection.

6.4.3.1 STPA Steps in RATP

STPA is a hazard analysis method that has been used in various application domains. However, this subsection focuses on a modified method derived from the traditional/original STPA steps proposed by Leveson and Thomas [84].

STPA-Sec [139] was primarily developed to extend the original STPA steps to cover the security concerns. Similar to STPA, STPA-Sec comprises four main steps. It begins with identifying the losses to be considered and leads to the insecure scenarios that may violate security constraints. STPA-Sec specifically emphasises analysing the vulnerability of the system to external attacks, whilst STPA prioritises the identification of hazards and enhancing the safety of the target system.

STPA-Priv [118] is another modification of STPA that focuses primarily on a system's privacy. It also adopts similar STPA steps, but it suggests using open-loop controls, when developing a control structure, to reflect the privacy properties. Open-loop controls do not require feedback loops from the controlled processes. For example, users may or may not read a privacy guideline, nonetheless there is often no feedback provided to the controller.

DeepSTPA [111] follows the same methodology as traditional STPA, and is proposed primarily for analysing Learning-Enabled Systems (LESs)². Compared to traditional STPA, deepSTPA enhances the development of a control structure by adopting layer-wise functionalities to establish links between development activities within the control structure.

In the RATP approach, the traditional STPA steps are presented in Steps 1 to 4. In contrast to previous modifications of STPA, we used a template at the beginning of

²Learning-Enabled Systems are complex systems that can learn from data and experiences in order to improve their performance over time [110].

the analysis to study system safety against identified hazardous states at various levels of abstraction. The main advantage of this view is that it enables analysis of system behaviours from a high-abstraction layer to a more detailed, concrete layer.

6.4.3.2 Formal Methods with STPA

The Event-B formal method was used with STPA to ensure that the identified hazards were prevented or mitigated at the early stage of design. Colley and Butler [35] developed a method to demonstrate and formally analyse the critical requirements (artefacts) generated by the STPA analysis method. The goal was to use modelling techniques, such as formal verification, via the use of the Event-B formal method and its Rodin toolset [7]. Similarly, Howard et al. [69] used STPA to define critical requirements and employed formal models in Event-B to ensure the effective mitigation of vulnerable system states. To achieve this, they constructed an abstract model based on the control actions outlined in the STPA control structure. Additionally, they developed a concrete/refined model that incorporated the critical requirements identified through analysis of these control actions. Dghaym et al. [40] then extended the work of [35, 69] to develop a compositional approach to elicit the critical requirements for autonomous functions, and next formalised these critical requirements into Event-B models.

STPA has been applied alongside other formal methods in various studies. Abdulkhaleq et al. [2] developed a safety engineering framework that combines STPA with formal verification. STPA is used to identify the safety requirements, which are then formalised and expressed using temporal language. Formal verification, specifically model checking, was used to ensure that a behaviour model met the controller's behaviour, where the verified model had passed the formalised requirements. Hata et al. [61] formally modelled the constraints/requirements obtained from STPA as pre- and post-conditions in VDM++. By incorporating STPA into the formal modelling process, they aimed to ensure that safety considerations are properly addressed. Thomas and Leveson [123] described a formal syntax for unsafe control actions, which are recognised by applying STPA. This formalisation facilitates the automatic generation of model-based requirements, along with the identification of inconsistencies in those requirements.

In the RATP approach, formal models are developed to capture safety requirements using a series of hierarchical patterns. The process starts with a high-level analysis layer and advances towards a more detailed concrete layer. This view enables the formal models to incrementally capture safety requirements while ensuring robust traceability and consistency through formal verification.

6.5 Conclusion

This chapter introduces the RATP methodology. This methodology consists of five systematic steps used to identify safety requirements and to develop a rigorous model for SDV systems. Within the RATP approach, modelling patterns are organised into four distinct layers. These layers enable the analysis of system behaviour, starting from a high-level abstract view and moving towards a detailed and concrete view. In the coming chapters the focus is on the practical application of these layers, demonstrating their use in a case study.

Chapter 7

LKA and DMS Functions in the ALC System

This chapter focuses on the application of the Rigorous Analysis Template Process (RATP) approach to the Lane Keeping Assist (LKA) and Driver Monitoring System (DMS) functions within the Automated Lane Centring (ALC) system, a process initially summarised in Section 4.2.2 of Chapter 4. This work contributes to the following research question (RQ4):

RQ4: *To what extent does the RATP methodology demonstrate its utility when applied to a case study of varying sizes and complexity, especially those involving interactions between human drivers and an SDV system?*

In alignment with RQ4, this chapter demonstrates how the RATP approach can be applied to model two primary functionalities in the ALC system: the LKA and the DMS functions. Section 7.1 expounds on the systematic iterative process inherent to the RATP approach, detailing its implementation within the context of the analysis of the LKA and DMS features. Section 7.2 gives a conclusion of this chapter.

Parts of this chapter were published in COMPSAC 2023 [13]. Specifically, we published a summary of the application of the RATP approach to the functionalities of LKA and DMS.

7.1 Modelling Pattern of the ALC System

Building upon the modelling patterns discussed in Section 6.2 of Chapter 6, we applied the systematic steps of the RATP methodology in four iterations. Each represents

an analysis process encapsulated within a single layer, gradually introducing the complexity of the ALC system from a high-abstraction layer to a more detailed concrete layer. The analysis aims to achieve the following objectives in each layer:

- **ALC-Layer 0:** Captures the main functionality of a system as the ALC system moves an Self-Driving Vehicle (SDV) to the middle of its desired (target) lane.
- **ALC-Layer 1:** Shows in further detail how the steering angle of an SDV is modified to reach a new position inside the target lane.
- **ALC-Layer 2:** Introduces the autonomous operations of a system such as perception, decision-making and control.
- **ALC-Layer 3:** Adds further details on how the awareness level of a driver is involved in autonomous operations.

Each layer is then further detailed in its own subsection.

7.1.1 Abstraction Level (ALC-Layer 0)

The first RATP aims to capture the high-level aspects of a system and allow the abstraction behaviours of an ALC system to be analysed and modelled. Specifically, we instantiated generic automation aspects discussed in Subsection 6.2.1 of Chapter 6 as follows:

- The limited ODD is instantiated as the target lane where the ALC system executes its functionalities.

Each step of the RATP approach is then further detailed as follows.

Step 1. Instantiation: Starting with the main functionality, the ALC system is designed to position an Self-Driving Vehicle (SDV) in the middle of its desired (or target) lane [94, 10]. Therefore, the SDV system can be defined as a system to align a vehicle with the centre of the target lane. The boundary of the system is defined by the target lane, which represents a specific Operational Design Domain (ODD) of the ALC system. Based on these definitions, the system boundary diagram is constructed, as illustrated in Figure 7.1.

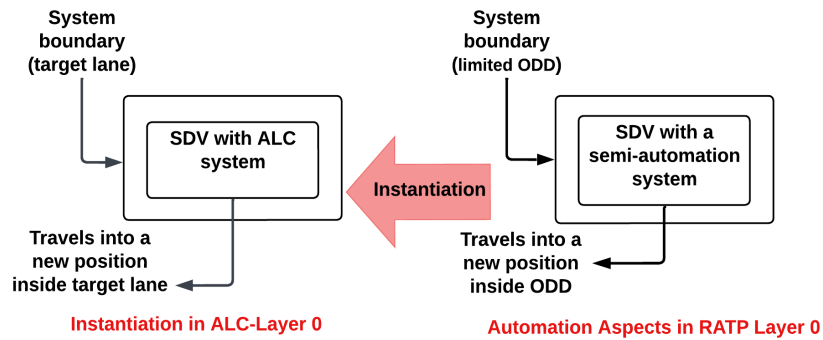


FIGURE 7.1: System boundary diagram in ALC Layer 0

Step 2. The purpose of analysis: With the help of a system component diagram (see Figure 7.1), the main goal of analysis (focusing on its purpose of keeping the vehicle in lane) is emphasised in an unwanted event, namely the System Loss (SL) of an ALC system, **SL0**:

- **SL0**: The SDV collides with an object outside its target lane.

As an SDV may move into a position outside its target lane, a high-level System Hazard (SH) associated with **SL0** is **SH0**:

- **SH0**: The SDV travels into a position outside its target lane.

A high-level Safety Constraint (SC) that satisfies the system conditions to prevent **SH0** is **SC0**:

- **SC0**: The SDV must always be located inside the target lane.

Step 3. Creating control structure: An abstract control structure is demonstrated in Figure 7.2. It involves two main components: 1) *the SDV with an ALC system* and 2) *the target lane*. To capture a high-level interaction between an SDV and the target lane, **SC0** is translated into a set of Control Actions (CAs) and Feedback Loops (FLs). In more detail, **CA1** implies that an SDV may change its position to a new position, while **F1** indicates the monitoring of a change in an SDV's position.

Step 4. Identifying unsafe control actions: The Unsafe Control Actions (UCAs) of **CA1** are demonstrated in Table 7.1. *UCA 1.1* indicates that the ALC system might travel to a new position outside of the target lane; therefore, the identification of Safety Requirements (SRs) for mitigating the occurrence of **SH0** is **SR0**:

- **SR0**: The SDV must travel into a new position inside the target lane[**SH0**].

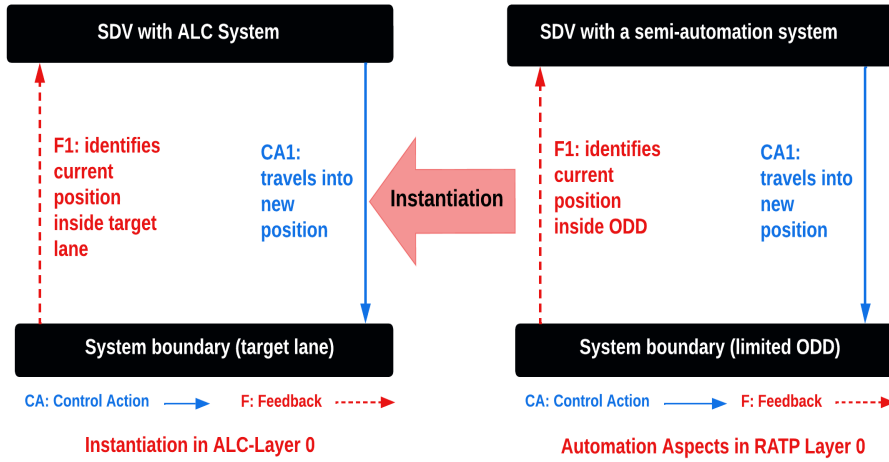


FIGURE 7.2: Control structure in ALC Layer 0

TABLE 7.1: Unsafe control actions in ALC Layer 0

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA1	N/A	UCA1.1: The SDV may travel from a position inside the target lane to a new position outside the target lane.	N/A	N/A
Causal Factor				
Unsafe movement of an SDV				

Step 5. Specifying Event-B model: Mapping is provided in Table 7.2 to demonstrate how CA1 and F1 link with the representation of an abstract model (m0). A full script of the initial machine (m0) can be seen in Appendix E.1.

At the abstraction level (m0 machine), we use a constant Lane to specify all positions of SDVs in the lane, i.e., Lane ⊆ POSITION. The physical position of the SDV (CA1) is modelled as a variable SDV_POSITION_env, with a safety invariant @safety: SDV_POSITION_env ∈ Lane, i.e., the SDV must always be within the lane (SC0). In addition, the initial position of an SDV (F1) is modelled as a constant init_position, where the initialisation position of SDV must be inside the lane, i.e., init_position ∈ Lane.

TABLE 7.2: Control actions and feedback loops in formal model for ALC Layer 0

Control Action & Feedback loop	Event-B elements
CA1	Variable, SDV_POSITION_env
F1	Action, SDV_POSITION_env := init_position

The initial machine (m0) involves four events. First, the initialisation event sets a valid default state of a physical position of the SDV as SDV_POSITION_env := init_position.

Second, the `move` event abstractly captures a movement of an SDV into a new position as follows:

```

event move
any
  new_position
where
  //A new position inside the lane
  @grd1 : new_position ∈ Lane
then
  //moved into a new position
  @act1 : SDV_POSITION_env :=
    new_position
end

```

Besides the `move` event, we added two events, `ALC_ON` and `ALC_OFF`, in order to show the status of the ALC system. These events can demonstrate the activation and deactivation functions as follows:

```

event ALC_ON
where
  @grd1 : ALC_Status = OFF
then
  @act1 : ALC_Status := ON
end

event ALC_OFF
where
  @grd1 : ALC_Status = ON
then
  @act1 : ALC_Status := OFF
  @reset-position : SDV_POSITION_env :=
    init_position
end

```

Table 7.3 shows how a safety requirement (`SR0`) has been captured in a formal model. At this level (machine `m0`), there is only one Assumption (A) involved in a formal model, as `A0`:

- **A0:** The current position of an SDV starts inside the target lane.

TABLE 7.3: Safety requirement (`SR0`) in formal model for ALC Layer 0.

Safety requirements	Event-B elements
SR0	Guard (<code>@grd1</code>) in the <code>move</code> event, <code>@grd1 : new_position ∈ Lane</code>

7.1.2 High-Level Analysis Process (ALC-Layer 1)

The subsequent RATP iteration was conducted to explore the methodology employed by an ALC system in determining its steering angle for transitioning to a new position within the target lane. Specifically, we instantiated generic automation aspects discussed in Subsection 6.2.2 of Chapter 6 as follows:

- The lateral control variable is specified by the steering angle, enabling the ALC system to adjust it to maintain the SDV within the target lane.
- The human driver retains the ability to modify the steering of the SDV, providing corrective action when necessary.

Each phase of the RATP approach is subsequently elaborated as follows.

Step 1. Instantiation: Semi-automation systems, such as an ALC system, assume that the driver provides corrective action if a system fails to operate as expected [101]. This assumption indicates that a human driver is part of the SDV system. Therefore, the SDV system is divided into two main components: 1) *the human fallback driver*; and 2) *the autonomous controller*, as shown in Figure 7.3. A driver might provide a steering angle manually, while the ALC system can apply a steering angle autonomously.

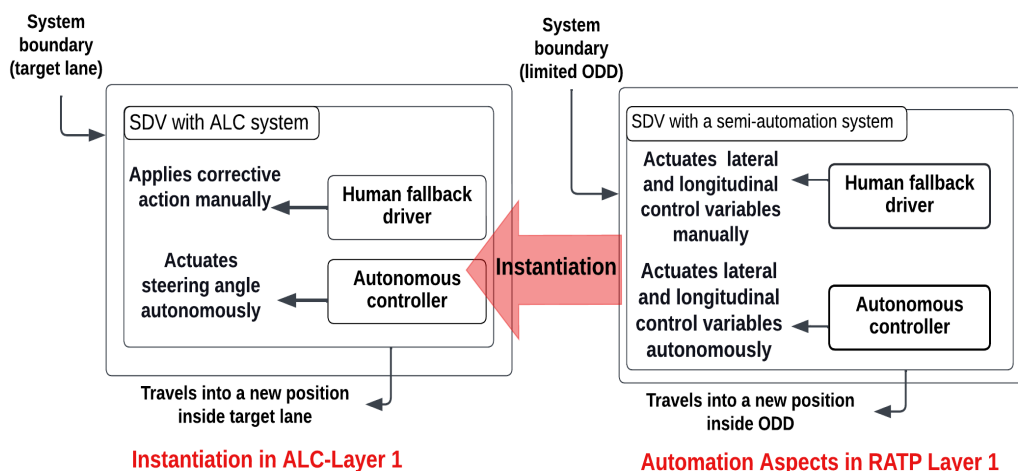


FIGURE 7.3: System boundary diagram in ALC Layer 1

Step 2. The purpose of analysis: A refined system loss is driven from the previously identified safety requirement **SR0** as **SL1**:

- **SL1:** A new steering angle moves an SDV to reach a new position outside of the target lane.

As the steering of an SDV might be modified by either a human driver or an ALC system, a system hazard of **SL1** is **SH1**:

- **SH1:** The steering angle of an SDV is actuated by either a human driver or an ALC system to reach a new position outside the target lane.

A high-level safety constraint associated with **SH1** is **SC1**:

- **SC1:** The steering angle of an SDV must be modified to reach a new position inside the target lane.

Step 3. Refining control structure: We refined a control structure according to how the ALC system may change the steering angle of an SDV, as shown in Figure 7.4. To show a responsible system component for identifying a steering angle, the ALC

system was split into two subcomponents: *the human fallback driver* and *the autonomous controller*. Based on the **SC1**, new CAs and FLs were added. In more detail, **CA2** sets a steering angle of an SDV autonomously, while **F2** monitors a change in a current steering angle of an SDV to identify new steering. For instance, the ALC system in [10] determines a new steering angle of an SDV based on the required manipulation of the current steering angle in a way that satisfies the movement into a new position. Because a human driver might be responsible for setting a new steering angle of an SDV, **CA3** indicates a high-level interaction between a human driver and the target lane when a driver may provide a corrective action if the ALC system fails to operate as expected.

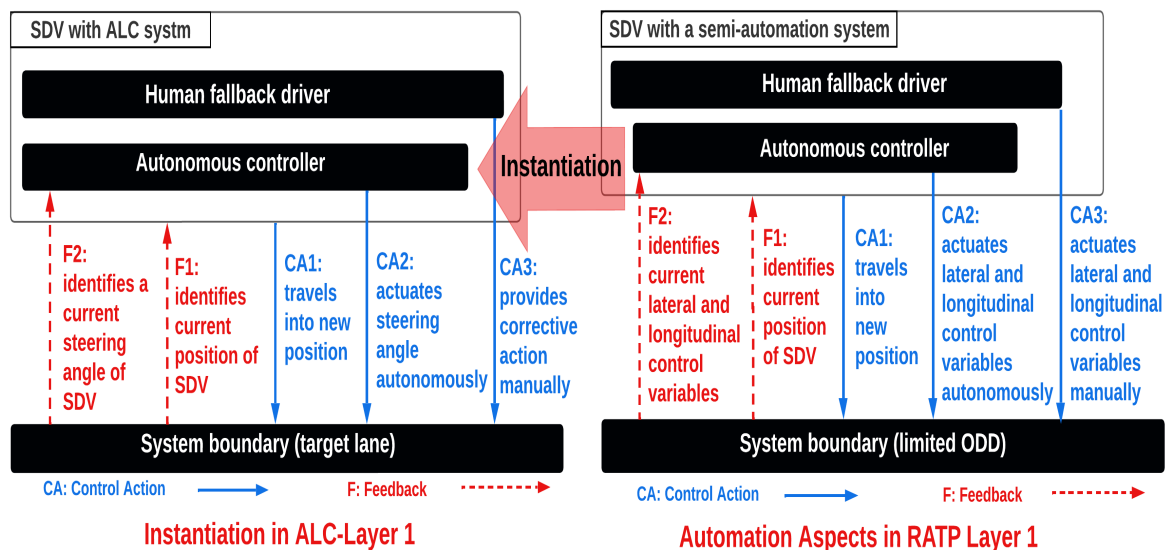


FIGURE 7.4: Control structure in ALC Layer 1

Step 4. Identifying unsafe control actions: The identification of SRs is associated with the UCAs of **CA2** and **CA3**, as shown in Table 7.4. For instance, *UCA 2.2* and *UCA 2.3* indicate that the steering angle of an SDV can be modified to reach a position outside of the target lane while the driver’s corrective action is missing. Therefore, the SRs of **SH1** are as follows:

- **SR1.1:** The ALC system must actuate a steering angle autonomously to reach a target position inside the target lane [**SH1**].
- **SR1.2:** The human driver can provide a corrective action that actuates a steering angle manually to reach a target position inside the target lane [**SH1**].

Step 5. Refining Event-B model: Mapping is provided in Table 7.5 to detail how **CA2**, **CA3** and **F2** are linked to the representation of a refined model. Full details of the refined model are provided in Appendix E.2.

TABLE 7.4: Unsafe control actions in ALC Layer 1

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2 and CA3	UCA 2.1: The ALC system moves a vehicle into a new position inside the target lane (CA1), but the specifications of steering are missing.	UCA 2.2: actuates steering variable autonomously (CA2) to reach a position outside the target lane.	UCA 2.3: driver's corrective action is missing (CA3) when an SDV moves outside the target lane.	N/A
Causal Factor				
Wrong adjustment of the actuating variables (steering).				

TABLE 7.5: Control actions and feedback loops in formal model for ALC Layer 1

Control Action & Feedback loop	Event-Elements
CA2	Variable, <code>SDV_STEERING_ANGLE.env</code>
CA3	Variable, <code>SDV_STEERING_ANGLE.env</code>
F2	Actions in <code>initialisation</code> event, <code>SDV_STEERING_ANGLE.env : ∈ STEERING_ANGLE</code>

At this refinement level (machine `m1`), we modelled the physical steering variable of an SDV as a variable called `SDV_STEERING_ANGLE.env`. This variable has two constants, `max_steering` and `min_steering`, that specify the defined range of steering angles. In addition, we used two constants, `max_steering_constraint` and `min_steering_constraint`, to model an acceptable adjustment of the actual steering to prevent a sudden manipulation of the steering.

The movement of an SDV into a new position is modelled as a constant `move`, where modifications of the steering angle enable the SDV to reach new positions. The definition of the `move` function is shown below.

$$\text{move} \in \text{POSITION} \times \text{STEERING_ANGLE} \rightarrow \mathbb{P}_1(\text{POSITION})$$

Two events model the high-level cases of the ALC system at this layer, i.e., `ALC_actuating` (CA2), `Manual_actuating` (CA3), as shown in Figure 7.4. The `ALC_actuating` event abstractly specifies an autonomous identification of a new steering angle to reach a new position inside the target lane (`SR1.1`).

```

event ALC_actuating                               //SR1.1: new steer leads into lane
any steering_angle_change                         @grd4: move(SDV_POSITION_env ↦ (
where                                             SDV_STEERING_ANGLE_env +
@grd1: ALC_Status = ON                             steering_angle_change)) ⊆ Lane
//steering angle change definition
@grd2: steering_angle_change ∈                    then
    STEERING_ANGLE_CHANGE                          //replace old steering
@grd3: steering_angle_change +                    @act1: SDV_STEERING_ANGLE_env := (
    SDV_STEERING_ANGLE_env ∈                      SDV_STEERING_ANGLE_env +
    STEERING_ANGLE                                 steering_angle_change)
end

```

In the same way, the `Manual_actuating` event describes how a human driver may provide a manual steering angle to reach a new position inside the target lane (**SR1.2**).

```

//SR1.2: driver steering leads to lane
@grd2: move(SDV_POSITION_env ↦
    manual_steering_angle) ⊆ Lane
then
//replace old steering
@act1: SDV_STEERING_ANGLE_env :=
    manual_steering_angle
end

```

Finally, the `move` event actuates either a manual or an autonomous steering angle to move an SDV into a new position (**SC1**).

```

event move extends move
where
//new (target) position must be within a set of positions inside the lane
@grd2: new_position ∈ move(SDV_POSITION_env ↦ SDV_STEERING_ANGLE_env)
end

```

Table 7.6 shows how the safety requirements (**SR1.1** and **SR1.2**) have been captured in a formal model. At this level (machine **m1**) there is a new Assumption (A) involved in a formal model, as **A1**:

- **A1**: The human driver provides the steering angle manually to keep an SDV inside the target lane.

7.1.3 Second Analysis Process (ALC-Layer 2)

This RATP layer refines the previous analysis process (ALC-Layer 1) to study how the ALC system observes the driving environment (target lane) in order to perform autonomous operations such as perception, decision-making and control. Specifically, we instantiated generic automation aspects discussed in Subsection 6.2.3 of Chapter 6 as follows:

TABLE 7.6: Safety requirements (SR1.1 and SR1.2) in formal model for ALC Layer 1

Safety requirements	Event-B elements
SR1.1	Guard (@grd5) in the <code>ALC.actuating</code> event, <code>@grd4: move(SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env + steering_angle_change)) ⊆ Lane</code>
SR1.2	Guard (@grd3) in the <code>Manual.actuating</code> event, <code>@grd2: move(SDV_POSITION_env ↦ manual_steering_angle)</code> <code>⊆ Lane</code>

- The perceived environmental features are defined as the desired path that the ALC system might identify based on the incoming images from the camera sensor.
- The detection score for identifying the left and right lane lines in the given images is specified as the confidence score.
- Based on the identified/desired path, the ALC system might either prompt a driver to intervene or specify a required steering angle change to keep an SDV inside the target lane.
- The newly identified steering angle would be actuated to allow the SDV to travel into a new position inside its target lane.

Each step of the RATP approach is then further detailed as follows:

Step 1. Instantiation: Based on the instantiated system component diagram (mentioned in Figure 5.3 of Chapter 5), the ALC system includes three main modules for performing the LKA operations (functions): 1) *perception*; 2) *decision*; and 3) *control*. Therefore, we assume that these functionalities are accomplished by the autonomous controller inside the ALC system.

The perception component of the ALC system employs a camera system to maintain the SDV within the target lane. It interprets images to identify a desired path based on detected lane markings and the confidence scores derived from the detection process. The decision component uses this desired path to adjust the steering angle, with the primary goal of keeping the SDV within the lane markings. It selects the target position and establishes the necessary adjustments in the steering angle. Subsequently, the control component actuates a modified steering angle to reach the selected/identified position, facilitating the autonomous driving of the SDV. In Figure 7.5, we refine a previous system boundary diagram (mentioned earlier in Figure 7.3) to capture these autonomous operations of an ALC system.

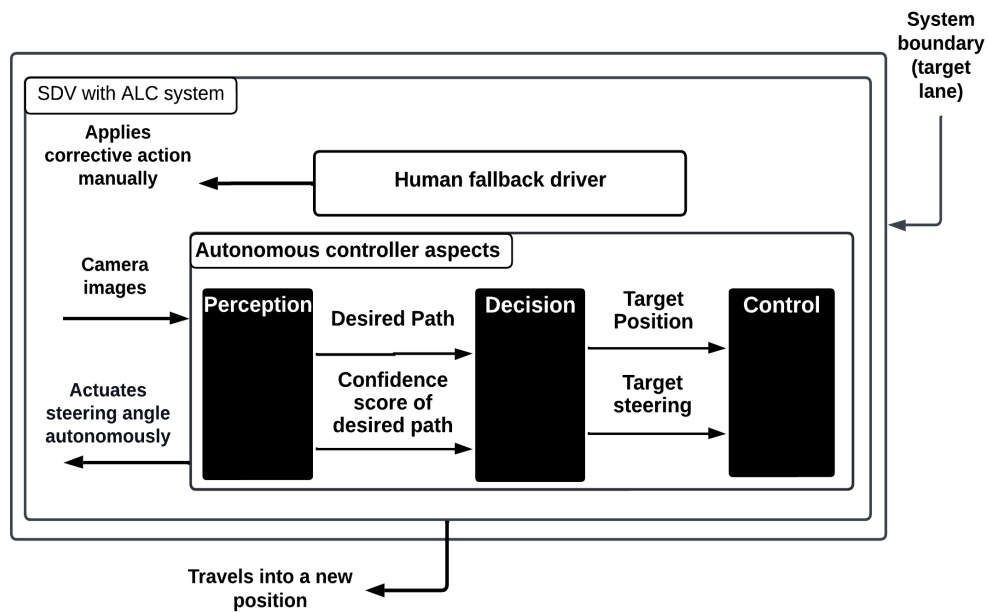


FIGURE 7.5: System boundary diagram in ALC Layer 2

Step 2. The purpose of analysis: The purpose of analysis is to study how the ALC system accomplishes its autonomous operations. A refined system loss can be driven from the previously identified safety requirement **SR1.1** as **SL2**:

- **SL2:** The autonomous controller of an ALC may actuate a steering angle autonomously to reach a new position outside of the target lane.

As the autonomous controller of an ALC system involves three main components (perception, decision and control), various SHs can be identified as follows:

- **SH2.1:** The perception component identifies the desired path that locates an SDV outside of its target lane.
- **SH2.2:** The decision component identifies a target position and steering angle based on the desired path; however, a steering angle modifies the physical (actual) steering angle of an SDV too quickly, which may cause unsafe reactions from the human driver.
- **SH2.3:** The control component may actuate a target steering angle that exceeds the SDV's power steering range.

A high-level safety constraint associated with these SHs is **SC2**:

- **SC2:** The ALC must perform its autonomous operations to keep an SDV inside its target lane.

Step 3. Refining control structure: A refined control structure is constructed according to how the ALC accomplished its autonomous operations, as shown in Figure 7.6. The autonomous controller has three components: *perception*, *decision* and *control*. The interactions (input/output) between these components are captured by using a set of CAs and FLs.

In the perception component, **F3** indicates that the ALC system must obtain an incoming image to identify the desired path. **CA4** and **CA5** indicate the possible system interactions between the perception and decision components when the autonomous controller identifies the desired path and its confidence score. **F4** covers a feedback loop between the perception and decision components when the ALC system may determine the desired path with a high/low confidence score.

In the decision component, **CA6** implies the responsibility of an ALC system to identify a target (new) position of an SDV. In addition, **CA7** indicates the responsibility of an ALC system for identifying the required change of steering angle to reach that new position. **F5** covers feedback loops between the decision and control components when a control component may accept/reject actuating the new steering.

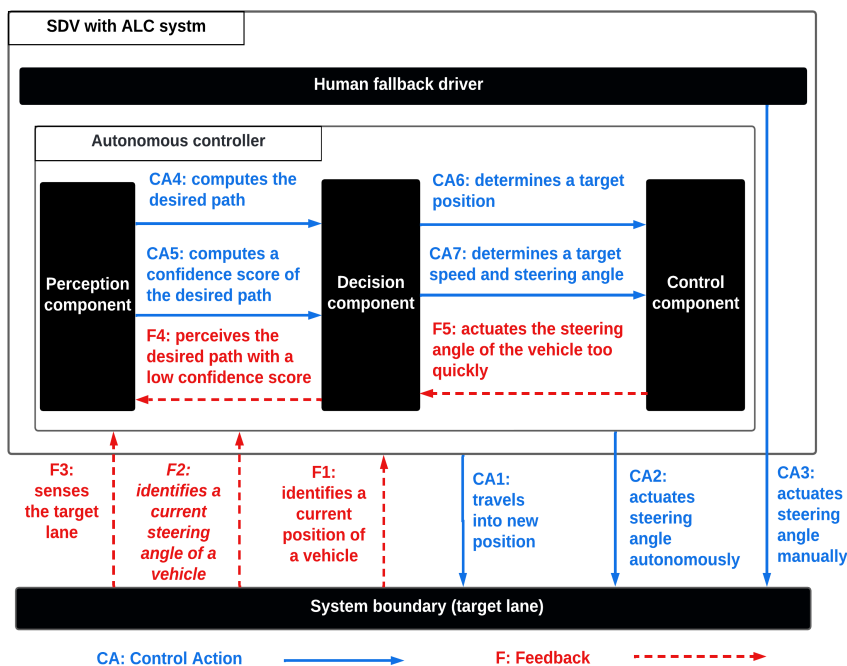


FIGURE 7.6: Control structure in ALC Layer 2

Step 4. Identifying unsafe control actions: At this layer, the UCAs of autonomous operations are demonstrated in three categories. The first category is unsafe operation, involving the perception component (**SH2.1**). The UCAs of a perception component are shown in Table 7.7. These UCAs are explained as follows:

- The autonomous controller may identify the desired path (**CA4**) without sensing the target lane (*UCA3.1*).
- The given/identified path (**CA4**) might not be updated (*UCA3.2*).
- Based on a low confidence score (**CA5**) of detecting a left/right lane lines, the identified path (**CA4**) might be recognised (*UCA3.3*).
- The sensing data (**F3**) used to identify the desired path are wrong (*UCA3.4*).

Therefore, the potential safety requirements associated with the perception component can be identified as follows:

- **SR2.1:** The perception component must update the sensing information (images) according to the current position of an SDV [**SH2.1**].
- **SR2.2:** The perception component must use an incoming image to identify the desired path for keeping an SDV inside the detected left and right lane lines in the target lane [**SH2.1**].
- **SR2.3:** The perception component must compute a confidence score regarding the detection process of a desired path [**SH2.1**].
- **SR2.4:** If the perception component identifies the desired path with a low confidence score, the ALC system issues a request to intervene [**SH2.1**].

TABLE 7.7: Unsafe control actions of perception component in ALC Layer 2

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA4 and CA5	<i>UCA3.1:</i> Autonomous controller may identify the desired path without sensing the target lane (F3).	<i>UCA3.2:</i> Desired path (CA4) does not update. <i>UCA3.3:</i> Desired path (CA4) identified according to a low confidence score (CA5).	<i>UCA3.4:</i> Desired path(CA4) identified according to wrong/late sensing data (F3).	N/A
Causal Factor				
Failure in the identification of a desired path.				

The second category is unsafe operation involved in the decision component. The UCAs of a decision component are shown in Table 7.8. These unsafe operations are explained as follows:

- Target position (**CA6**) may locate an SDV outside of the target lane (*UCA3.5*).

- Target steering angle (**CA7**) may actuate the steering angle of an SDV too quickly (*UCA3.6*).
- When a steering angle of an SDV is actuated too quickly (**F5**), the driver might fail to provide corrective action (*UCA3.7*).

Therefore, the potential safety requirements associated with the decision component (**SH2.2**) can be identified as follows:

- **SR2.5:** The decision-making component must identify a target position based on the identified path (desired path) [**SH2.2**].
- **SR2.6:** The decision-making component must determine a required change of steering angle based on the current (physical) steering angle and the identified (target) position [**SH2.2**].
- **SR2.7:** The decision-making component must not propose a target steering angle that actuates a steering angle of an SDV too quickly, in order to avoid poor human driver intervention [**SH2.2**].

TABLE 7.8: Unsafe control actions of decision component in ALC Layer 2

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA6 and CA7	<i>UCA3.5:</i> Autonomous controller may identify a target position (CA6) outside the target lane.	<i>UCA3.6:</i> Autonomous controller may identify a target steering angle (CA7) that modifies the steering angle of an SDV too quickly (F5).	<i>UCA3.7:</i> When autonomous controller modifies the steering angle of a vehicle too quickly (F5), a driver may not be able to provide a corrective action (CA3).	N/A
Causal Factor				
Unsafe identification of target position and steering angle.				

The third category is unsafe operation involved in the control component. These UCAs are demonstrated in Table 7.9 and are explained as follows:

- Target steering (**CA7**) cannot be actuated due to the steering limit of a power steering system involved in the SDV (*UCA3.8*).
- Target steering (**CA7**) exceeds the maximum and minimum steering limit (*UCA3.9*).

Therefore, the potential safety requirements associated with the control component (**SH2.3**) can be identified as follows:

- **SR2.8:** If a target steering angle exceeds the minimum steering angle, then the control module modifies the target steering angle to the minimum steering angle [SH2.3].
- **SR2.9:** If a target steering angle exceeds the maximum steering angle, then the control module modifies the target steering angle to the maximum steering angle [SH2.3].
- **SR2.10:** If the control component of an autonomous controller exceeds the steering angle range of an SDV, the ALC system issues a request to intervene [SH2.3].

TABLE 7.9: Unsafe control actions of control component in ALC Layer 2

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2	UCA3.8: Autonomous controller cannot actuate new steering (CA2).	UCA3.9: Autonomous controller may propose a target steering angle (CA7) outside the steering angle range of an SDV (F5).	N/A	N/A
Causal Factor				
Unsafe actuation of a target steering.				

Step 5. Refining Event-B model: Mapping is provided in Table 7.10 to demonstrate how new control actions (CA4, CA5, CA6, CA7) and feedback loops (F3, F4, F5) link with the representation of a refined machine. Full details of the refined machine (machine m2) can be seen in Appendix E.3.

TABLE 7.10: Control actions and feedback loops in formal model for ALC Layer 2

Control Action & Feedback loop	Event-B elements
CA4	Variable, <code>desirePath</code>
CA5	Variable, <code>confidenceScore</code>
CA6	Variable, <code>targetPosition</code>
CA7	Variable, <code>targetSteeringAngle</code>
F3	Variable, <code>IMAGE.env</code>
F4	Guard in <code>lowConfidenceScore.interven</code> event, <code>@grd3: confidenceScore < 80</code>
F5	Guards in <code>correct_exceeding</code> events, <code>@grd3: SDV_STEERING_ANGLE.env + steeringAngleChange > max.steering</code> and <code>@grd3: SDV_STEERING_ANGLE.env + steeringAngleChange < min.steering</code>

A constant function, `camera`, is introduced to capture the sensing information in multiple positions. The function is defined as follows:

$$\text{camera} \in \text{POSITION} \rightarrow \text{IMAGE}$$

In the context of the ALC system, it is crucial for the system to receive the incoming images on the basis of the SDV 's physical position (**SR2.1**). To ensure consistency, the following invariant is added.

$$\text{@consistency: IMAGE_env} = \text{camera}(\text{SDV_POSITION_env})$$

In the perception stage, two constant functions, `Seen_image` and `OEDR_task`, play a role in the computation performed by the autonomous controller to identify the desired path on the basis of the incoming images. These functions are defined as follows:

$$\begin{aligned} \text{Seen_image} &\in (\text{IMAGE} \times \text{LeftLane} \times \text{RightLane}) \rightarrow \text{Confidence_score} \\ \text{OEDR_task} &\in (\text{IMAGE} \times \text{LeftLane} \times \text{RightLane} \times \text{Confidence_score}) \rightarrow \mathbb{P}(\text{POSITION}) \end{aligned}$$

First, the function `Seen_image` interprets the incoming images and produces the confidence score for detecting the left and right lane lines. Second, the function `OEDR_task` uses the interpretations from the previous functions to identify the desired path. Overall, these functions contribute to the perception stage by analysing the sensory data and providing the necessary inputs for a subsequent decision component in the autonomous controller.

In the decision stage, two constant functions, `target_position` and `target_steering_angle`, take part in the computation performed by the autonomous controller to determine the target position and steering angle. These functions are defined as follows:

$$\begin{aligned} \text{target_position} &\in \text{Set}(\text{POSITION}) \rightarrow \text{POSITION} \\ \text{target_steering_angle} &\in \text{POSITION} \times \text{STEERING_ANGLE} \rightarrow \text{STEERING_ANGLE_CHANGE} \end{aligned}$$

First, the function `target_position` takes the set of positions derived from the perception component, which represents the desired path, and selects one position from that set as the target position. Second, the function `target_steering_angle` calculates the change in steering angle needed to reach the selected/target position.

The high-level operations of an ALC system are organised into five stages: `Perception`, `Decision`, `Control`, `Intervention` and `AutonomousDriving`. These stages are defined in variable `stage`, where the gluing invariant `gluing_inv` is added to ensure consistency between the status of an ALC system and the internal stages involved in the ALC system. For instance, this gluing invariant states that the ALC system might be in any stage when a system is activated (`ALC_Status = ON`).

$$\text{@gluing_inv: ALC_Status} = \text{ON} \Rightarrow \text{stage} \in \{\text{Perception}, \text{Decision}, \text{Control}, \text{AutonomousDriving}, \text{Intervention}\}$$

Boolean flag `signal` is added to demonstrate the movement of an SDV when either the steering angle of a system is manually or autonomously actuated. For instance, if the signal is `TRUE`, the new steering of a system is actuated to move the SDV into a new position.

To ensure that the actuation task of an autonomous controller maintains the safety of the system, i.e., $SDV_POSITION_env \in Lane$, an environmental invariant `Environment_consistency` is added as follows:

```
stage = AutonomousDriving  $\Rightarrow$  move(SDV_POSITION_env  $\mapsto$  targetSteeringAngle)  $\subseteq$  Lane
```

The relationship between the input and output of autonomous operations is modelled in sequential order, from the `Perception` stage into either the `AutonomousDriving` or `Intervention` stage. In the perception stage, the ALC abstractly identifies the desired path (`CA4`) and its confidence score (`CA5`) as follows:

```

event perception
any leftLane rightLane
when
  @grd1: leftLane  $\subseteq$  LeftLane
  @grd2: rightLane  $\subseteq$  RightLane
  @grd3: ALC_Status = ON
  @grd5: stage = Perception
then
  //image for current position
  @act1: IMAGE_env := camera (
    SDV_POSITION_env)
  //expected to identify these features
  @act2: leftLanePoints  $\in$  LeftLane
  @act3: rightLanePoints  $\in$  RightLane
  //confidence score for the detection process
  @act4: confidenceScore := Seen_image (
    IMAGE_env  $\mapsto$  leftLanePoints  $\mapsto$ 
    rightLanePoints)
  // estimates the desired path
  @act5: desirePath := OEDR_task ( IMAGE_env  $\mapsto$ 
    leftLanePoints  $\mapsto$  rightLanePoints  $\mapsto$ 
    confidenceScore  $\mapsto$  leadingVehicleSet)
  // change a stage of ALC to be in Decision
  @act6: stage := Decision
end
```

Based on the `perception` event, two invariants are added. These invariants are defined as follows:

```

//SR2.2: based on the detected lane lines and scores, the desired path is identified
@perceivedImage: stage = Perception  $\Rightarrow$  IMAGE_env  $\in$  ran(camera)  $\wedge$  (IMAGE_env  $\mapsto$ 
  leftLanePoints  $\mapsto$  rightLanePoints  $\mapsto$  confidenceScore)  $\in$  dom(OEDR_task)
//The desired path must be identified in perception's stage
@perceptionTask: stage = Decision  $\Rightarrow$  desirePath  $\in$  ran(OEDR_task)
```

While the stage of the ALC system is changed to `Decision`, a model can be divided into two stages: `Intervention` or `Control`. In the intervention stage, the ALC system will issue a request to intervene if the low confidence score (`F4`) is used to compute the desired path (`SR2.4`).

```

event lowConfidenceScore_interven
where
  @grd1: ALC_Status = ON
  @grd2: stage = Decision
  //denotes a low confidence score
  @grd3: confidenceScore < 80
then
  @act1: stage := Intervention
end
```

In the preparation of the **Control** stage, the decision component of ALC abstractly computes a target position (**CA6**) and a required change of steering angle (**CA7**) according to the desired path (**SR2.5**) and the physical (current) steering angle of an SDV (**SR2.6**).

```

//SR2.5: position obtained from desired path
@act1: targetPosition := target_position(
    desirePath)
event decision
when
@grd1: ALC_Status = ON
@grd2: stage = Decision
//high confidence score
@grd3: confidenceScore ≥ 80
then
//SR2.6: steering computed based on the target
    position and current steering
@act2: steeringAngleChange :=
    target_steering_angle(targetPosition ↦
    SDV_STEERING_ANGLE_env)
//stage of ALC changed to be in Control
@act3: stage := Control
end

```

Based on the **decision** event, two invariants, namely **targetPosition** and **changeSteer**, are introduced. These invariants are defined as follows:

```

// SR2.5: a target (new) position will obtain from the desired path
@targetPosition: stage = Decision ∧ confidenceScore ≥ 80 ⇒ desirePath ∈ dom(
    target_position)
//SR2.6: steering computed based on the target position and current steering
@changeSteer: stage = Control ∧ confidenceScore ≥ 80 ⇒ (targetPosition ↦
    SDV_STEERING_ANGLE_env) ∈ dom(target_steering_angle)

```

During the actuation of a target steering angle, the ALC system ensures that the necessary adjustments to the current steering angle do not exceed the maximum or minimum range of the SDV 's low-level components. The monitoring process for actuating a target steering angle can be broken into three categories:

- First is the **control** event, which initiates the actuation of a target steering and transitions the system state to **AutonomousDriving**.

```

event control
when
@grd1: ALC_Status = ON
@grd2: stage = Control
//new steering in the defined range
@grd3: SDV_STEERING_ANGLE_env +
    steeringAngleChange ∈
    STEERING_ANGLE
//a new position inside the target lane
@grd4: move(SDV_POSITION_env ↦
    targetSpeed ↦ (
    SDV_STEERING_ANGLE_env +
    steeringAngleChange)) ⊆ Lane
then
@act1: targetSteeringAngle :=
    steeringAngleChange +
    SDV_STEERING_ANGLE_env
@act2: stage :=
    AutonomousDriving
end

```


- Second is the `correct_steering` event, which adjusts the target steering angle to be within the minimum or maximum steering if the new steering angle falls outside of the range of the SDV 's power steering system. This event changes the stage of the ALC system to `Intervention` when such a violation occurs.

<pre> event correct_exceeding_min_steering when @grd1 : ALC_Status = ON @grd2 : stage = Control //new steering violates the minimum steering @grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange < min_steering then @act1 : stage := Intervention @act2 : targetSteeringAngle := min_steering end </pre>	<pre> event correct_exceeding_max_steering when @grd1 : ALC_Status = ON @grd2 : stage = Control //new steering violates the maximum steering @grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange > max_steering then @act1 : stage := Intervention @act2 : targetSteeringAngle := max_steering end </pre>
---	---

- Last is the `correct_out_of_lane` event, which detects an incorrect movement outside of the target lane and potentially adjusts the target steering to a minimum or maximum steering. If the system identifies such a deviation, it maintains the previous or current steering as the target steering and transitions the ALC system stage to `Intervention`.

<pre> event correct_out_of_lane where @grd1 : ALC_Status = ON @grd2 : stage = Control //new steering moving SDV outside the lane @grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange ∈ STEERING_ANGLE ⇒ move(SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env + steeringAngleChange)) ∉ Lane </pre>	<pre> then /* stage changed to be in ' Intervention'*/ @act1 : stage := Intervention @act2 : targetSteeringAngle := SDV_STEERING_ANGLE_env end </pre>
--	---

The `ALC_actuating` event is refined to setting a steering angle autonomously. A new steering angle is applied in order to move an SDV into a new position inside the target lane, where an Event-B *witness* is used for mapping between the abstract parameter and the new variable (`steeringAngleChange`) as follows:

```

event ALC_actuating refines
  ALC_actuating
where
  @grd1: ALC_Status = ON
  @grd2: steeringAngleChange ∈
    STEERING_ANGLE_CHANGE
  @grd3: steeringAngleChange +
    SDV_STEERING_ANGLE_env ∈
    STEERING_ANGLE
  @grd4: move(SDV_POSITION_env ↦ (
    SDV_STEERING_ANGLE_env +
    steeringAngleChange)) ⊆ Lane
  @grd5: stage = AutonomousDriving
  @grd6: steeringAngleChange +
    SDV_STEERING_ANGLE_env =
    targetSteeringAngle
  @grd7: signal = FALSE
then
  @act1: SDV_STEERING_ANGLE_env := (
    SDV_STEERING_ANGLE_env +
    steeringAngleChange)
  /* ready to move */
  @act2: signal := TRUE
with
  @steering_angle_change:
    steering_angle_change =
    steeringAngleChange
end

```

In any case of Intervention, the `Manual_actuating` event is extended in order to ensure the responsiveness of a human driver by providing a steering angle manually when the stage of the ALC system is in `Intervention`.

```

event Manual_actuating
extends Manual_actuating
where
  @grd3: stage = Intervention
  @grd4: signal = FALSE
then
  /* ready to move */
  @act2: signal := TRUE
end

```

Finally, the `move` event is extended to show how the SDV can move into a position inside the target lane, whether a new steering angle is actuated manually (`Intervention`) or autonomously (`AutonomousDriving`). In addition, the camera updates the incoming image according to the new position of the SDV.

```

event move extends move
where
  @grd3: signal = TRUE
  @grd4: stage ∈ {
    AutonomousDriving ,
    Intervention}
then
  @act2: stage := Perception
  @act3: IMAGE_env := camera(new_position)
  @act4: signal := FALSE
end

```

Table 7.11 shows how a formal model captures the safety requirements (`SR2.1` to `SR2.10`). In the refined model (machine `m2`) are several Assumptions (A) in the formal model:

- **A2.1:** The camera sensor always provides images for the current state of a target lane.
- **A2.2:** In order to identify the desired path, the current position of a vehicle is always in the lane, where the lane markings (left and right) are visible.

- **A2.3:** The autonomous controller can obtain an image from the sensor camera that is attached to an SDV.
- **A2.4:** The autonomous controller can make a judgement on the detection process of a desired path based on the confidence score. Therefore, we assume that if the confidence score is less than 80% the ALC must issue a request to intervene.
- **A2.5:** A human driver is always receptive when the ALC system issues a request to intervene.

TABLE 7.11: Safety requirements(SR2.1 and SR2.10) in formal model for ALC Layer 2

Safety requirements	Event-B elements
SR2.1	Invariant, @consistency: IMAGE_env = camera(SDV_POSITION_env)
SR2.2	Invariant, @perceivedImage: stage = Perception \Rightarrow IMAGE_env \in ran(camera) \wedge (Perception) (IMAGE_env \mapsto leftLanePoints \mapsto confidenceScore) \in dom(OEDR_task)
SR2.3	Action @act4 in the perception event @act4: confidenceScore := Seen_image(IMAGE_env \mapsto leftLanePoints \mapsto rightLanePoints)
SR2.4	Guard and action in lowConfidenceScore.interven event: @grd3: confidenceScore_ac < 80 @act1: stage_ac = Intervention
SR2.5	Invariant,@targetPosition: stage = Decision \wedge confidenceScore \geq 80 \Rightarrow desirePath \in dom(target_position) ,where targetPosition is computed as targetPosition := target_position(desirePath)
SR2.6	Invariant,@changeSteer: stage = Control \wedge confidenceScore \geq 80 \Rightarrow (targetPosition \mapsto SDV_STEERING_ANGLE_env) \in dom(target_steering_angle) ,where steeringAngleChange is computed as steeringAngleChange := target_steering_angle(targetPosition)
SR2.7	Invariant,@newSteer: stage = Control \wedge confidenceScore \geq 80 \Rightarrow steeringAngleChange \in ran (target_steering_angle) ,where steeringAngleChange is in defined range as steeringAngleChange \in min_steering_constraint .. max_steering_constraint
SR2.8 & SR2.9	Guards and actions in the correct_exceeding events: @grd3: SDV_STEERING_ANGLE_env + steeringAngleChange > max_steering @grd3: SDV_STEERING_ANGLE_env + steeringAngleChange < min_steering @act2: targetSteeringAngle := max_steering @act2: targetSteeringAngle := min_steering
SR2.10	Actions in the correct_exceeding events: @act1: stage := Intervention when the violation of the defined steering range is detected @grd3: SDV_STEERING_ANGLE_env + steeringAngleChange > max_steering @grd3: SDV_STEERING_ANGLE_env + steeringAngleChange < min_steering

7.1.4 Third Analysis Process (ALC-Layer 3)

This layer refines the previous analysis process (ALC-Layer 2) to investigate how the awareness level of a driver is involved in the autonomous operation of a system. Knowing the awareness level of the driver aims to ensure a response from the fallback driver when the ALC system issues a request to intervene. Specifically, we instantiated generic automation aspects discussed in Subsection 6.2.4 of Chapter 6 as follows:

- The human monitoring features that the DMS system might utilise to compute the awareness level of a driver are specified as the hands-on steering wheel and sensitive monitoring feature.
- The intervention request is defined as the warning messages that might be sent by the ALC system to prompt the driver to assume control of the SDV as needed.

Each step of the RATP approach is then further detailed as follows:

Step 1. Instantiation: The system component diagram (mentioned in Figure 5.3 of Chapter 5) demonstrates how the ALC system integrates with the DMS to ensure driver alertness through two particular features: *hands-on at the steering wheel* and *sensitive monitoring features*. It is assumed that the driver's awareness level is computed on the basis of the detection and computation of these features. In addition, intervention requests urging the driver to take control of the SDV may be issued by semi-automated systems such as the ALC. These requests are represented as warning messages, which are sent under various driving scenarios. In Figure 7.7 we refine a previous system boundary diagram to capture these operation of an ALC system.

Step 2. The purpose of analysis: This step is to study how the driver's awareness level is involved in the autonomous operation of a system when the ALC system may issue a request to intervene. Based on the system boundary diagram (mentioned in Figure 7.7), a refined system loss can be driven from the previous safety requirements (SR2.4 and SR2.10) as SL3:

- **SL3:** The human driver does not take the control of a vehicle when the ALC issues a request to intervene.

Because the human fallback controller is responsible for ensuring the reactions of a driver, the SHs associated with the system loss (SL3) are as follows:

- **SH3.1:** The human driver does not pay attention to the autonomous operations of an ALC system.

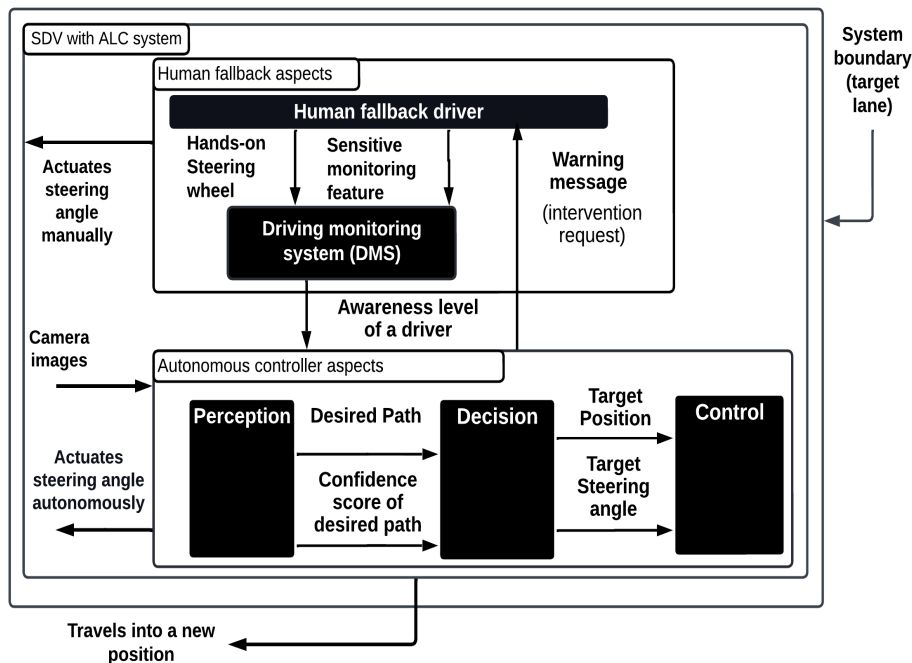


FIGURE 7.7: System boundary diagram in ALC Layer 3

- **SH3.2:** The autonomous controller of an ALC system issues a request to intervene while the awareness level of the driver is unknown.

A high-level safety constraint (SC) that satisfies the system conditions to prevent **SH3.1** and **SH3.2** is **SC3**:

- **SC3:** The driver must pay attention to the autonomous operation of a system when the autonomous controller is moving the vehicle autonomously.

Step 3. Refining control structure: The control structure is refined to demonstrate how the human fallback controller ensures driver awareness, as in Figure 7.8. In this refined structure, the DMS is incorporated to illustrate the computation of a driver's awareness level. For instance, **CA9** and **CA10** highlight the driver's responsibility to provide hands on steering wheel, also a sensitive monitoring feature, thereby enabling the DMS to ensure the driver's awareness level. In circumstances where the driver fails to provide these features, the DMS would be unable to verify the driver's awareness level, leading to immediate deactivation of the ALC. Furthermore, **CA8** points to the responsibility of the DMS to share the driver's awareness level with the ALC system. Based on **CA8**, the autonomous controller might restrict or initiate its autonomous operation. Moreover, **F6** and **F7** illustrate two scenarios where the ALC might issue an intervention request:

1. **F6:** The ALC system is unsure whether the identified/desired path can keep the SDV inside its target lane or might cause it to move outside of its target lane.
2. **F7:** The ALC system attempts to exceed the maximum or minimum range of the SDV 's power steering system.

In these driving situations the driver needs to take control of the SDV. As a result, the ALC system swiftly initiates the SDV 's transition to manual driving mode.

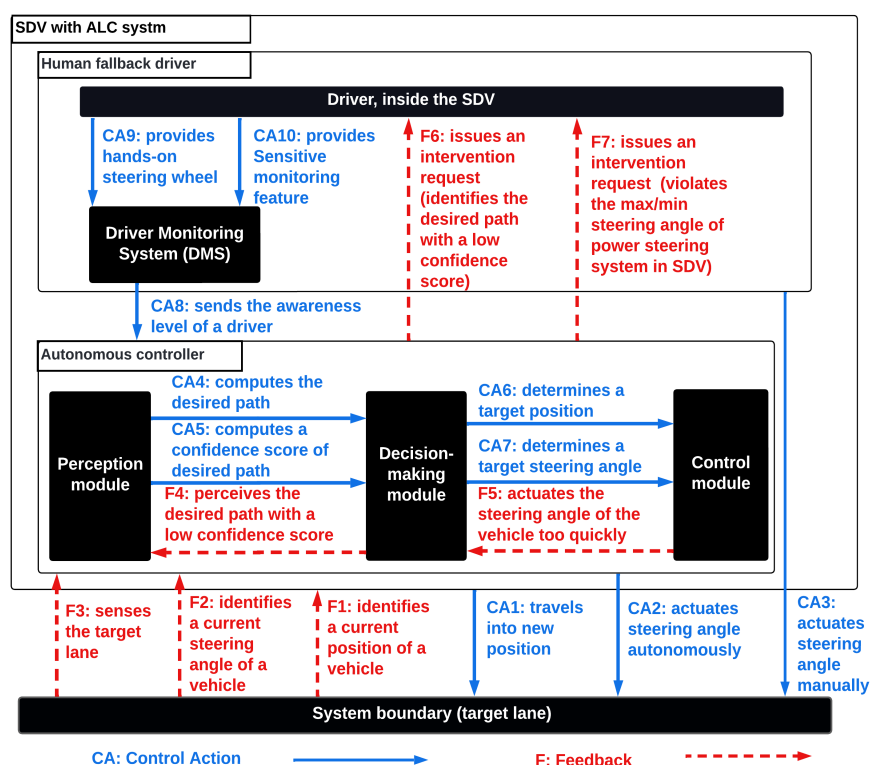


FIGURE 7.8: Control structure in ALC Layer 3

Step 4. Identifying unsafe control actions: At this layer, the UCAs fall into two categories. The first comprises unsafe operation of the DMS (**SH3.1**), as presented in Table 7.12 and explained below.

- The autonomous controller performs its operations, such as perception, while the awareness level of the human fallback driver is unverified by the DMS (*UCA 4.2*).
- The ALC may function before the DMS verifies the awareness level of the human fallback driver (*UCA 4.3*).

As a result, the potential SRs related to the DMS component can be identified as follows:

- **SR3.1:** To activate the ALC system, the DMS must ensure the awareness level of the human fallback driver [**SH3.1**].

TABLE 7.12: Unsafe control actions of DMS component in ALC Layer 3

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA8	UCA4.1: Autonomous controller performs its operations while the awareness level of a driver is unknown.	UCA4.2: Autonomous controller performs its operations while the awareness level of a driver is unverified by the DMS (CA8).	UCA4.3: Autonomous controller performs its operations before the DMS verifies the driver's awareness level (CA8).	N/A
Causal Factor				
A human driver is unaware of autonomous operations.				

- **SR3.2:** The DMS must compute the awareness level of a driver based on hands on steering wheel, also a sensitive monitoring feature [SH3.1].
- **SR3.3:** If the driver's alertness is not indicated by means of hands on steering wheel or a sensitive monitoring feature, the ALC system is immediately deactivated [SH3.1].

The second category includes unsafe operation of a system when a request to intervene is sent (SH3.2), as shown in Table 7.13 and explained as follows:

- The intervention request might be sent when the awareness level of the human fallback driver is unknown (UCA 4.5).
- The autonomous controller relies on the corrective action of the human fallback driver taken while their awareness level is unknown (UCA 4.6).

TABLE 7.13: Unsafe control actions of driver intervention in ALC Layer 3

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA7	UCA4.4: Autonomous controller performs its operations while the awareness level of a driver is unaware.	UCA4.5: Autonomous controller issues a request to intervene (F6,F7) when the status of the human fallback driver is unaware (CA8).	UCA4.6: Autonomous controller relies on the correction action of the human fallback driver (CA3) while their awareness level is unknown (CA8).	N/A
Causal Factor				
Unknown reactions of a fallback driver when a semi-automated system issues a request to intervene.				

Therefore, the potential safety requirements associated with SH3.2 can be identified as follows:

- **SR3.4:** The human fallback controller must share the driver’s level of awareness with the autonomous controller [SH3.2].
- **SR3.5:** If the autonomous controller system issues a request to intervene, the driver is responsible for performing the entire driving task (manual driving) [SH3.2].

Step 5. Refining Event-B model: Mapping is provided in Table 7.14 to show how new control actions (CA8, CA9, CA10) and feedback loops (F6, F6) link to the representation of a formal model. Full details of the refined machine (machine m3) can be seen in Appendix E.4. We modelled the awareness level of a driver and the intervention request (warning messages) using Booleans as follows:

```
@typeof-awarenessLevel: awarenessLevel ∈ BOOL
@typeof-warining: warningMessage ∈ BOOL
@typeof-handsOnSteeringWheel: handsOnSteeringWheel ∈ BOOL
@typeof-sensitiveMonitoringFeature: sensitiveMonitoringFeature ∈ BOOL
```

TABLE 7.14: Control actions and feedback loops in formal model for ALC Layer 3

Control Action & Feedback loop	Event-B elements
CA8	Variable, awarenessLevel
CA9	Variable, handsOnSteeringWheel
CA10	Variable, sensitiveMonitoringFeature
F6 & F7	Variable, warningMessage

A Boolean variable awarenessLevel (CA8) is introduced to compute the awareness level of a driver (SR3.2) on the basis of how the DMS detects the handsOnSteeringWheel (CA9) and a sensitiveMonitoringFeature (CA10); therefore, a new invariant is written, as follows;

```
@compute_awarenesslevel: awarenessLevel = TRUE ⇒ (handsOnSteeringWheel = TRUE ∧
sensitiveMonitoringFeature = TRUE)
```

In DMS operations, DMS_hands_on_wheel and DMS_detect_sensitiveMonitoringFeature events serve to model the detection of the human monitoring features when a driver’s alertness is indicated by either hands on steering wheel or a sensitive monitoring feature, as follows:

```
event DMS_hands_on_wheel
where
@grd1: ALC_Status=ON
@grd2: awarenessLevel=FALSE
@grd3: handsOnSteeringWheel=FALSE
then
@act1: handsOnSteeringWheel:=TRUE
@act3: awarenessLevel:=bool(
sensitiveMonitoringFeature=TRUE)
end

event DMS_detect_sensitiveMonFeature
where
@grd1: ALC_Status=ON
@grd2: awarenessLevel=FALSE
@grd3: sensitiveMonitoringFeature=
FALSE
then
@act1: sensitiveMonitoringFeature:=
TRUE
@act3: awarenessLevel:=bool(
handsOnSteeringWheel=TRUE)
end
```


In addition, the `DMS_hands_off_wheel` and the `DMS_lost_sensitiveMonitoringFeature` events model the possible changes in the status of either *the hands-on steering wheel* or *a sensitive monitoring feature*; i.e., driver does not provide the hands-on steering wheel. Therefore, the awareness level of a driver is computed according to these changes. Because the awareness level of a driver is one of the preconditions for achieving safe operation of the semi-autonomous systems [101], the ALC system must switch off its functionalities if drivers fail to prove their awareness level (**SR3.3**). Therefore, both events are extended from the `ALC_OFF` event in order to switch off the system safely, as follows:

<pre> event DMS_hands_off_wheel extends ALC_OFF where @grd3: awarenessLevel = TRUE @grd4: handsOnSteeringWheel = TRUE then @act3: awarenessLevel := FALSE @act4: warningMessage := FALSE @act5: handsOnSteeringWheel := FALSE @act6: sensitiveMonitoringFeature := FALSE end </pre>	<pre> event DMS_lost_sensitiveMonitoringFeature extends ALC_OFF where @grd3: awarenessLevel = TRUE @grd4: sensitiveMonitoringFeature = TRUE then @act3: awarenessLevel := FALSE @act4: warningMessage := FALSE @act5: sensitiveMonitoringFeature := FALSE @act6: handsOnSteeringWheel := FALSE end </pre>
--	--

Moreover, we added a new invariant (`@send_req`) to ensure that the `warningMessage` is sent if the stage of ALC system is in Intervention (**SR3.5**).

```
@send_req: warningMessage = TRUE ⇒ stage = Intervention
```

Based on the intervention cases in the previous machine (`m2`), the ALC system may issue a request to intervene in three driving scenarios:

1. If the perception module of an autonomous controller identifies the desired path with a low confidence score (**F6**), modelled in event `lowCofidanceScore_interven`.
2. If the control module of an autonomous controller attempts to exceed the steering angle range of an SDV (**F7**), modelled in the events `correct_exceeding_max_steering` and `correct_exceeding_min_steering`.
3. If the ALC system specifies a change in the steering angle that would move an SDV into a new position outside of the target lane, modelled in event `correct_out_of_lane`.

Therefore, we added a new invariant (`@interv_cases`) for issuing a request to intervene in these three driving scenarios (**SR3.5**).

`@interv_cases: warningMessage = TRUE ⇒ confidenceScore < 80 ∨
SDV_STEERING_ANGLE_env + steeringAngleChange ∉ STEERING_ANGLE ∨ move(
SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env + steeringAngleChange)) ∉ Lane`

In order to ensure that a driver is receptive to these intervention scenarios, we added a new invariant (`@driver_aware`) to verify their awareness level when the ALC performs its operations (**SR3.1**). Therefore, the DMS must verify the awareness level of a driver when the ALC system is at any stage, such as `perception` etc.

`@driver_aware: awarenessLevel = TRUE ⇒ stage ∈ {Perception , Decision , Control ,
Intervention , AutonomousDriving}`

Table 7.15 shows how the safety requirement (**SR3.1** to **SR3.5**) has been satisfied in a formal model. In the refined model (machine `m3`), two Assumptions (A) are involved in a formal model:

- **A3.1:** A human driver is receptive to any intervention requests, where the ALC is immediately turned off (manual driving).
- **A3.2:** In order to compute the awareness level of a driver, we assume that in DMS events there is an attached sensor that can monitor changes in human monitoring, whether by hands on the steering wheel or a sensitive monitoring feature.

TABLE 7.15: Safety requirements (**SR3.1** to **SR3.5**) in formal model for ALC Layer 3

Safety requirements	Event-B elements
SR3.1	Invariant, <code>@driver_aware: awarenessLevel = TRUE ⇒ stage ∈ STAGE</code>
SR3.2	Invariant, <code>@awarenesslevel: awarenessLevel = TRUE ⇒ (handsOnSteeringWheel = TRUE ∧ sensitiveMonitoringFeature = TRUE)</code>
SR3.3 & SR3.5	New guards in autonomous events, such as <code>@grd5</code> in <code>perception</code> event , <code>@grd5: awarenessLevel = TRUE</code>
SR17	An invariant for issuing intervention requests, <code>@interv_cases: warningMessage = TRUE ⇒ confidenceScore < 80 ∨ (SDV_STEERING_ANGLE_env + steeringAngleChange) ∉ STEERING_ANGLE</code> New guards and actions in the <code>Manual_actuating</code> event to ensure the responsiveness of a driver when the ALC issues a request to intervene <code>@grd6: warningMessage = TRUE</code> <code>@act3: warningMessage := FALSE</code>

7.2 Conclusion

This chapter explored the application of the RATP to the ALC system, with specific emphasis on its LKA and DMS features. Through a systematic application of the RATP approach we exemplified how, during autonomous operation, the ALC system autonomously controls the steering angle, concurrently maintaining driver alertness via

the DMS. The analysis was conducted in four layers, starting from a high-level analysis layer and moving towards a more detailed, concrete layer. The following chapter focuses on how the ALC system autonomously sets both the speed and steering of a SDV.

Chapter 8

LKA and ACC Functions in the ALC System

In the previous chapter the Automated Lane Centring (ALC) system was examined through the application of the Rigorous Analysis Template Process (RATP), which was deployed across four layers. The main focus was on the Lane Keeping Assist (LKA) and Driver Monitoring System (DMS) functions, based on five systematic steps. However, it is important to note that the ALC system also integrates the Adaptive Cruise Control (ACC) function, which modulates the speed of an Self-Driving Vehicle (SDV) to maintain a safe distance from leading vehicles. Thus, this chapter extends the scope to incorporate the ACC function within the ALC system. The aim is to elucidate how the ALC autonomously controls both steering and speed to keep the SDV within its target lane, a process initially summarised in Section 4.2.2 of Chapter 4.

Similar to the previous chapter, this work contributes to research question (RQ4):

***RQ4:** To what extent does the RATP methodology demonstrate its utility when applied to a cases of varying sizes and complexity, especially those involving interactions between human drivers and an SDV system?*

In line with **RQ4**, the ALC layers will be modified to include the ACC functionality within the scope of analysis. In Section 8.1, the changes that were made to include the ACC feature in the ALC layers are explained. Section 8.2 elucidates how the RATP methodology addresses **RQ4** with regard to the ALC case study. Section 8.3 provides the evaluation and review of RATP application to the ALC case study. Section 8.4 presents a conclusion of this chapter.

8.1 Adapting ALC Layers for ACC Functionality

The ALC layers initially described in Section 7.1 of Chapter 7 required modifications to incorporate additional automation aspects, specifically those related to the ACC function. These modifications include:

- Radar reading points, which denote the sensor information used to identify potential leading vehicles.
- The target speed, representing the speed that is autonomously determined.
- The application of both target speed and steering angle to move the SDV autonomously.

Once these aspects are integrated, the ALC layers were updated to demonstrate how the ALC system autonomously handles both speed and steering. The subsequent sections give a brief summary of these modifications to the ALC layers, according to the steps of the RATP approach. Please note that we discuss only those changes to the ALC layers developed in Section 7.1 of Chapter 7.

8.1.1 Instantiating System Boundary Diagram

The system boundary diagram was updated, as illustrated in Figure 8.1. This diagram now includes the radar reading points, the potential leading vehicle, the specification of a target speed and the actuation of both speed and steering to guide the SDV to a new position.

The perception component initially interpreted *the radar reading points* to identify the position of a leading vehicle. Following this, the decision component decided on *a target speed*, aiming to maintain a safe distance from any potential leading vehicles. Finally, the control component *actuated the adjusted speed and steering angle* to reach the intended position, thus enabling autonomous driving of the SDV.

8.1.2 Identifying the Purpose of Analysis

The objective is to encompass the influence of speed identification on the performance of the ALC system. Consequently, the previously identified System Losses (SLs) and System Hazards (SHs) were revisited to include the aspect of autonomous speed identification.

The new SLs based on the potential manipulations of speed are as follows:

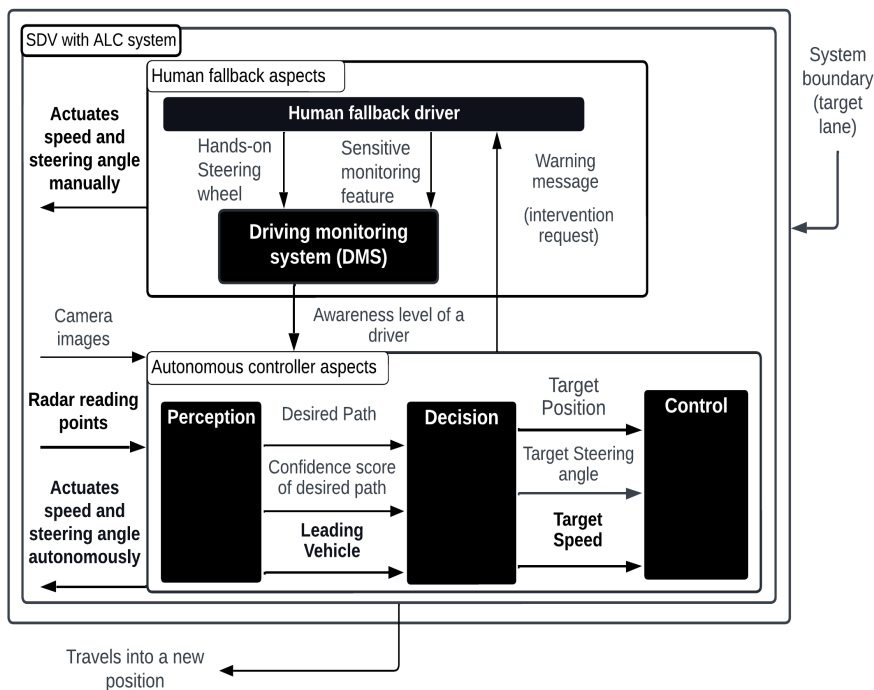


FIGURE 8.1: System boundary diagram incorporating speed identification

- **SL1:** A newly determined speed and steering angle could lead an SDV to a position outside of its target lane.
- **SL2:** The autonomous controller of an ALC may autonomously actuate a speed that leads an SDV to a position outside of the target lane.

Based on these new SLs, the SHs were identified as follows:

- **SH1:** The speed of an SDV, controlled by either a human driver or the ALC system, could potentially guide an SDV to a position outside of its target lane.
- **SH2:** The perception component may identify a desired path that violates the safe distance from a potential leading vehicle.
- **SH3:** The control component may actuate a target speed that violates the safe distance from a leading vehicle.

A high-level safety constraint associated with these SHs is **SC1**:

- **SC1:** The ALC must identify and actuate the target speed to keep an SDV within its target lane while maintaining a safe distance from any leading vehicles.

8.1.3 Updating Hierarchical Control Structure

The control structure for the ALC system has been revised to incorporate modifications essential for the autonomous control of the SDV 's speed. These changes are illustrated in Figure 8.2 and explained as follows:

- The sensing data (**F3**) includes both the incoming images and radar reading points.
- A new control action (**CA11**) is added to demonstrate how the perception component does interpret the radar reading points (**F3**) to detect any leading vehicles.
- A new feedback loop (**F8**) is established between the decision and control components to monitor the identification of the target speed (**CA7**), ensuring that it does not violate the safe distance from any potential leading vehicles.
- An additional feedback loop (**F9**) signals intervention requests when violation of a safe distance from a leading vehicle is detected (**F8**).
- The control action (**CA2**) illustrates both the speed and the steering angle that would be used to navigate an SDV autonomously into its target lane.

These adjustments were made to facilitate the investigation of how the ALC system autonomously identifies and actuates an SDV 's speed.

8.1.4 Identifying Unsafe Control Actions

New Unsafe Control Actions (UCAs) were introduced to emphasise potential risks when autonomously determining and adjusting the speed of an SDV. These UCAs are in three categories. The first highlights incorrect manipulations of the speed control variables (**SH1**) and is detailed in Table 8.1. These UCAs are as follows:

- *UCA1* illustrates a scenario where the speed of an SDV is autonomously adjusted to reach a position within the target lane but the detailed specifications of the speed variable are not provided.
- *UCA2* represents a scenario where the SDV 's speed is changed to reach a position outside of the target lane.

To mitigate these identified risks, the Safety Requirements (SRs) were identified as follows:

- **SR1:** The ALC system must autonomously control the speed to keep the SDV within the target lane [**SH1**].

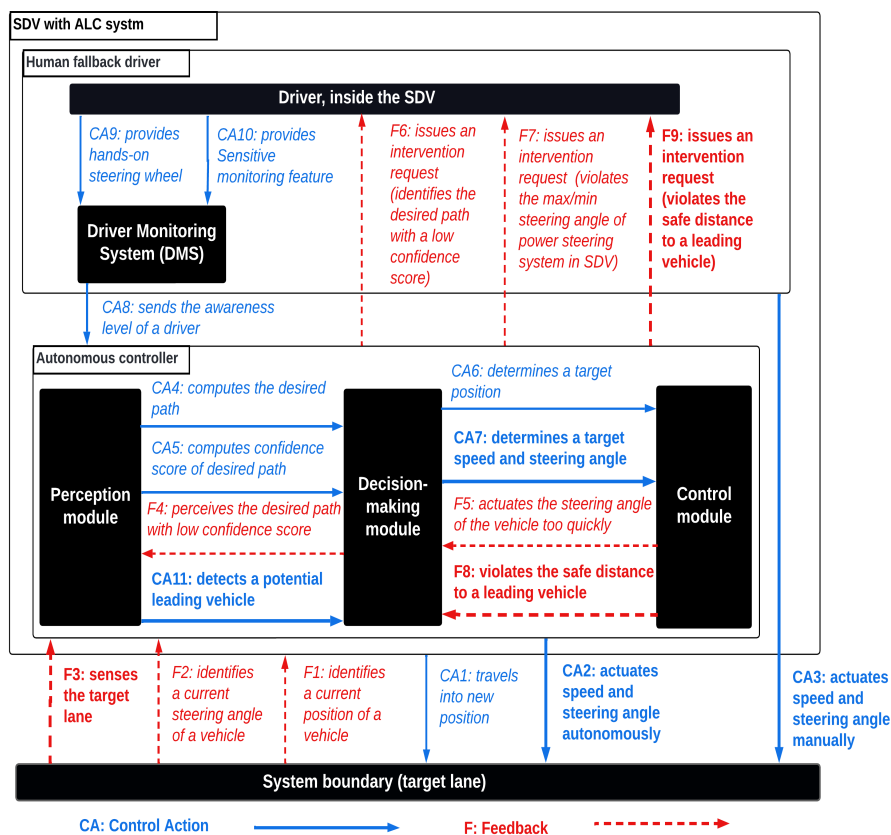


FIGURE 8.2: Control structure incorporating speed identification

- **SR2:** A human driver should be able to specify the target speed that the ALC system aims to achieve [SH1].

TABLE 8.1: Unsafe control actions in speed variable of ALC system

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2	UCA1: The ALC system moves a vehicle into a new position inside the target lane (CA1), but the specifications of speed are missing.	UCA2: actuates speed variable autonomously (CA2) to reach a position outside the target lane.	N/A	N/A
Causal Factor				
Wrong adjustment of the actuating variables (speed).				

In a similar manner, the second category covers the UCAs that lead to the SH2, where the perception component might identify the desired path without considering the potential leading vehicle. These UCAs are detailed in Table 8.2. Specifically:

- *UCA3* refers to the scenario in which the identified desired path does not take into account any potential leading vehicle in the target lane.
- *UCA4* indicates the case in which the desired path is calculated without considering the radar reading points that could identify any potential leading vehicles.

To mitigate these risks, the following safety requirements are suggested:

- **SR3:** The perception component must leverage the incoming radar reading points to detect any potential leading vehicle within the identified/detected desired path [**SH2**].

TABLE 8.2: New unsafe control actions in the perception component

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA4	<i>UCA3:</i> Autonomous controller may identify the desired path without considering the potential leading vehicle (CA11).	<i>UCA4:</i> Desired path (CA4) does not take into account the radar reading points (F3).	N/A	N/A
Causal Factor				
Failure in the identification of a desired path.				

Last, the third category highlights the UCAs contributing to **SH3**, where the control component might unsafely actuate the target speed. These UCAs are detailed in Table 8.3 and comprise:

- *UCA5* outlines the situation in which the target or identified speed cannot be actuated.
- *UCA6* refers to the circumstance in which the actuation of the target speed may violate the safe distance from a potential leading vehicle.

To mitigate these potential issues, the subsequent SRs are proposed as follows:

- **SR4:** The decision component must suggest a target speed in alignment with the maximum speed set by the human driver [**SH3**].
- **SR5:** If the target speed could cause violation of a safe distance to a leading vehicle, the control component should reduce the speed of the SDV [**SH3**].
- **SR6:** If the control component of an autonomous controller violates the safe distance to a leading vehicle, the ALC system issues a request to intervene [**SH3**].

TABLE 8.3: New unsafe control actions in the control component

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA2	<i>UCA5</i> : Autonomous controller cannot actuate new speed (CA2).	<i>UCA6</i> : Autonomous controller may propose a target speed (CA7) that violates a safe distance from a potential leading vehicle (F8).	N/A	N/A
Causal Factor				
Unsafe actuation of speed.				

8.1.5 Updating the Event-B Model

Mapping is provided in Table 8.4 to demonstrate how control actions (**CA2**, **CA11**) and feedback loops (**F3**, **F8**, **F9**) link to the representation of the formal models. Full details of the formal models are in Appendix F. However, the discussion here is centred primarily on the formalism used for the autonomous identification of speed.

TABLE 8.4: Control actions and feedback loop incorporating speed identification

Control Action & Feedback loop	Event-B elements
CA2	Variable, <i>SDV_SPEED_env</i>
CA11	Variable, <i>leadingVehicleSet</i>
F3	Variable, <i>RADAR_reading_env</i>
F8	Guard in <i>safe_distance_violation</i> event, <i>@grd1:distance < SAFE.DISTANCE</i>
F9	Variable, <i>warningMessage</i>

The actual speed of an SDV is represented by the *SDV_SPEED_env* variable. This variable comes with two constants, *max_speed* and *min_speed*, which specify the specified range for the speed variables.

```
@def-speed: SPEED = min_speed .. max_speed
@typeof-speed: SDV_SPEED_env ∈ SPEED
```

Taking into account the possibility that a human driver sets the target speed for the ALC system (**SR2**), the SDV's target speed is represented by the *ACC_target_speed* variable. Therefore, the *ALC_on* event is extended to include the potential target speed when operation of the ALC system is initiated.

```

event ALC_ON extends
ALC_ON
any
//Driver may specify a target speed
tsp
where
@grd2: tsp ∈ SPEED
then
@act2: ACC_target_speed := tsp
end

```

A constant function `radar` has been introduced to capture the sensing information (F3) in multiple positions. This function is defined as follows:

```
radar ∈ POSITION → RADAR_READING
```

In the context of the ALC system, it is crucial for the system to receive the incoming radar points on the basis of the SDV 's physical position (CA1). Therefore, the following invariant was added:

```
@consistency_2: RADAR_reading_env = radar(SDV_POSITION_env)
```

The leading vehicle could be seen as the vehicle that is located in the same target lane as the SDV. However, the ALC system can operate even without a leading vehicle; therefore, we modelled the leading vehicles as a variable `leadingVehicleSet`, defined as follows:

```

// there could or couldn't be a leading vehicle
LEADING_VEHICLE = {x · x ∈ Lane | {x}} ∪ {∅}
// defined a leading vehicle as variable
leadingVehicleSet ∈ LEADING_VEHICLE

```

A constant function `Seen_radar_reading` is used by the autonomous controller to identify a potential leading vehicle. This function is defined as a partial function, signifying the method by which the ALC system interprets radar reading points to detect a possible leading vehicle within the target lane:

```

//partial function enables a system to optionally detect a leading vehicle.
Seen_radar_reading ∈ (RADAR_READING) → Lane

```

Considering the radar reading points, the identification of the desired path was updated to accommodate the potential position of a leading vehicle:

```

//identification of the desired path considers the position of a leading vehicle
OEDR_task ∈ ( LeftLane × RightLane × Confidence_score × LEADING_VEHICLE ) → P(
POSITION)

```

A constant function `target_speed` takes part in the computation by the autonomous controller to determine the target speed. This function is defined as follows:

```
target_speed ∈ POSITION × SPEED × LEADING_VEHICLE → SPEED
```

The movement of an SDV into a new position is modelled as a constant `move`, where modifications to both speed and steering angle enable the SDV to reach new positions. The definition of the `move` function is shown below:

$$\text{move} \in \text{POSITION} \times \text{SPEED} \times \text{STEERING_ANGLE} \rightarrow \mathbb{P}_1(\text{POSITION})$$

To maintain the system safety during an actuation task by an autonomous controller, whereby the SDV remains within the lane ($\text{SDV_POSITION_env} \in \text{Lane}$), an environmental invariant, *Environment consistency*, was adjusted. This adjustment takes into account the autonomous determination of both speed and steering angle needed to reach a new position within the target lane (**SR1**), as shown below:

$$\text{@Environment-consistency: stage} = \text{AutonomousDriving} \Rightarrow \text{move}(\text{SDV_POSITION_env} \mapsto \text{targetSpeed} \mapsto \text{targetSteeringAngle}) \subseteq \text{Lane}$$

In the perception stage, the identification of a possible leading vehicle was modelled as follows:

```

event perception
any leftLane rightLane
when
  @grd1: leftLane  $\subseteq$  LeftLane
  @grd2: rightLane  $\subseteq$  RightLane
  @grd3: ALC_Status = ON
  @grd5: stage = Perception
then
  @act1: IMAGE_env := camera (
    SDV_POSITION_env)
  @act2: RADAR_reading_env := radar (
    SDV_POSITION_env)
  @act3: leftLanePoints  $\in$  LeftLane
  @act4: rightLanePoints  $\in$  RightLane
  @act5: confidenceScore := Seen_image(
    IMAGE_env  $\mapsto$  leftLanePoints  $\mapsto$ 
    rightLanePoints)
  //SR3: potential leading vehicle
  @act6: leadingVehicleSet :=
    Seen_radar_reading[ {
    RADAR_reading_env} ]
  //SR3: estimates the desired path
  @act7: desirePath := OEDR_task(
    leftLanePoints  $\mapsto$  rightLanePoints  $\mapsto$ 
    confidenceScore  $\mapsto$  leadingVehicleSet
    )
  @act8: stage := Decision
end

```

Based on the *perception* event, a new invariant *LeadingV* was added as follows:

```

// indicates that the ALC system might identify a leading vehicle
@LeadingV: leadingVehicleSet  $\neq \emptyset \Rightarrow (\exists x \cdot \text{leadingVehicleSet} = \{x\})$ 

```

In the decision stage, the ALC decides the new speed (**CA7**), including identification of steering angle and the target position, based on the identified path.

```

event decision
when
  @grd1: ALC_Status = ON
  @grd2: stage = Decision
  @grd3: confidenceScore  $\geq$  80
then
  @act1: targetPosition := target_position(
    desirePath)
  @act2: steeringAngleChange :=
    target_steering_angle(targetPosition
     $\mapsto$  SDV_STEERING_ANGLE_env)
  @act3: targetSpeed := target_speed(
    targetPosition  $\mapsto$  ACC_target_speed  $\mapsto$ 
    leadingVehicleSet)
  @act4: stage := Control
end

```

Moreover, the `ALC_actuating` event has been divided into `ALC_actuating_with_LV` and `ALC_actuating_without_LV`. These events autonomously set the SDV 's target speed and steering angle. For instance, the `ALC_actuating_with_LV` event calculates the distance to a leading vehicle, and is designed for scenarios where there is a leading vehicle in the target lane.

```

event ALC_actuating_with_LV refines
ALC_actuating any
SDV_lon SDV_lat
LV_lon LV_lat
where
// leading vehicle set not empty
@mainGuard: leadingVehicleSet ≠ ∅
@grd1: targetPosition = SDV_lon ↦
    SDV_lat
@grd2: leadingVehicleSet = {LV_lon ↦
    LV_lat}
@grd3: ALC_Status = ON
@grd4: steeringAngleChange ∈
    STEERING_ANGLE_CHANGE
@grd5: steeringAngleChange +
    SDV_STEERING_ANGLE_env ∈
    STEERING_ANGLE
@grd6: targetSpeed ∈ min_speed ..
    ACC_target_speed
@grd8: move(SDV_POSITION_env ↦
    targetSpeed ↦ (
    SDV_STEERING_ANGLE_env +
    steeringAngleChange)) ⊆ Lane

@grd9: stage = AutonomousDriving
@grd10: steeringAngleChange +
    SDV_STEERING_ANGLE_env =
    targetSteeringAngle
@grd11: signal = FALSE
then
@act1: SDV_STEERING_ANGLE_env := (
    SDV_STEERING_ANGLE_env +
    steeringAngleChange)
@act2: SDV_SPEED_env := targetSpeed
// ready to move
@act3: signal := TRUE
// distance to leading vehicle
@act4: distance := LV_lon - SDV_lon
with
@steering_angle_change:
    steering_angle_change =
    steeringAngleChange
@sp: sp = targetSpeed
end

```

In driving scenarios that involve a leading vehicle, the ALC system must ensure maintenance of a safe distance. To address potential violations of this constraint, the `safe_distance_violation` event was introduced. If the ALC system breaks the safe distance parameter, this event reduces the SDV 's speed and prompts an intervention request. The modelling of this event is as follows:

```

event safe_distance_violation
where
// violates the safe distance to a leading
    vehicle
@grd1: distance < SAFE_DISTANCE
@grd2: signal = TRUE
then
    /*the stage of ALC changed to be '
    Intervention'*/
@act1: stage := Intervention
    /*function to assume a system would
    reduce the speed*/
@act2: targetSpeed := speed_reduced
    (SDV_SPEED_env)
@rest_distance: distance := 0
end

```

The movement of an SDV is divided into two major categories. The first refers to autonomous movement, which includes two events: `auto_move_with_LV` and `auto_move`

`_without_LV`. These events enable the SDV to move towards a target position by actuating and using the target speed and steering angle. For instance, the `auto_move_with_LV` event is modelled as follows:

```

event auto_move_with_LV
refines auto_move any
LV_lon LV_lat
target_SDV_lon
target_SDV_lat
where
@grd0: ALC_Status = ON
@grd1: targetPosition ∈ Lane
@grd2: targetSpeed ∈ SPEED
@grd3: targetSteeringAngle ∈
    STEERING_ANGLE
/* new (target) position must be within set
   of position inside the lane */
@grd4: targetPosition ∈ move(
    SDV_POSITION_env ↦ targetSpeed ↦
    targetSteeringAngle)
@SDV: targetPosition = target_SDV_lon ↦
    target_SDV_lat
@grd5: signal = TRUE
@grd6: stage = AutonomousDriving
@grd7: LV_lon ↦ LV_lat ∈ Lane
/* new position keep safe distance */
@safe: SAFE_DISTANCE > LV_lon -
    target_SDV_lon
then
@act1: SDV_POSITION_env :=
    targetPosition
@act2: SDV_SPEED_env := targetSpeed
@act3: SDV_STEERING_ANGLE_env :=
    targetSteeringAngle
@act4: stage := Perception
@act5: IMAGE_env := camera(
    targetPosition)
@act6: RADAR_reading_env := radar(
    targetPosition)
@act7: signal := FALSE
/* remove the old detecting of leading
   vehicle */
@act8: leadingVehicleSet := {LV_lon ↦
    LV_lat}
@reset-distance: distance := 0
with
@new_position: new_position =
    targetPosition
@sp: sp = targetSpeed
@steer: steer = targetSteeringAngle
end

```

In scenarios where the SDV operates in proximity to a leading vehicle, the guard `@grd7` within the `auto_move_with_LV` event is designed to ensure that the SDV's navigation does not compromise the prescribed safe distance to the leading vehicle. In addition, to cover potential changes in the position of the leading vehicle, the action `@act8` assigns a new position for the leading vehicle within the target lane. This step is based on the assumption that the leading vehicle stays within the boundaries of its current lane. Thus, these conditions allow for dynamic adjustment to safety guidelines during SDV movement.

The second category refers to manual movement, encapsulated by the `manual_move` event. This event represents the circumstances when the SDV moves manually due to the corrective action provided by the human driver. It operates under the assumption that the human driver maintains the SDV within the target lane.

```

event manual_move extends
  manual_move
where
  @grd0: ALC_Status = ON
  @grd3: signal = TRUE
  @grd4: stage = Intervention
then
  @act2: stage := Perception
  @act3: IMAGE_env := camera(
    new_position)
  @act4: RADAR_reading_env := radar(
    new_position)
  @act5: signal := FALSE
  /* remove the old detecting of leading
    vehicle */
  @act6: leadingVehicleSet := ∅
  @reset-distance: distance := 0
end

```

Table 8.5 shows how the safety requirement (SR1 to SR6) has been satisfied in a formal model. The assumptions related to autonomous speed identification include:

- A1: The radar sensor consistently provides reading points for any potential leading vehicle within the target lane.
- A2: The autonomous controller can access radar reading points from the sensor radar attached to the SDV.

TABLE 8.5: Safety requirements (SR1 to SR7) in formal model for ACC functionality

Safety requirements	Event-B elements
SR1	Environmental invariant, <i>Environment-consistency</i> , $stage = AutonomousDriving \Rightarrow move(SDV_POSITION_env$ $\mapsto targetSpeed \mapsto targetSteeringAngle) \subseteq Lane$
SR2	Action, @act2, in the <i>ALC_ON</i> event @act2: $ACC_target_speed := tsp$ When a driver may specify the ACC target speed.
SR3	Actions, @act6 and @act7, in the <i>perception</i> event @act6: $leadingVehicleSet := Seen_radar_reading[RADAR_reading_env]$ @act7: $desirePath := OEDR_task(leftLanePoints \mapsto rightLanePoints \mapsto$ $confidenceScore \mapsto leadingVehicleSet)$ When the ALC leverages radar reading points to identify the desired path.
SR4	Action, @act3, in the <i>decision</i> event @act3: $targetSpeed := target_speed(targetPosition \mapsto ACC_target_speed \mapsto$ $leadingVehicleSet)$
SR5 & SR6	Guard, @grd1, in the <i>safe_distance_violation</i> event. @grd1: $distance < SAFE_DISTANCE$, where actions are @act1: $stage := Intervention$ @act2: $targetSpeed := speed_reduced(SDV_SPEED_env)$

8.1.6 Proof Statistics

Table 8.6 shows the proof statistics of the ALC model that includes the LKA, DMS and ACC functions. A step-by-step approach was used to model the automation aspects

across four layers, with each layer corresponding to a single machine in Event-B. The results emphasise that most of the proof obligations were discharged automatically. This was achieved by using modelling patterns. These patterns were developed in Section 6.2 of Chapter 6 and were specifically designed to encapsulate the automation aspects within Event-B models.

TABLE 8.6: Proof statistics of ALC case study

Machine	Generated PO	Automatically Proved	%
M0	3	3	100
M1	17	17	100
M2	220	208	94
M3	61	61	100

8.2 Discussion

Within chapters 7 and 8, the functionalities of the ALC system are seamlessly integrated into RATP’s modelling patterns. This structured approach facilitates a thorough evaluation of system behaviours and interactions.

The ALC system encompasses three main functionalities: LKA, ACC, and DMS. Each of these components adds complexity, necessitating careful analysis to ensure safety.

RATP’s systematic approach enables a thorough examination of potential hazards within the ALC system. This involves evaluating the system’s capability to autonomously adjust steering and speed, while also considering the possibility of human interventions. By analysing these factors, RATP helps to uncover inherent safety challenges in the ALC system. For instance, these challenges include:

- Progressing from abstract behavioural models of the ALC system to concrete scenarios where human driver intervention may be necessary.
- Gradually identifying safety requirements and refining safety properties through a chain of refinements.
- Investigating both the capabilities of autonomous features and the decision-making process for human driver interventions.

Through RATP, specific safety requirements and assumptions related to the ALC system are identified and analysed. This process involves modelling critical properties that emerge during iterative analysis, leading to refined safety measures. By systematically addressing these requirements, RATP ensures effective management of safety concerns throughout ALC system development and deployment.

In summary, the RATP methodology provides a structured and systematic approach to addressing safety complexity in the ALC system. By integrating ALC functionalities, investigating hazardous events, and identifying safety requirements, RATP offers valuable insights for enhancing the safety and reliability of autonomous driving systems.

8.3 Evaluation of RATP Method in the ALC Case Study

Employing STPA for Analysis of the ALC System: Systems Theoretic Process Analysis (STPA), serving as a method for hazard identification, provides a systematic approach to examining the behaviours of SDVs [130]. Abdulkhaleq et al. [3] propose a dependable architecture for SDVs based on STPA. In addition, the main idea is that the functional and architectural design of SDVs can decompose into three levels: 1) *vehicle level*, which is a high-level view of the SDV; 2) *system level*, which demonstrates the multiple interdependent software components; and 3) *component level*, which is a low-level view of the SDV.

Abdulkhaleq and Wagner [1] also applied STPA to the ACC in order to show that the result of STPA is applicable to identification of potential accident scenarios, for instance human decision-making errors and component interaction accidents. Hanneet et al. [89] applied STPA to a LKA system to derive safety constraints and requirements. However, they considered only the LKA system, and the driven requirements did not cover the interactions of LKA with the DMS or other autonomous systems such as ACC. In comparison to these studies, the RATP approach takes a more comprehensive view of automation. It includes the role of the human driver as a fallback option, while the ALC system identifies autonomously both speed and steering.

Specifying Automation Aspects of the ALC System: In Ref [2], the ACC function was analysed using the STPA approach. The corresponding Linear Temporal Logic (LTL) formula was developed, based on the requirements identified through STPA. This LTL formula specifies essential conditions for an SDV, including the requirement for an acceleration signal when the lane is clear (i.e., the distance is greater than the safe distance). Furthermore, the Signal Temporal Logic (STL) was employed to encapsulate the requirements of the perception module [127]. For example, requirement R2 is stated as “Sensor S should detect its visible/target obstacle within T1 time unit”. Subsequently, this requirement was translated into an STL formula to aid the control design and testing phase of the system under investigation. Based on the four-variable model by Parnas and Madey [107], the Event-B models were constructed to verify and validate the critical automation aspects of the ALC system. For instance, studies such as the Cruise Control System (CCS) [137], Lane Departure Warning (LDW) [136] and Speed Control System (SCS) [90] have identified numerous properties related to the ACC and LKA.

However, these studies consider only certain automation aspects of speed or steering, without considering both of them together. Comparing these studies to the application of RATP to the ALC case study, the driven STPA requirements were gradually identified to specify system behaviours at various abstraction levels in Event-B. New automation aspects were examined, such as the intervention request and the driver's awareness level. However, the temporal properties of a system remain a significant concern.

8.4 Conclusion

This chapter expanded the application of the RATP approach to include the ACC function. We detailed how modifications in the ALC layers enabled the system to autonomously determine the speed of the SDV. The following chapter explores the temporal properties of the SDV system, with particular focus on investigating a driver's response when the ALC system issues an intervention request.

Chapter 9

Modelling the Driver Reactions

This chapter describes our third contribution, entitled '*Intervention Timing Pattern*'.

In the preceding two chapters the Automated Lane Centring (ALC) system was examined by using the Rigorous Analysis Template Process (RATP), which was applied across four layers to include the Lane Keeping Assist (LKA), Adaptive Cruise Control (ACC) and Driver Monitoring System (DMS) functions. However, the examination of the temporal properties is also required, particularly when the ALC system issues a request for the human driver to intervene.

In this chapter, a deeper exploration into a specific aspect linked to the main Research Question (RQ3.2) of this thesis.

RQ3.2: *How can a methodology be developed to formally analyse human responses when the SDV may issue a request to intervene?*

RQ3.2 seeks to develop a method for examining the complexity of human responses during the operation of Self-Driving Vehicle (SDV) systems. Building upon the modelling patterns proposed earlier in Section 6.2 of Chapter 6, we introduce the intervention timing pattern. This pattern specifically examines driver reactions to potential intervention requests from SDVs. Moreover, the ALC layers are extended to validate the intervention timing pattern within the context of the ALC system.

Section 9.1 introduces the intervention timing pattern for modelling driver responses. Section 9.2 applies this pattern to the ALC case study. Section 9.3 elaborates on the advantages and outcomes derived from employing the intervention timing pattern in addressing RQ3.2. Section 9.4 discusses related work. Section 9.5 concludes this chapter.

9.1 Intervention Timing Pattern

The intervention timing pattern investigates how a human driver might respond when the SDV system asks for intervention. Our pattern not only models the reaction of a human driver but introduces new requirements and assumptions that need to be considered to make the SDV system safe.

The intervention timing pattern explains our modelling choice and offers a broad context for understanding key properties such as *time progression*, *the clock (timer)*, *human reaction time* and *alert time*. The primary concept involves using guarded events with time constraints; thus these guarded events can be triggered only when the system reaches a specific time.

The time progression is also designed as *an event*; therefore, there is no need to modify the underlying language of Event-B. The variable *time* is defined as a natural number, which allows time constraints, such as *alert time*, to be expressed as constants or as relationships between different times. Moreover, time observations can be represented by other events determining future states (events) of a system.

To enhance clarity and consistency in the modelling patterns of SDV systems, we introduce an additional layer called *Layer 4*. This layer builds upon the modelling patterns discussed in Chapter 6, Section 6.2, following the same steps in the RATP approach. Layer 4 of the intervention timing pattern is explained in the next subsection.

9.1.1 Layer 4: Intervention Timing Pattern

Step 1. Instantiation: This step builds upon the system boundary diagram developed earlier in *Layer 3* as part of the modelling pattern discussed in Subsection 6.2.4 of Chapter 6. We incorporated timing considerations related to certain automation aspects as follows:

- The time when a semi-automated system issues an intervention request.
- The time when a semi-automated system might trigger an auditory notification.
- The time when a driver might respond to either the intervention request or auditory notification.

Based on these timing considerations, Figure 9.1 adds the timestamps associated with the specific situations under which the semi-automated system is required to issue an auditory notification if an intervention request is ignored. Such a notification could be activated, for example, when the driver does not take the necessary corrective action. These timing aspects are considered only when the intervention request is sent. The

auditory notification is mostly designed to ensure the driver's responsiveness. In general, automotive companies such as OpenPilot [10] and General Motors [66] use alert systems for two primary tasks: 1) *ensuring that the driver remains alert and responsive*; and 2) *warning the driver about potentially hazardous driving conditions*. The alert function is designed to work in conjunction with these tasks and to interact with them over a period of time. For instance, OpenPilot triggers an auditory notification if the driver fails to respond to an intervention request within 6 seconds.

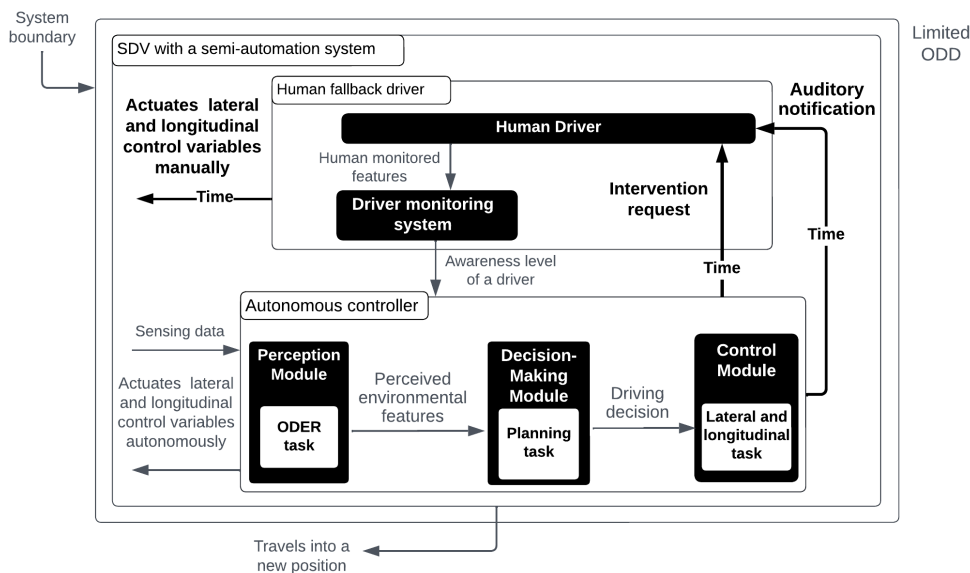


FIGURE 9.1: System boundary diagram in RATP Layer 4

Step 2. The purpose of analysis: The purpose of analysis is to study how a driver may react to a request to intervene. The time constraints associated with alerting a driver are investigated when the intervention request might be ignored. Therefore, a System Loss (SL) is identified as **SL1**:

- **SL1:** The SDV collides with an object outside of its Operational Design Domain (ODD) because the driver does not take the necessary corrective action.

As a driver might not react when the semi-automated system issues a request to intervene, a System Hazard (SH) associated with the system loss (**SL1**) is identified as **SH1**:

- **SH1:** The driver does not react to a request to intervene.

A high-level Safety Constraint (SC) that satisfies the system conditions to prevent **SH1** is **SC1**:

- **SC1:** The semi-automated system must alert a driver to take control of an SDV if the driver does not react to a request to intervene.

Step 3. Control structure: The construction of a control structure also builds upon the control structure established in *Layer 3*, detailed in Subsection 6.2.4 of Chapter 6. Figure 9.2 presents a more refined control structure, organised around a system that alerts a driver if intervention requests are ignored (**SC1**). A feedback loop (**F7**) represents that the semi-automated system delivers an auditory notification if the driver fails to respond to an intervention request (**F6**). In addition, the semi-automated system allocates a restricted timeframe in which the driver should react, as represented by Control Action (**CA3**). If a driver ignores these requests, the semi-automated system is responsible for raising the alarm and releasing control of the SDV immediately.

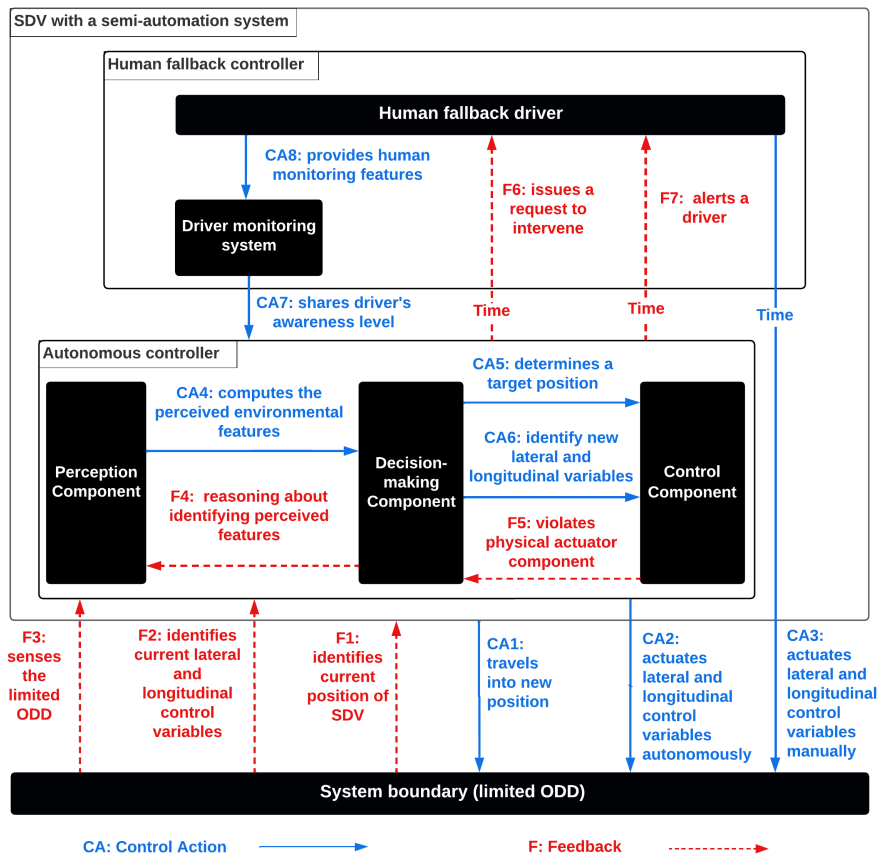


FIGURE 9.2: Control structure in RATP Layer 4

Step 4. Identifying unsafe control Table 9.1 presents the Unsafe Control Actions (UCAs) are presented, which correspond to system hazard (**SH1**). These UCAs denote unsafe system behaviours that may participate in **SH1**. The behaviours are described as follows:

- A driver is aware of the intervention request but fails to respond.

- A driver acknowledges the intervention request but responds late.
- When the driver is aware but does not respond, the semi-automated system may stop working and initiate the switch to manual driving.

TABLE 9.1: Unsafe control actions in RATP Layer 4

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA7 and CA3	UCA1.1: Semi-automated system does not alert a driver if a driver's correction action is not provided.	UCA1.2: A driver is aware (CA7) of a request to intervene (F6), but does not respond (CA3).	UCA1.3: A driver is aware (CA7) but does respond too late (CA3).	UCA1.4: Semi-automated system may release a control of an SDV when a driver is aware (CA7) but does not react (CA3).
Causal Factor				
A poor reaction of a driver when the semi-automated system issued a request to intervene.				

In view of these system behaviours, the potential Safety Requirements (SRs) pertinent to these behaviours can be outlined as follows:

- **SR1:** When an intervention request is issued, the semi-automated system should give the driver a limited time to react (driver correction action) [SH1]
- **SR2:** If the driver fails to react within the limited time, the semi-automated system should immediately trigger an alarm (auditory notification) [SH1].
- **SR3:** Once the system triggers an alarm, the driver is responsible for performing the entire driving task (manual driving) [SH1].

Step 5. Event-B model: During the Event-B modelling step, our goal is to enhance a previously presented formal model from *Layer 3* as detailed in Subsection 6.2.4 of Chapter 6. The enhancement involves incorporating timing aspects into the formal model. The intervention pattern is explained through an example Event-B model. This model can be reused in order to add different time considerations. As shown in the below, the intervention pattern has six variables:

```

machine m0
variables
redFlag //denotes a system enters a hazardous event
time //indicates any time of a system
requestTime //time when automated system issues a request to intervene
alarmFlag //sounding an alarm
alarmTime //time waiting for a response before the alarm is sounding
reactionTime //time when a human operator may react

```

invariants

@inv1: `time` $\in \mathbb{N}_1$ @inv2: `requestTime` $\in \mathbb{N}$ @inv3: `alarmTime` $\in \mathbb{N}$
 @inv4: `redFlag` $\in \text{BOOL}$ @inv5: `alarm` $\in \text{BOOL}$ @inv6: `reactionTime` $\in \mathbb{N}$

- `time`: This represents the current time of a system. The incrementation of this value implies the time progression.
- `requestTime`: This indicates any time in the future when a system may issue a request to intervene.
- `reactionTime`: This indicates any time in the future when a driver may respond to a request to intervene.
- `alarmTime`: This denotes a future time when a driver does not react to a request to intervene, and the auditory notification is immediately sounded.
- `redFlag`: This is a boolean flag that indicates a system issuing a request to intervene.
- `alarmFlag`: This is also a boolean flag that explains the status of the sound alert.

The three categories focus on various aspects of timing. The first category involves establishing an intervention timer within hazardous events that require intervention. An example can be found in the `request` event, which indicates the entrance of a hazardous event when a system waits for a response before an alarm is raised. This event is triggered when the machine prompts a request for intervention. Consequently, the intervention timer is configured within this event as follows:

```
event request
any
/* Maximum time of a system waiting for a response before raises an alert*/
duration
when
/*Any time is given for waiting for a human's response*/
@grd1: duration  $\in \mathbb{N}_1$ 
/*No intervention request and alarm is OFF*/
@grd2: redFlag = FALSE  $\wedge$  alarmFlag = FALSE
then
/*Specify a time of waiting for a driver before the alarm sounds*/
@act1: alarmTime := time + duration
/*Update the time of issuing a request to intervene*/
@act2: requestTime := time
/*Update a flag of issuing a request to intervene*/
@act3: redFlag := TRUE
/*No reaction from human yet*/
```

```

@act4: reactionTime := 0
end

```

After the creation of the intervention timer in the `request` event, the timeline for the intervention pattern is as outlined in Figure 9.3. It includes the first Requirement (SR1), which must be considered to allow the human driver to respond when the automation issues a request to intervene. The initialisation of time for a request to intervene (`requestTime`) is defined according to the current time of a system (`time`). In order to give a driver a chance to respond, the parameter `duration` indicates the waiting time of a system before the alarm is sounded. Therefore, the alert time (`alarmTime`) can be defined as the end of the waiting time.

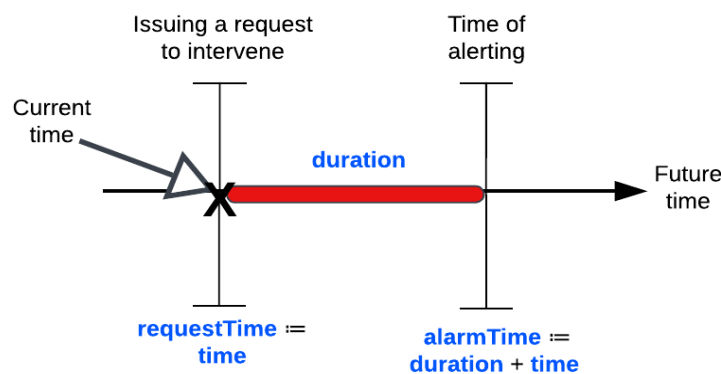


FIGURE 9.3: Creation of intervention timer

The second category of the intervention timer is time progression, as shown in Figure 9.4. In this modelling technique, the current time is frozen and can change with an observation of the `tick` event as follows:

```

event tick
where
/*Work only if a system issued a request to intervene*/
@flag_intervene: redFlag = TRUE
/*System time doesn't reach an alert time yet*/
@no_alarm: redFlag = TRUE ∧ alarmFlag = FALSE ⇒ time ≤ alarmTime
/*System time arrives on alert time, so the alarm must be operating*/
@alarmOn: (time = alarmTime ∧ redFlag = TRUE) ⇒ alarmFlag = TRUE
then
/* Increment timer */
@act1: time := time + 1
end

```

The `tick` schedules the time progression associated with the alarm property. For instance, the guard `alarmOn` captures a critical specification when the current time is already at the alert time; therefore, the auditory notification must be sounded before

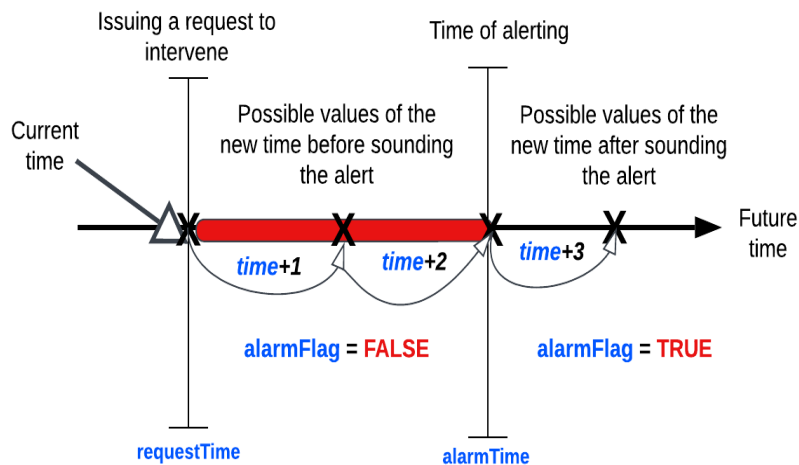


FIGURE 9.4: Time progression in intervention timer

computing the new value of time. To model the alarm property, the second Requirement (SR2) is introduced, signifying that the autonomous system will send an auditory notification if the human response is still pending. This aspect is modelled in the `notify` event as follows:

```

event notify
where
  /*System issues a request to intervene, while an alarm is not sounding */
  @grd1 : alarmFlag = FALSE  $\wedge$  redFlag = TRUE
  /*System time equal to or has moved beyond alert time*/
  @timeAlarm : time  $\geq$  alarmTime
then
  /*Update value of alarm*/
  @act1 : alarmFlag := TRUE
end

```

The third category of the intervention timer models human interventions that occur either before or after the auditory notification sounds. To capture these two potential forms of human reaction, the `intervene` event is outlined with various time intervals. The first variant of this event, addressing human reactions prior to the auditory notification, is modelled as follows:

```

event intervene
when
  @grd1 : redFlag = TRUE
  /*Possible values of system time when a human may react*/
  @grd2 : time < alarmTime
then
  /*Update a driver reaction time*/
  @receivedReaction : reactionTime := time
  @updateflag : redFlag := FALSE
end

```

The guards within the first variant of the `intervene` event play a crucial role in encapsulating the third Requirement (SR3.1) derived from SR3, which highlights the potential for a human response prior to the activation of the auditory alarm. The `grd2` guard in this initial form of the `intervene` event sets a confined timeframe, allowing for human reaction before the system's current time aligns with the alert time. Under these specified conditions, Figure 9.5 demonstrates the narrow window of opportunity available for a human to respond before triggering the auditory notification.

SR3.1: A human driver might respond before sounding an auditory notification.

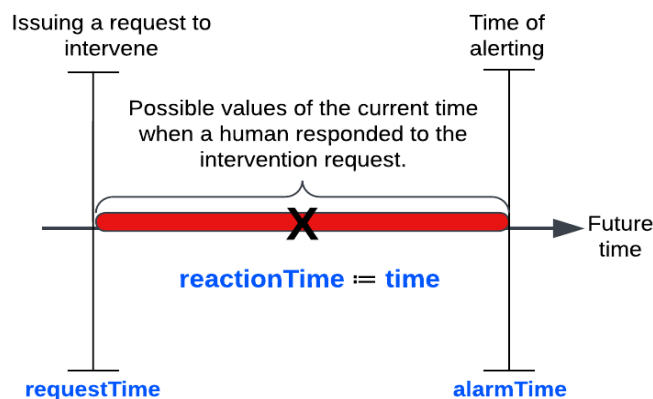


FIGURE 9.5: Human's response window time before alert notification

Similarly, the second form of the `intervene` event outlines several conditions that allow a human driver to react after the auditory notification has sounded, which is modelled as follows:

```

event intervene
when
  @grd1: redFlag = TRUE
  /*System has already raised an alarm*/
  @grd2: time ≥ alarmTime
then
  /*Update a driver reaction time*/
  @receivedReaction: reactionTime := time
  /*Update values of flag*/
  @updateflag: redFlag := FALSE
end

```

The adjustment in the `grd2` guard contributes to capturing the fourth Requirement (SR3.2) derived from SR3 that indicates the possibility of receiving a human response after the alarm sounds. Specifically, it implies that the system's current time has already surpassed the alert time. Given these conditions, Figure 9.6 illustrates the possible window of time in which a human may react after the auditory notification is triggered.

SR3.2: A human driver might respond after sounding an auditory notification.

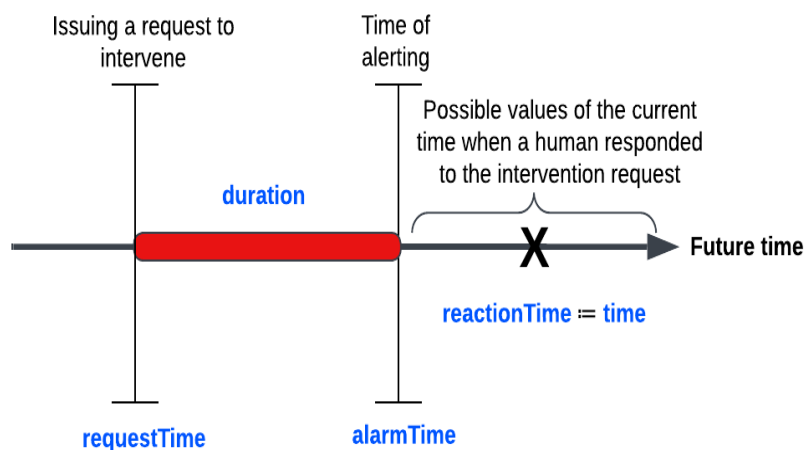


FIGURE 9.6: Human's response window time after alert notification

Since the intervention timer is executed only when a request to intervene is issued, the invariants between an SDV and a human driver are simple, and we have only to satisfy the following three invariants.

@alarm_state: $\text{alarmFlag} = \text{TRUE} \Rightarrow \text{redFlag} = \text{TRUE}$

@waiting_response: $\text{redFlag} = \text{TRUE} \wedge \text{alarmFlag} = \text{FALSE} \Rightarrow \text{requestTime} \leq \text{time} \wedge \text{time} \leq \text{alarmTime}$

@alerting: $\text{alarmFlag} = \text{TRUE} \wedge \text{redFlag} = \text{TRUE} \Rightarrow \text{time} \geq \text{alarmTime}$

The invariant **alarm_state** indicates that the alarm signal would be sent if there is still a need for human intervention. Additionally, the invariant **waiting_response** underscores that a system allocates a specific duration for the human to respond if an intervention request is dispatched (i.e., $\text{redFlag} = \text{TRUE}$). Specifically, the current time of the system (**time**) can go beyond the moment of issuing the intervention request (**requestTime**) up to the alert time (**alarmTime**). This duration is thus represented as $\text{requestTime} \leq \text{duration} \leq \text{alarmTime}$, where the system time equals the time of issuing the intervention request. On the other hand, the invariant **alerting** denotes that an auditory notification is only activated (i.e., $\text{alarmFlag} = \text{TRUE}$) if the system's current time exceeds this defined duration.

In this modelling strategy, three Assumptions (As) are incorporated into the formal model:

- **A1:** A human driver might respond immediately or after the auditory notification is activated.

- **A2:** The waiting time, or duration, is not strictly defined, for example 3 seconds. Instead, it is treated as a parameter representing any positive number.
- **A3:** The SDV is assumed to be in a safe state during the entire process of alerting and receiving responses from the human driver.

9.2 Modelling Driver Reactions in the ALC System

This section presents the application of an intervention timer in the ALC case study. It mainly models the driver's reactions when the ALC system issues a request to intervene, a process initially summarised in Section 4.2.4 of Chapter 4.

The various scenarios in which an intervention request might be issued by the ALC system have been addressed in the ALC layers discussed in Chapters 7 and 8. For a more precise representation of intervention timings, these layers are refined by introducing a new layer, '*ALC-Layer 4*', to the intervention timing pattern.

The following subsection presents the intervention timing pattern of the ALC case study according to the steps of the RATP approach.

9.2.1 ALC-Layer 4: Modelling the Intervention Timing Pattern

Step 1. Instantiation: Based on the system boundary diagram detailed in Subsection 8.1.1 of Chapter 8, this step introduces temporal properties to simulate human reactions upon receiving an intervention request from the ALC system. Some automotive companies, such as Volvo [131] and Comma.ai [10], already implement notification systems to ensure driver alertness during autonomous vehicle operation. For example, OpenPilot, the autopilot software developed by Comma.ai [10], provides a 4-second window for the driver to respond to an intervention request. If this request is ignored, the system issues an auditory notification after 6 seconds and progressively reduces the SDV's speed until it is completely stopped. These procedures are illustrated in Figure 9.7, where temporal properties associated with the driver's potential responses are incorporated, such as the auditory notification to be sent if the intervention request is ignored or not addressed by the driver.

Step 2. The purpose of analysis: The purpose of analysis is to study how a driver may react to a request to intervene (warning messages). The time constraints associated with the driver reactions are investigated, especially when the ALC issues a request to intervene. Consequently, a SL is identified as **SL2**:

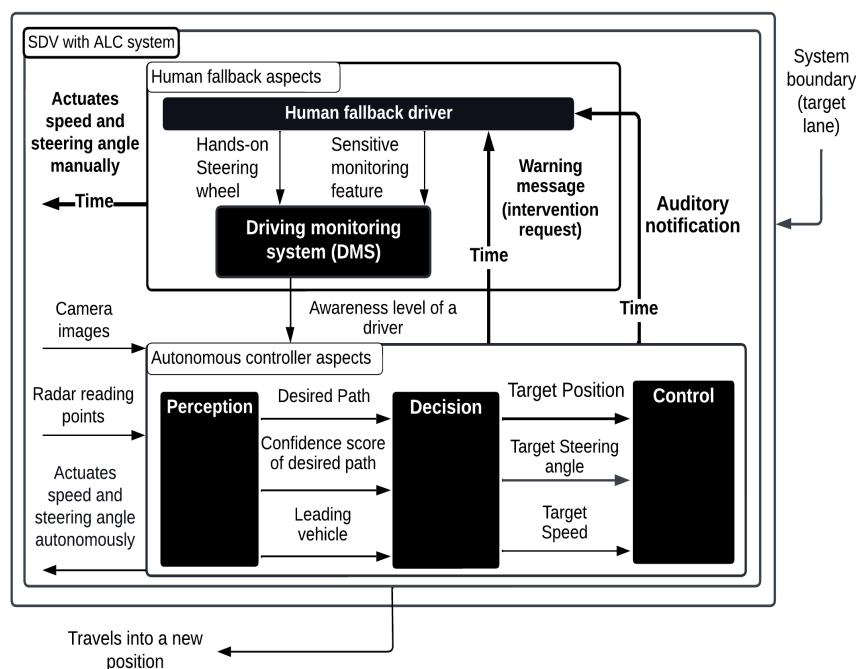


FIGURE 9.7: System boundary diagram incorporating intervention timing pattern

- **SL2:** The SDV collides with an object outside of its target lane because the driver's corrective action is missing.

As a driver might not react when the ALC issues a request to intervene, a SH of **SL2** is **SH2**:

- **SH2:** The driver does not react when the ALC system issues a request to intervene

A high-level SC that satisfies the system conditions to prevent **SH2** is **SC2**:

- **SC2:** The ALC must alert a driver to take the control of an SDV if the driver does not react to a request to intervene.

Step 3. Refining control structure: This step refines the control structure based on the structure discussed in Subsection 8.1.3 of Chapter 8. It considers a scenario where the ALC system issues a request for intervention. Figure 9.8 illustrates a more detailed control structure designed around the ALC system, which alerts the driver if intervention requests are ignored (**SC2**). This structure includes the Control Actions (CAs) and Feedback Loops (FLs) necessary for the ALC system to send an auditory notification (**F10**) if the driver fails to respond (**CA3**) to the intervention requests (**F6**, **F7**, **F9**). The ALC system allows a certain amount of time for the driver to react. Nonetheless, in instances where the intervention requests are ignored by the human driver, the ALC system is

responsible for initiating an auditory notification, subsequently releasing control of the SDV.

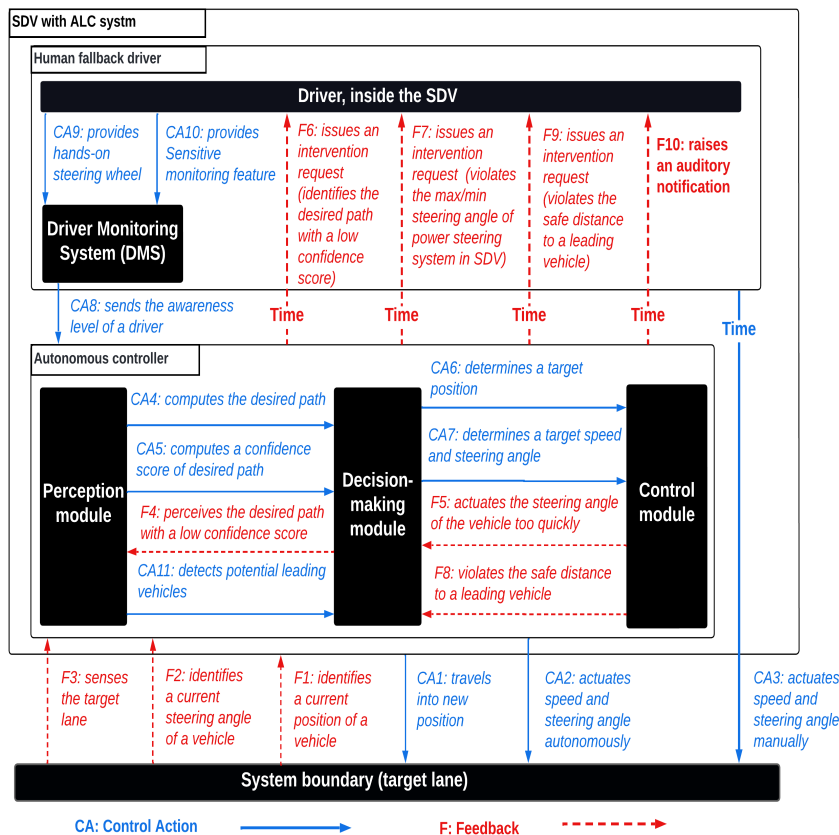


FIGURE 9.8: Control structure in ALC Layer 4

Step 4. Identifying unsafe control actions: Table 9.2 presents the UCAs related to the system hazard (**SH2**). These UCAs indicate potentially dangerous system operation that may contribute to **SH2**, including:

- Driving scenarios where the human driver recognises the request for intervention but takes no action.
- Driving scenarios where the human driver acknowledges the intervention request but delays taking action.
- Driving scenarios in which the ALC system is continuously operated despite non-response from the human driver, leading to unknown safety risks.

Therefore, the SRs of **SH2** can be identified as follows:

- **SR4:** If the ALC issues a request to intervene, the ALC system must allow the driver a specified time in which to react (driver correction action) [**SH2**].

- **SR5:** If a driver does not react within the specified time, the ALC immediately raises the alarm (auditory notification) [SH2].
- **SR6:** If a system raises the alarm, the driver is responsible for performing the entire driving task (manual driving) [SH2].

TABLE 9.2: Unsafe control actions in ALC Layer 4

Control Action (CA)	Not providing CA leads to hazards	Providing CA leads to hazards	CA applied early, late, or out of order	CA stopped too soon or applied too long
CA8	<i>UCA1.5:</i> ALC system does not alert a driver if a driver's corrective action is not provided.	<i>UCA1.6:</i> A driver is aware (CA8) of a request to intervene (F6,F7,F9), but does not respond to a request to intervene (CA3).	<i>UCA1.7:</i> A driver is aware (CA8) but does respond too late (CA3).	<i>UCA1.8:</i> ALC system may release a control of an SDV when a driver is aware but does not react (CA3).
Causal Factor				
A poor reaction of a driver when the ALC issued a request to intervene.				

Step 5. Refining Event-B model: Mapping is provided in Table 9.3 to demonstrate how the new feedback loop (**F10**) links to the representation of a formal model. Full details of the refined machine (machine **m4**) are in Appendix G.2. The auditory notification is modelled using a Boolean, as follows:

```
/* The alert sound might be (TRUE) or (FALSE) */
@typeof-audiotoryNotification : auditoryNotification ∈ BOOL
```

TABLE 9.3: Feedback loop linked to ALC Layer 4

Feedback loop	Event-B elements
F10	Variable, auditoryNotification

In addition to the auditory notification, four temporal variables are introduced to model the reactions of the human driver within the ALC systems.

```
time //current time of a ALC system           @typeof-time : time ∈ ℕ
sentRequest //ALC time of issuing a request    @typeof-sentTime : sentRequest ∈ ℕ
driverReact //ALC time when a driver may react @typeof-actTime : driverReact ∈ ℕ
alertTime //ALC time of alerting a driver      @typeof-alarm : alertTime ∈ ℕ
```

The three categories represent different timing aspects. First, the creation of the intervention timer is included in the intervention driving scenarios. These scenarios were modelled in several Event-B events as follows:

1. `lowConfidenceScore_interven`: The intervention timer starts if the desired path is identified with low confidence.
2. `correct_exceeding`: The timer also starts when the steering attempts to go beyond the SDV's range.
3. `correct_out_of_lane`: The timer is activated if the ALC system suggests changes in speed and steering angle that would move the SDV out of the target lane.
4. `safe_distance_violation`: The timer starts if the ALC system speeds up the SDV so that it is too close to the vehicle ahead.

The driving circumstances leading to the above events were analysed in the previous ALC layers. This was done when applying the RATP approach to the LKA and ACC functions, as discussed in Chapters 7 and 8. However, we extended these events to include the creation of an intervention timer. For instance, the `lowConfidenceScore_interven` event is extended based on the definition of the `request` event in the intervention timing pattern as follows:

```

event lowConfidenceScore_interven extends lowConfidenceScore_interven
any
  /*maximum time of a system waiting for a response before raising an alert*/
  duration
when
  /*time of waiting for a reaction from a driver*/
  @grd6: duration ∈ ℕ1
  @grd7: warningMessage = FALSE ∧ auditoryNotification = FALSE
then
  /*specify a time of waiting for a driver before the alarm sounds */
  @act3: alertTime := duration + time
  /*update a time of issuing a request to intervene*/
  @act4: sentRequest := time
end

```

Initiation of the timer within the intervention events results in the definition of action `act4` as a variable `sentRequest` according to the current `time` of the system. To provide the driver with an opportunity to respond, the parameter `duration` indicates the waiting time before an alarm is activated. Therefore, the action `act3` sets a variable `alertTime` as the end of the waiting interval.

The second category of the intervention timer is time progression. Based on the `tick` event in the intervention timing pattern, the current `time` is frozen and it can change with an observation as follows:

```

event tick
where
  /* works only if a system issued a request to intervene */

```

```

@grd1: warningMessage = TRUE
/* time less than a time of alert, so there is no need for alert*/
@grd2: warningMessage = TRUE ∧ auditoryNotification = FALSE ⇒ time ≤ alertTime
/*time is reach alert time, auditory notification must be operating*/
@grd3: (time = alertTime ∧ warningMessage = TRUE) ⇒ auditoryNotification = TRUE
then
/* accept increment time */
@act1: time := time + 1
end

```

The `tick` event schedules the time progression associated with the alarm property. For instance, the guard `grd3` captures a critical specification when the current time is already at the alert time; therefore, the auditory notification must be sounded before computing the new value of time. To represent the alarm property, we model the `alert` event by leveraging the `notify` event in the intervention timing pattern. This enables the ALC system to send an auditory notification as follows:

```

event alert
when
/* ALC issues a request to intervene, while an alarm is OFF */
@grd1: auditoryNotification = FALSE ∧ warningMessage = TRUE
/*current time equal to or has moved beyond alert time */
@grd2: time ≥ alertTime
then
/*update value of auditory notification*/
@act1: auditoryNotification := TRUE
end

```

The third category of the intervention timer is to model the driver intervention before/after sounding the auditory notification. Based on the `intervene` event in the intervention timing pattern, two events represent the possible reactions from a driver: `response_before_alarm` and `response_after_alarm`.

```

event response_before_alarm extends
Manual_actuating when
//ALC issues a request, while alarm 'OFF'
@grd8: warningMessage = TRUE ∧
    auditoryNotification = FALSE
//(SR6): possible values of current time
    when a driver may react
@grd9: time ≥ sentRequest ∧ time <
    alertTime
then
//update a driver reaction time
@upd-receivedTime: driverReact := time
end

```

```

event response_after_alarm extends
Manual_actuating when
//ALC issues a request, while alarm 'On'
@grd8: warningMessage = TRUE ∧
    auditoryNotification = TRUE
//(SR6):ALC raised an alarm
@grd9: time ≥ alertTime
then
@upd-receivedTime: driverReact := time
// rest an alarm
@upd-alarm: auditoryNotification :=
    FALSE
end

```

These events involve several conditions for allowing the driver to react before/after sounding the auditory notification. For instance, the guards `grd9` in both events indicate time constraints that explain the time when a driver may react (**SR5**).

Since our pattern is executed only when a request to intervene is issued, the invariants between the ALC system and a human driver are simple, and we have only to satisfy the following two invariants.

`@waiting_response: warningMessage = TRUE \wedge auditoryNotification = FALSE \Rightarrow sentRequest \leq time \wedge time \leq alertTime`

`@alerting: auditoryNotification = TRUE \wedge warningMessage = TRUE \Rightarrow time \geq alertTime`

The invariant `waiting_response` emphasises that a system gives the driver specified time in which to respond if the intervention request is sent (`warningMessage = TRUE`). Precisely, the current time of a system (`time`) can move beyond the time of issuing a request to intervene (`sentRequest`) to the alert time (`alertTime`); that is the duration (**SR4**) can be written as `sentRequest \leq duration \leq alertTime` where the current time of a system is equal to the time of issuing a request to intervene. Second, the invariant `alerting` indicates that a system sounds an auditory notification (`sentRequest = TRUE`) only if the current time of a system goes beyond that duration (**SR5**).

Table 9.4 details how the safety requirement (**SR4** to **SR6**) has been satisfied in a formal model. There are three Assumptions (As) involved in a formal model:

- **A4:** A human driver reacts immediately after the auditory notification is sounded.
- **A5:** The waiting time/duration is not strictly defined, such as 3 seconds. Instead it is treated as a parameter representing any positive number.
- **A6:** The ALC system is assumed to be in a safe state during the entire process of alerting and receiving responses from the human driver.

TABLE 9.4: Safety requirements (**SR4** to **SR6**) in formal model for ALC Layer 4

Safety requirements	Event-B elements
SR4	Invariant, <code>@waiting_response: warningMessage = TRUE \wedge auditoryNotification = FALSE \Rightarrow sentRequest \leq time \wedge time \leq alertTime</code>
SR5	Invariant, <code>@alerting: auditoryNotification = TRUE \wedge warningMessage = TRUE \Rightarrow time \geq alertTime</code>
SR6	Guards in events, <code>response_before_alarm</code> and <code>response_after_alarm</code> where <code>@grd9</code> in both events specifies a time that a driver may react <code>@grd9: time \geq sentRequest \wedge time $<$ alertTime</code> <code>@grd9: time \geq alertTime</code>

9.3 Discussion

The intervention timing pattern is a critical component in the development of SDV systems, especially when considering the human driver as a fallback option during hazardous driving events. It outlines a structured method for specifying the timing properties needed to model the windows of opportunity for a driver's intervention response. This pattern is particularly relevant for systems such as ALC, where the precise measurement of human interaction is vital for ensuring safety and reliability.

The analysis utilises an intervention timing pattern to accurately map the timing of a driver's potential responses to the ALC system's prompts. It differentiates between scenarios in which a driver might respond before the urgency of auditory notification is perceived, and situations where the driver's reaction occurs after acknowledging the notification. This approach facilitates a thorough examination of the range of a driver's possible reaction times in relation to the auditory signals from the ALC system.

The benefits of utilising the intervention timing pattern in SDV systems are multifaceted:

- **Requirement and Assumption Identification:** The intervention timing pattern serves as a structured framework that identifies specific requirements and assumptions related to the driver's engagement in the SDV's autonomous operations. By explicitly defining these parameters, it assists in shaping a comprehensive understanding of the role and expectations of human intervention, ensuring that both system developers and stakeholders have a clear blueprint to refer to.
- **Addressing Modelling Challenges:** One of the complex aspects of modelling SDV systems lies in accounting for the uncertain nature of a driver's response during autonomous operations. The intervention timing pattern plays a crucial role in overcoming this issue. It establishes a methodical framework that integrates the possibility of human responses into the system's functional procedures, ensuring that these variables are included in the system design even if they are not directly detectable. Therefore, it ensures that the model remains robust and reflective of real-world scenarios where driver reactions may not always be apparent.
- **Driver Inclusion Beyond Fallback:** Traditionally, the role of a driver in an SDV system is often relegated to that of a mere fallback option—intervening only when the system fails or is unsure of the next course of action. However, the intervention timing pattern encourages developers to transcend this limited view. It prompts the consideration of the driver as an active participant, capable of varying responses across different scenarios. This, in turn, aids in crafting a more robust and realistic model of fallback mechanisms within the SDV system.

In summary, the intervention timing pattern not only accentuates the intricacies of human intervention in automated systems but also propels a forward-thinking approach to SDV system development. This pattern is instrumental in acknowledging the range of possible human reactions, guiding the creation of systems that excel technically while also being attuned to human behaviour and needs. Such an approach promotes a seamless integration of humans and machines, aiming for a balanced and cooperative relationship.

9.4 Related Work

Formalising and verifying discrete timing properties. Cansell et al. [29] developed a pattern to model the timing and order of events within systems, using time-stamped actions and reactions to simulate real-time processes. Their model uses a clock (timer) to track current time and events set to trigger at future times. The system advances time and activates events accordingly. However, this model does not handle interruptions during event sequences.

Butler and Falampin [27] proposed a refinement strategy of timing properties that introduces a clock variable representing the current time and an operation that progresses the clock. Therefore, time constraints are added to the clock to handle interruptions during event sequences where the clock cannot move beyond the specific point at which the deadline is violated.

Based on this methodology, many studies, such as [117, 142], have been carried out to extend Event-B with timing properties. Sarshogh and Butler [117] propose a trigger response pattern to develop Event-B models with several timing properties such as deadline, delay and expiry. Their approach assigns timestamps for trigger and response events and employs a tick event to prevent the global clock from moving to a point where time constraints between the trigger and response events would be violated. Zhu et al. [142] extended the work of [27, 117] to provide formally the semantics and syntax between the trigger and response events.

In our pattern, we specify an intervention timeline based on the human reaction time and the alert time, namely the deadline can be seen as a time when a driver may react, where the alert time has combined the delay and expiry based on the received human's response. Nonetheless, the time-sensitive characteristics associated with the autonomous functions of the ALC system continue to be a significant issue. For instance, the timing aspects related to observing the driving environment and determining the desired path are not fully investigated.

9.5 Conclusion

This chapter introduced the intervention timing pattern into the modelling patterns of SDV systems. Following the systematic steps of the RATP methodology, the new layer, Layer 4, was presented to model the human driver responses when the SDV systems may issue a request to intervene. In the line for RATP Layer 4, the ALC layers were extended to cover how the driver may respond to various intervention requests of the ALC system. Future work will further refine these investigations, with a particular focus on expanding the understanding of temporal properties associated with autonomous functions of the SDV system. A second area of focus could centre around refining the instantiation of patterns. Specifically, the machine inclusion plug-in [64] within Event-B could be employed to enhance the transformation of a pattern into a concrete example, thereby simplifying the application of intervention timing patterns across various use cases when the SDV may require driver interventions.

Chapter 10

Conclusions

In this chapter we present a summary of our main contributions, outline the limitations of our approach, share the lessons learned during the development of our case study and explore avenues for further research opportunities.

10.1 Contributions

The main contributions of this thesis can be summarised as follows:

- **Defining a template for analysing automation in Self-Driving Vehicle (SDV) systems:** Introduced as the Rigorous Analysis Template (RAT) in Chapter 5, this template provides a framework for the analysis of Self-Driving Vehicle (SDV) systems. It illustrates how a generic template can be instantiated into a concrete example, such as in the Automated Lane Centring (ALC) system. Based on the instantiated template, the Systems Theoretic Process Analysis (STPA) and Event-B are adopted to identify safety requirements and develop formal models, respectively. The contributions here involve two aspects:
 1. Clarify and identify the automation aspects between human drivers and SDVs during the performance of Dynamic Driving Tasks (DDTs), especially if a system requires a human driver to play a fallback role to ensure the safety of a system.
 2. Link the automation aspects of DDTs with the internal components of SDV systems in order to specify the responsibility of either human drivers or SDV systems during the performance of DDTs.
- **Developing an iterative analysis method:** Presented as the Rigorous Analysis Template Process (RATP) in Chapter 6, this method involves five systematic steps

inspired by STPA and Event-B methodologies. The contributions here also involve two aspects:

1. Novel systematic analysis steps that integrate STPA and Event-B to formally analyse and model safety concerns in SDV systems.
 2. Modelling patterns that use these systematic analysis steps to analyse SDV system behaviours. These patterns transition from a high-level abstract layer down to a more detailed, concrete layer. A primary benefit of these patterns is their ability to facilitate the analysis of system behaviours, enabling a progressive investigation into the complexity of safety concerns involved in the SDV systems.
- **Defining the Intervention Timing Pattern:** This pattern, presented in Chapter 9, explores how human drivers might respond when an SDV system issues an intervention request. It is included as the final layer inside the modelling patterns.
 - **Validation and Evaluation:** Our methodology has been validated and evaluated through application to a complex case study involving various aspects of the ALC system. This includes:
 1. High-level interactions between human drivers and the ALC system, presented in Chapter 5.
 2. The integration of the Lane Keeping Assist (LKA) and Driver Monitoring System (DMS) functions, presented in Chapter 7.
 3. The incorporation of the Adaptive Cruise Control (ACC) function within both the LKA and DMS functions, presented in Chapter 8.
 4. A detailed examination of potential driver reactions when the ALC system may issue an intervention request, presented in Chapter 9.

Specifically, these contributions can be associated with different users:

- **For developers of Self-Driving Vehicle (SDV) systems:** Modelling patterns offer a rigorous and traceable analysis that progressively investigates safety requirements related to the Self-Driving Vehicle (SDV) at the design level. This enables a detailed and methodical exploration of safety concerns that might be considered during the development of SDV systems.
- **For safety analysts:** The systematic steps of the Rigorous Analysis Template Process (RATP) initiate with the instantiation of a template. The process begins with a gradual consideration of the abstract representation of the system. As the analysis progresses, additional system components are integrated, facilitating a methodical identification of associated hazards and requirements.

- **For users of refinement-based approach:** The RATP approach provides a methodical solution to the challenging task of finding an effective refinement strategy. This includes selecting the specific abstractions, requirements and assumptions that need to be modelled in each refinement.

10.2 Limitations

In Section 6.4.2 of Chapter 6 we explained certain limitations and constraints of the Rigorous Analysis Template Process (RATP) approach and its template prototype. Some of these constraints are essential to guarantee the appropriate behaviour of Self-Driving Vehicle (SDV) systems. For instance, the human driver must be ready to assume control if the SDV system sends an intervention request.

Other limitations are related to various assumptions identified within the modelling patterns that were generated by the RATP approach. For example, SDV systems might use sensors to detect the perceived environmental features. In fact, these features must be precisely defined in advance to fully comprehend how environmental constraints may affect the performance of Dynamic Driving Tasks (DDTs).

Furthermore, the RATP approach inherits some weaknesses from the Systems Theoretic Process Analysis (STPA) and Event-B methodologies. For instance, the systematic steps of the RATP, along with their modelling patterns, are primarily designed to focus on dynamic system control and the underlying causes of hazards in the absence of component failure. Consequently, formal verification can only validate the critical properties that emerge from the interactions between multiple system components.

10.3 RATP Methodology: Lessons from ALC Case Study

In this section, we discuss the lessons learned and refinements made to the methodology while utilising RATP in analysing the Automated Lane Centring (ALC) case study.

10.3.1 Analysing Dynamic Driving Tasks of SDV Systems

In our investigation, the template for SDV was developed on the basis of collaboration between human drivers and autopilot software in order to achieve safe DDTs. To integrate this template into the ALC case study, two instantiation strategies were presented.

The first strategy proposes handling the instantiation process independently from the informal and formal methods, as outlined in Section 5.2.3 of Chapter 5. Nonetheless,

we cannot systematically assess how the instantiation process and its associated automation aspects might influence the analysis of the safe DDTs in the ALC study. For instance, this includes how the perception component may identify the left and right lane lines on the basis of the incoming images and what assumptions might arise from this detection process.

The second strategy integrates the instantiation process within a systematic and iterative analytical approach. The ALC case study was dissected across varying layers, transitioning from a high-level abstraction layer to a more detailed, concrete layer, as deeply explained in Chapter 7. Adopting the instantiation step within both informal and formal analysis phases facilitates the step-by-step examination of a system's behaviours.

To summarise, we present two key insights:

1. **Depth and refinement:** Iterative analysis allows for consistent improvement of the template. Initially, the process tackles broader concerns or overarching features. With each subsequent iteration, more specific details are examined and refined. By the end, the template comprehensively covers both general and detailed elements of the DDTs.
2. **Feedback integration:** In the iterative approach, feedback from early analysis layers is directly integrated into subsequent analysis layers. This ensures that the template evolves to capture the automation aspects of the DDTs better, leading to a more accurate representation.

10.3.2 Bridging STPA and Event-B: A Dual Strategy Examination

The combination of STPA and Event-B focuses on how well a rigorous model can meet the driven STPA requirements. To understand this better, we also presented two strategies.

Our initial approach treated Event-B and the STPA steps as distinct entities. As detailed in Section 5.2.3 of Chapter 5, this method leaned heavily on separating the two, thereby operating without a clearly defined refinement strategy. As a result, the creation of a rigorous model was left primarily to our modelling choices and potentially left room for inconsistencies.

Realising the limitations of the first strategy, our subsequent approach aimed at a more cohesive integration. Therefore, we began introducing Event-B into the process at the conclusion of the systematic analysis steps during every iterative process, a methodology illustrated in Figure 6.1 of Chapter 6. By adopting this iterative process, we established a systematic pathway, aligning the rigorous model with the specific objectives of

analysis at each abstraction layer. This ensured that the model and analysis progressed together, capturing insights at every iteration/refinement.

From this work, we learned two main points:

1. **Comprehensive view:** Each Event-B machine is not an isolated entity. Instead, it serves a specific purpose, incrementally incorporated into the overarching behaviours of SDV systems. This idea supports the need for a comprehensive view when approaching system modelling and analysis.
2. **Purpose-driven refinement:** While the freedom of modelling choices can be strong, our modelling patterns emphasised the significance of aligning refinement strategies with the analytical objectives of each iterative layer. By doing so, we ensure that the model remains relevant, precise and dynamically adjusted to the evolving landscape of the analysis.

10.3.3 Modelling New Behaviours and Features

From the development and application of the RATP approach, several key lessons were learnt. A primary takeaway was the method adaptability in exploring various automation aspects, specifically its capacity to integrate behaviours previously challenging to model.

For instance, understanding driver reactions based on the ALC system intervention requests was complex without clarity on the internal behaviours of a system. Consequently, the methodology underwent iterative refinements. Chapter 5 investigates the high-level interactions between human drivers and the ALC system. Various driving scenarios, concerning intervention requests, were explored in Chapters 7 and 8. Based on this, Chapter 9 presents models of driver reactions, taking into account earlier examinations of how the ALC system issues a request to intervene.

In summary, we highlight two primary insights:

1. **Flexibility:** The RATP demonstrated its adaptability by accommodating behaviours that were initially difficult to model. This flexibility ensured a comprehensive view of the system as the analysis progressed.
2. **Understanding human responses** As the research progressed, the analysis revealed a clearer understanding of human responses. This was particularly apparent in observing how the ALC system initiated intervention requests. The findings underscore the value of adopting a systematic and adaptable methodology, ensuring that critical components such as human intervention are comprehensively examined.

10.4 Research Opportunities

A range of future developments could be applied for both iterative enhancements and further validation of the RATP approach.

Automation Aspects: Improvements to the RATP methodology can incorporate various automation dynamics, as follows:

1. **Temporal dynamics:** One can consider the temporal properties integral to autonomous operations in SDV systems. These include timing details of sensing data reception and environment feature prediction.
2. **Automation in absence of human responses:** Certain situations might lead to the SDV system receiving no human response upon issuing an intervention request. Therefore, more investigation on the mitigation strategies might be required. An example scenario: If no intervention occurs within a limited timeframe, the SDV could initiate an emergency stop by locating a safe position.

Domain study field: The autonomy observed in SDV systems finds parallels in other sectors. The Unmanned Aerial Systems (UASs), also known as drones, present a suitable example. Drones are categorised into levels based on human (trained pilot) supervision [112]. This similarity provides potential for expanding the RATP approach adaptability, as follows:

1. **Template development:** It would be beneficial to organise and generalise a template addressing the unique automation dynamics present in both UASs and SDV systems.
2. **Human interventions in UAS systems:** Another compelling area is exploring human interventions during UAS operations. For instance, a situation where a trained pilot remotely intervenes upon a drone identifying critical circumstances.

Validation methods: To robustly validate the RATP methodology, engaging with domain experts is pivotal. Such engagement can be channelised through:

1. **Industrial case studies:** Introducing a new case study could provide insights for modifying or extending the modelling patterns. Collaboration with local experts offers a valuable perspective. This real-world validation would emphasise the RATP approach applicability.

2. **Comparative Analysis with Industry Practices:** Different automotive companies adopt various strategies to achieve high-level automation in SDV systems. For instance, Tesla and OpenPilot (software by comma.ai) develop autopilot software with pre-recorded/learned maps [31]. In contrast, companies such as General Motors (GM) and Ford emphasise the use of pre-passive recorded maps over active driving environment analysis [59]. Exploring how the RATP approach aligns with these strategies could pave a significant new path in the field.
3. **Scenario-Based Approaches for Autonomous Vehicles:** This involves creating diverse sets of real-world situations to test and validate their performance [114]. These scenarios encompass various road conditions, traffic scenarios, and unexpected events to ensure the SDV's capability to navigate safely and efficiently. By simulating these scenarios, researchers can assess the SDV's responses, identify potential weaknesses, and refine its algorithms for improved autonomy.

Appendix A

The NHTSA Safety Protocol

In this appendix, the National Highway Traffic Safety Administration (NHTSA) safety protocol is demonstrated. In general terms, the NHTSA safety protocol comprises 12 comprehensive guidelines for developing an Self-Driving Vehicle (SDV) system. These guidelines are categorised into two sections: design and testing. Tables [A.1](#) and [A.2](#) present these guidelines; for more detailed information, readers can refer to [\[8\]](#).

TABLE A.1: Conceptual details of the NHTSA safety framework.

Element	Description	Group
An efficient safety approach	A comprehensive analysis approach contains sufficient plans and control processes for developing an autonomous vehicle. It involves some standards, such as ISO 26262 [67] and ISO 21448 [73], to be adopted in order to demonstrate the vehicle safety at the design level.	<i>General</i>
ODD	The ODD (operational design domain) must be carefully considered when designing SDVs. It involves in which conditions SDVs are intended to operate. These conditions can be categorized into five factors: 1) The design of roadway (e.g. single line, highway, etc.), 2) Geographic zone (e.g. desert, city, etc.), 3) Speed limit and range, 4) Environmental events (e.g. daytime, weather, etc.), and 5) Other domain constraints (e.g. operate only in specific road, etc.). Moreover, In the case where the automated vehicle is outside of its ODD, SDVs should establish a safe transition to the fallback human component. However, if SDVs do not receive indications that a human fallback component is receptive to fully take control of a vehicle, the autonomous controller should mitigate risks and reduce speed to ensure the safe parking of a vehicle.	<i>Design</i>
OEDR	The OEDR (Object and Event Detection and Response) indicates the detection and responsive mechanisms of SDV to deal with any driving circumstances that might happen. Therefore, SDVs require performing OEDR tasks while it satisfies and obeys its ODD. In general, the ODD specifies the target environment in which the OEDR system performs its functionalities. For instance, the OEDR would be expected to detect and respond to a variety of obstacles such as other vehicles, bicyclists, pedestrians, and any ODD conditions that might affect the safe operation of SVDs.	<i>Design</i>
A reliable fallback mechanism	The SDV should be able to detect malfunctions in its operations and also be able to notify the human fallback component of such unmanaged events in order to safely change the control of a vehicle to be managed by the human driver.	<i>Design</i>
Traffic Laws	SDVs are also supposed to obey public or government standards. These governmental standards involve the expected performance of an SDV when deployed on public roads.	<i>Design</i>

TABLE A.2: Conceptual details of the NHTSA safety framework (continued).

Element	Description	Group
HMI (Interfaces)	It demonstrates a well-designed interface for showing the driving information, such as the current motion plan, which obstacles might affect driving behaviours, and so on. In general, SDVs should ensure a safe communication channel among the operator (passengers, human driver) and the autonomous controller.	Design
Cybersecurity threats	An SDV is also responsible for managing any malicious attacks. These attacks might be an anonymous person who remotely controls the autonomous vehicle through unauthorized access permission through the internal input and output channels, such as Bluetooth, and wireless maintenance ports , and more [103].	Design
Testing	NHTSA recommends an extensive testing approach before deploying any release of SDVs.,This approach involves three types of testing: 1) simulation, 2) indoor road, and 3) public road testing. In addition, the process of testing a system might be evaluated and performed by either the development team or an independent third party.	Testing and crash mitigation
Avoidance crash method	NHTSA also encourages the design of crash avoidance methods that minimize the extent of injury during a crash event. Although crashes remain a reality of public road driving, SDVs can apply a variety of techniques, such as the ABS (anti-lock braking system), in order to minimize crash damages.	Testing and crash mitigation
A safe post-crash method	SDVs should rapidly return to a safe state after being engaged in an accident. Therefore, NHTSA encourages automotive companies to identify the strategies that would be applied when a crash happens.	Testing and crash mitigation
Data recording	NHTSA recommends recording and reviewing the behaviours of the SDV during its mission. This kind of behavioural data can help to identify potential software bugs and errors. Moreover, these failures will be considered in the next major release.	Testing and crash mitigation
Training	NHTSA recommends having a well-defined consumer training session. It could involve the ways that a human fallback driver can engage within SDVs, especially if he/she may need to take over control of the vehicle. In addition, it also contains a training session on effective communications between the vehicle's consumers and the internal interface system in order to understand the capability and limits of the SDV and its HMI.	Testing and crash mitigation

Appendix B

Automated Lane Centring in OpenPilot

This appendix explains our exploration of the Automated Lane Centring (ALC) system developed by OpenPilot, conducted within the Carla simulator. Subsection B.1 provides a Python script used to monitor and record the behaviours of the ALC system in the simulator. Subsection B.2 summarises the analysis of the recorded behaviours, aiming to encapsulate and provide a clear summary of the main goals the ALC system seeks to accomplish. The complete Python scripts are accessible for download at this provided link¹.

B.1 Recording Behaviours using Python Scripts

OpenPilot, a semi-automated driving system developed by Comma.ai [10], is used in the Carla simulator [43] to simulate self-driving and test different algorithms in a safe environment.

To connect OpenPilot with Carla, a bridge plugin [10] was created. This bridge, a Python script, acts as an interpreter, allowing OpenPilot and Carla to exchange information. This makes it possible for researchers and developers to safely test self-driving algorithms in a realistic simulation environment.

The bridge in OpenPilot gathers data from Carla's sensors, like cameras, LiDAR, GPS, and vehicle motion details, similar to what real Self-Driving Vehicle (SDV) would collect. Once the data is collected, the bridge converts it into a format that OpenPilot can use. OpenPilot then uses this data to make decisions about how the SDV should steer.

¹The Python scripts used for analysing and recording the behaviours of the ALC system in the Carla simulator are available in a GitHub repository at <https://github.com/fhdotb/openPilotCarla.git>.

Therefore, we used OpenPilot’s bridge to study how the ALC system behaves in different conditions like ‘clear’ and ‘rainy’ weather and at different speeds, like 25, 30, and 40 miles per hour. To do this, we added some Python functions to OpenPilot’s bridge, primarily as follows:

- **policing_function:** This function extracts important information from OpenPilot’s ALC software. For instance, it calculates a detection score, which gives us how likely the system is to correctly identify the desired path by analysing the lane lines in incoming images.
- **spawn_leading_vehicle:** This function places a new vehicle in front of OpenPilot in the Carla simulation. This allows us to study how OpenPilot responds when there is a vehicle ahead.
- **humanDriverStateOutputs:** This function monitors a human driver’s facial expressions and eye movements to assess their level of alertness when OpenPilot is in operation. If the driver’s behaviour indicates they may not be paying adequate attention, OpenPilot’s software will alert a driver and then automatically deactivate.

The corresponding code for these functions is provided below.

```
def policing_function(currentspeed,currentVelocity,current_vehicle_angle,
old_vehicle_angle,steer_now,steer_old,is_openpilot_engaged,current_location,
max_steer_angle,rk,count_f,risk_priority):

leftProb, rightProb , desiredPathPro , desire = lateralPlanOutputs()
cruise_control_based, hasLead = longitudinalPlan()
valid, yawRate, steerRatio, sensorValid, angleOffsetAverageDeg, angleOffsetDeg =
    liveParametersOutputs()
awarenessStatus , awarenessActive , awarenessPassive = humanDriverStateOutputs (0)
alertStatus,alertText1,alertText2,alertSound = carStateParameters ()
if desiredPathPro < 0.6 and desiredPathPro > 0.3:
    risk_priority = risk_priority +1
    text_error_message = 'detection probability less than 60'
    safety_procedure = 'mitigation strategy'
elif desiredPathPro <= 0.3:
    risk_priority = risk_priority +2
    text_error_message = 'detection probability less than 30'
    safety_procedure = 'mitigation strategy'
else:
    risk_priority =0
    text_error_message = 'detection probability more than 60'
    safety_procedure = 'Pass'
case = {'time': getTime(),
'is_openpilot_engaged': is_openpilot_engaged,
'target position': current_location,
```

```

'currentspeed': currentspeed,
'currentVelocity': currentVelocity,
'current_vehicle_angle': round(current_vehicle_angle, 3),
'old_vehicle_angle': round(old_vehicle_angle, 3),
'max_steer_vehicle_angle': round(max_steer_angle, 3),
'steer_rate_limit': 0.5,
'current_steer': round(steer_now, 3),
'old_steer': round(steer_old, 3),
'leftLaneProb': round(leftProb, 3),
'rightLaneProb': round(rightProb, 3),
'desiredPathProb': round(desiredPathPro, 3),
'HasLead': hasLead,
'specifyCruise': cruise_control_based,
'awarenessStatus': awarenessStatus,
>alertStatus': alertStatus,
>alertText1': alertText1,
>alertText2': alertText2,
>alertSound': alertSound,
'detectionProbability': text_error_message,
'safety_procedure': safety_procedure}
return case, risk_priority

def lateralPlanOutputs():
    leftProb = 0.0
    rightProb = 0.0
    desiredPathPoints = []
    desiredPathPro = 0.0
    curvatures = []
    curvatureRates = []
    leftProb = sm['lateralPlan'].lProb
    rightProb = sm['lateralPlan'].rProb
    desiredPathPoints = sm['lateralPlan'].dPathPoints
    desiredPathPro = sm['lateralPlan'].dProb
    desire = sm['lateralPlan'].desire
    curvatures = sm['lateralPlan'].curvatures
    curvatureRates = sm['lateralPlan'].curvatureRates
    return leftProb, rightProb, desiredPathPro, desire

def longitudinalPlan():
    sm.update(4)
    hasLead = sm['longitudinalPlan'].hasLead
    speeds = sm['longitudinalPlan'].speeds
    accels = sm['longitudinalPlan'].accels
    jerks = sm['longitudinalPlan'].jerks
    vCruiseDEPRECATED = sm['longitudinalPlan'].vCruiseDEPRECATED
    cruise_control_based = sm['longitudinalPlan'].longitudinalPlanSource
    return cruise_control_based, hasLead

def liveParametersOutputs():
    sm.update(5)
    valid = sm['liveParameters'].valid

```

```

yawRate = sm['liveParameters'].yawRate
steerRatio = sm['liveParameters'].steerRatio
sensorValid = sm['liveParameters'].sensorValid
angleOffsetAverageDeg = sm['liveParameters'].angleOffsetAverageDeg
angleOffsetDeg = sm['liveParameters'].angleOffsetDeg
return valid, yawRate, steerRatio, sensorValid, angleOffsetAverageDeg, angleOffsetDeg

def humanDriverStateOutputs (humanFallbackMode):
    if humanFallbackMode == 1:
        sm.update(2)
        faceProb = 0.0
        leftEyeProb = 0.0
        rightEyeProb = 0.0
        leftBlinkProb = 0.0
        rightBlinkProb = 0.0
        distractedProb = 0.0
        distractedEyes = 0.0
        eyesOnRoad = 0.0
        phoneUse = 0.0
        faceProb = sm['driverState'].faceProb
        leftEyeProb = sm['driverState'].leftEyeProb
        rightEyeProb = sm['driverState'].rightEyeProb
        leftBlinkProb = sm['driverState'].leftBlinkProb
        rightBlinkProb = sm['driverState'].rightBlinkProb
        distractedProb = sm['driverState'].distractedProb
        distractedEyes = sm['driverState'].distractedEyes
        eyesOnRoad = sm['driverState'].eyesOnRoad
        phoneUse = sm['driverState'].phoneUse
        awarenessStatus = sm['driverMonitoringState'].awarenessStatus
        awarenessActive = sm['driverMonitoringState'].awarenessActive
        awarenessPassive = sm['driverMonitoringState'].awarenessPassive
    return awarenessStatus, awarenessActive, awarenessPassive

def carStateParameters ():
    vTargetLead = sm['controlsState'].vTargetLead
    vCruise = sm['controlsState'].vCruise
    aTarget = sm['controlsState'].aTarget
    #UI tests
    engageable = sm['controlsState'].engageable
    alertStatus = sm['controlsState'].alertStatus
    alertText1 = sm['controlsState'].alertText1
    alertText2 = sm['controlsState'].alertText2
    alertSound = sm['controlsState'].alertSound
    return alertStatus, alertText1, alertText2, alertSound

def safeData(dict_data, header):
    print('myList len:', len(dict_data))
    csv_file = "c1.csv"
    try:
        with open(csv_file, 'w') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=header)
            writer.writeheader()

```



```

for data in dict_data:
    writer.writerow(data)
    print('DataSaved,OK')
except IOError:
    print("error = data cannot saved")

def spawn_leading_vehicle(client, world, blueprint_library, vehicle):
    vehicle_location = vehicle.get_transform().location
    vehicle_waypoint = world.get_map().get_waypoint(vehicle_location)
    new_vehicle_waypoint = vehicle_waypoint.next(20)[0]
    new_vehicle_location = new_vehicle_waypoint.transform.location + carla.Location(0, 0, 2
    )
    new_vehicle_rotation = new_vehicle_waypoint.transform.rotation
    new_vehicle = blueprint_library.filter('vehicle.tesla.*')[1]
    batch = [
        carla.command.SpawnActor(new_vehicle, carla.Transform(new_vehicle_location,
            new_vehicle_rotation)).then(
            carla.command.SetAutopilot(carla.command.FutureActor, True))
    ]
    lead_vehicle_id = client.apply_batch_sync(batch)[0].actor_id
    return lead_vehicle_id

```

B.2 Assessing and Understanding Recorded Behaviours

The objective of this section is to analyse the behavioural responses collected from OpenPilot in the Carla simulator. These responses illustrate how the ALC system of OpenPilot achieves its primary functionality, specifically, maintaining a vehicle in the centre of its target lane.

The ALC system was tested with various environmental variables, such as speed and weather. As illustrated in Figure B.1, different weather scenarios are represented. The primary focus is to assess the ability of the ALC system in OpenPilot to detect lane lines on the target lane accurately. Gaining a deeper understanding of these aspects can enhance the analysis outcomes of the ALC system. This, in turn, can study its safety and reliability, making it a more reliable component of autonomous vehicle technology.

The capability of ALC to identify left and right lane lines can vary depending on environmental conditions. Factors like weather, lighting, and road surface affect how well the ALC system detects these lines. OpenPilot explains this detection process using probabilities, with scores ranging from 0 to 100. These scores are crucial in calculating the desired path for keeping a vehicle in its designated lane. The calculation considers the likelihood of detecting the left and right lane lines in images from the camera system. Therefore, OpenPilot relies on these probabilities to determine the confidence level of the ALC system in identifying lane lines for lane-keeping.

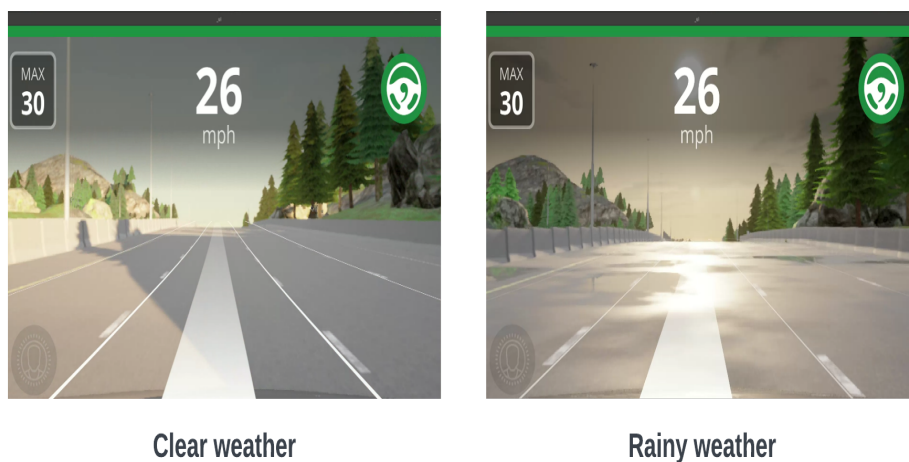


FIGURE B.1: Testing ALC system in OpenPilot under diverse weather settings

For instance, Figure B.2 illustrates the relationship between the detection probabilities of the left and right lane lines and the resulting desired path. These detections are based on various images obtained from the Carla simulator. By analysing changes in these detection probabilities, it becomes evident that the ALC system might generate an unsafe desired path when it fails to detect lane lines with high confidence scores.

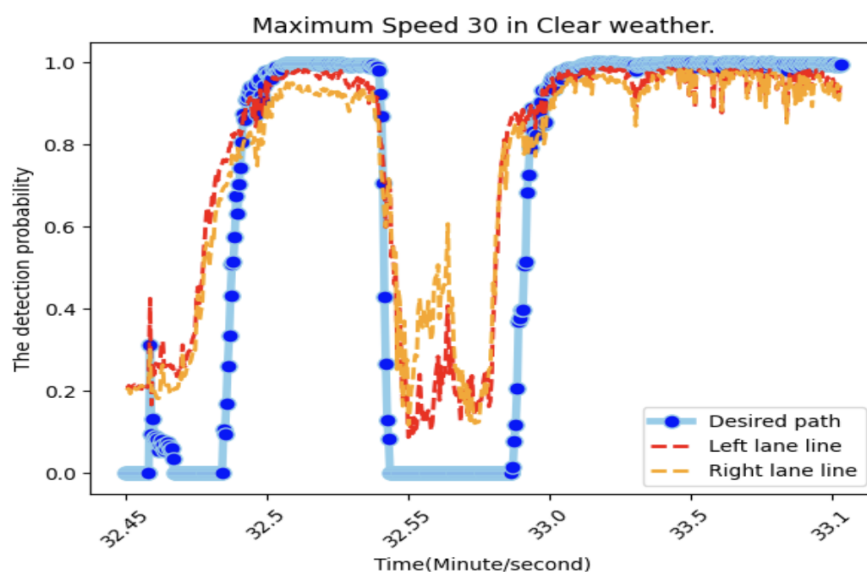


FIGURE B.2: Desired path probability over the time period, illustrating changes in lane detection scores

The ALC system primarily aims to maintain the vehicle's position within the target lane by recommending adjustments to the steering angle. These suggested changes are then applied to the vehicle's physical steering wheel. Furthermore, the steering angle adjustments are constrained within specific limits, specifically -70 and 70 degrees. These limits correspond to the vehicle's steering capabilities, ensuring that the steering

adjustments proposed by the ALC system can be executed by the vehicle's steering mechanism.

For a visual representation of this concept, refer to Figure B.3, which demonstrates how the steering angle evolves as the vehicle aligns itself within the target lane. In conjunction with the detection probabilities introduced in Figure B.2, this figure visually illustrates the ALC system's operation in lane detection and steering control, offering a clear understanding of its functionality.

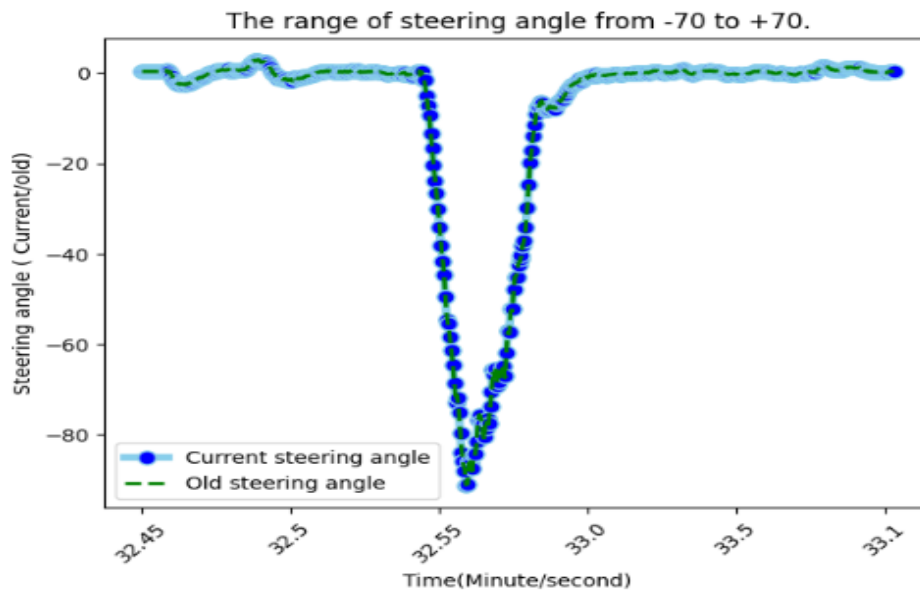


FIGURE B.3: Changes in the steering angle over time, demonstrating adjustments in the vehicle's physical wheels

Alongside lane detection and steering adjustments, OpenPilot also prioritises monitoring the human driver's attentiveness during the operation of the ALC system. The main objective of monitoring human driver behaviours is to ensure that the driver can play a supervisory role and intervene if the ALC system encounters hazardous driving events. Therefore, OpenPilot incorporates a Driver Monitoring System (DMS), which keeps track of various indicators of driver engagement. For instance, the Driver Monitoring System (DMS) verifies whether the human driver inside the vehicle maintains their hands on the steering wheel, and it also monitors their eyes and head to assess their level of awareness.

OpenPilot employs a system that prioritises safety by actively requesting driver intervention under certain circumstances. These intervention requests generally fall into two categories:

1. The system may send a *Take Control* request if a turn exceeds the steering limit, which can occur in particularly sharp turns or abrupt lane changes. For instance, Figure B.4 illustrates the monitoring process of applying steering angle adjustments, with a red line indicating the intervention request.

- OpenPilot's Driver Monitoring System (DMS) may not be able to verify the driver's awareness level, possibly due to the driver not keeping their hands on the steering wheel or showing signs of fatigue or inattention

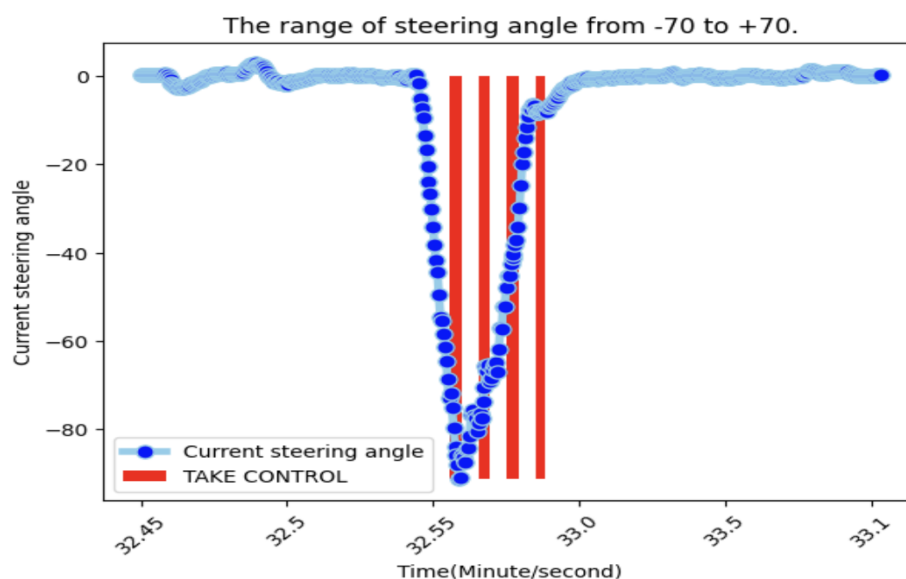


FIGURE B.4: OpenPilot's 'Take Control' request during excessive steering adjustment

When OpenPilot sends an intervention request, it allows a window of four seconds for the driver to respond and regain control of the vehicle. If no response is detected within a further two seconds, reaching six seconds since the initial request, OpenPilot activates an auditory notification to alert the driver. If the driver still does not respond, the system gradually reduces the vehicle's speed until it comes to a complete stop, ensuring the safety of the passengers and other road users.

B.3 Driver Monitoring System in Autonomous Vehicles

The DMS is designed to monitor the driver carefully and issue alerts upon detecting signs of drowsiness, distraction, or disengagement. This enhances safety and security during SDV operation. It is important to note that the DMS is not exclusive to fully autonomous vehicles; it also serves semi-automated systems where the human driver remains an essential part of the control loop.

One of the primary features of the DMS is active monitoring. This requires the driver to keep their hands on the steering wheel, even in a semi-automated system. It acts as a safety check, ensuring that the driver is prepared to resume control if and when hazardous driving events occur. This guarantees that a driver can intervene appropriately.

Besides the active monitoring feature, the DMS integrates sophisticated monitoring capabilities using an in-car camera. This feature employs advanced algorithms for facial

recognition and eye tracking to determine the driver's level of attention and readiness to take over the SDV's control. By detecting and tracking the position of the driver's eyes and head, the DMS confirms that the driver remains alert and focused on the road ahead. An example of a sensitive monitoring feature is provided in the subsection below.

B.3.1 Sensitive Monitoring Feature

The mechanism for determining sensitive monitoring features relies mainly on on-line facial recognition algorithms. For instance, the online facial recognition algorithm called Dlib² [120] can be used to measure the awareness level of a human driver by extracting the face variables of the driver during the operation of the ALC. The driver's eyes are the essential features for estimating the driver's awareness level. The eyes may blink during the extracting process, which must be considered when counting the number of images (frames) in which open eyes could not be detected. Therefore, the Eye Aspect Ratio (EAR) can be used to measure either lost or blinking frames in the sequence of images. The EAR is an example of a sensitive feature that could be used to measure the awareness level of a human driver. The summary of this algorithm is shown in Figure B.5, where the image processing unit, the state machine diagram of the awareness system and algorithm design are illustrated.

Specifically, the DMS in OpenPilot is designed to monitor the driver's level of awareness by using a variety of indicators, most commonly through the use of cameras that focus on the driver's face and upper body. In addition, the general operation of the DMS in OpenPilot can be described as follows:

1. **Facial Recognition:** OpenPilot uses facial recognition to ensure that the driver is facing forward and has their eyes open. If the driver looks away from the road for an extended period or closes their eyes, the system will typically issue a warning.
2. **Head Position:** The angle and orientation of the driver's head may also be monitored to identify where the driver is looking
3. **Audible and Visual Alerts:** If the DMS detects that the driver is not paying sufficient attention, it can issue audible and/or visual warnings to alert the driver.
4. **Escalation:** In extreme cases where the driver does not respond to alerts, the system may be designed to safely bring the car to a stop.
5. **Integration:** The DMS does not operate in isolation. It is typically integrated with other systems in the vehicle to create a comprehensive safety view. For instance, if

²Dlib [77] is a C++ toolkit that includes machine learning algorithms and tools for creating complex software, and its functionalities can be leveraged to detect faces and landmarks on them, which can then be used to train models for facial expression recognition.

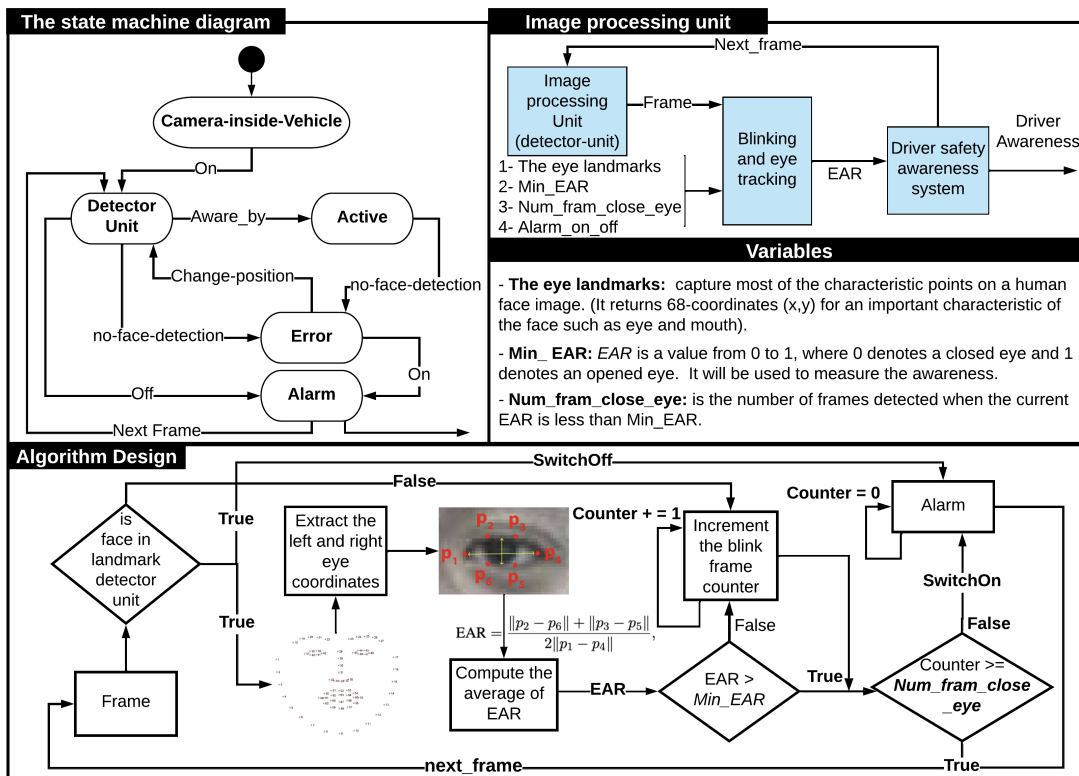


FIGURE B.5: Summary of the detection process of a sensitive monitoring feature based on the Dlib algorithm

the adaptive cruise control is following another car and the DMS detects that the driver is not paying attention, it can take more conservative actions like reducing the current speed of a vehicle.

Appendix C

Modelling High-Level System Aspects in ALC System

In this appendix, we present the complete Event-B version of the high-level system component interactions between the human drivers and the Automated Lane Centring (ALC) system, as outlined in Chapter 5. The complete Event-B models ¹ are presented in the following sections.

C.1 Context c0

context c0

sets

STAGE

POSITION

IMAGE

constants

Perception

Decision

Control

Intervention

Environment

Lane

init_position

STEERING_ANGLE

camera

path_recognition

compute_target_position

compute_target_steering_angle

¹The model is available as a Rodin archive at <https://tinyurl.com/SESS2022>.

move

axioms

@def-STAGE: partition(STAGE, {Perception}, {Decision}, {Control}, {Intervention}, {Environment})

@def-STEERING_ANGLE: STEERING_ANGLE = -100 .. 100

@typeof-Lane: Lane \subseteq POSITION

@typeof-init_position: init_position \in Lane

@typeof-camera: camera \in POSITION \rightarrow IMAGE

@typeof-path_recognition: path_recognition \in IMAGE \rightarrow \mathbb{P} (POSITION)

@typeof-compute_target_position: compute_target_position \in \mathbb{P} (POSITION) \rightarrow POSITION

@typeof-compute_target_steering_angle: compute_target_steering_angle \in \mathbb{P} (POSITION) \rightarrow STEERING_ANGLE

@typeof-move: move \in POSITION \times STEERING_ANGLE \rightarrow POSITION

@consistency:

\forall position · position \in POSITION \Rightarrow

move(position \mapsto compute_target_steering_angle(path_recognition(camera(position)))) \in Lane

end

C.2 Machine m0

machine m0

sees c0

variables

stage // stage of ACL system

position // The (physical) position of the vehicle

image // The image to be used as the input to the perception stage

steering_angle // The (actual) steering angle of the vehicle

desire_path // output of the perception stage

confident_score // output of the perception stage

target_position // output of the planning stage

target_steering_angle // output of the planning stage

invariants

@typeof-stage: stage \in STAGE

@typeof-position: position \in Lane // Safety: the position must be in Lane

@typeof-image: image = camera(position) // Consistency: The camera always show the image of the current position

@typeof-steering_angle: steering_angle \in STEERING_ANGLE


```

@typeof–desire_path: desire_path ⊆ POSITION
@typeof–perceived_confident: confident_score ∈ 0..100

@typeof–target_position: target_position ∈ POSITION
@typeof–target_steering_angle: target_steering_angle ∈ STEERING_ANGLE

// Deriving from the consistency
@Environment–consistency: stage = Environment ⇒ move(position ↦ steering_angle)
    ∈ Lane
events
event INITIALISATION
begin
  @init–stage: stage := Perception
  @init–position: position := init_position
  @init–image: image := camera(init_position)
  @init–steering_angle: steering_angle ∈ STEERING_ANGLE
  @init–desire_path: desire_path := ∅ // No desire path
  @init–confident_score: confident_score := 0 // No confident score
  @init–target_position: target_position ∈ POSITION // No initial target position (hence
    random)
  @init–target_steering_angle: target_steering_angle ∈ STEERING_ANGLE // No initial
    target steering angle (hence random)
end

event perception
when
  @grd1: stage = Perception
then
  @act1: stage := Decision
  @act2: desire_path := path_recognition(image)
  @act3: confident_score := 0..100
end

event decision
when
  @grd1: stage = Decision
then
  @act1: stage := Control
  @act2: target_position := compute_target_position(desire_path)
  @act3: target_steering_angle := compute_target_steering_angle(desire_path)
end

event control
when
  @grd1: stage = Control
then

```

```

@act1 : stage := Intervention
@act2 : steering_angle := target_steering_angle
end

event accept
when
@grd1 : stage = Intervention
@grd2 : move(position ↦ steering_angle) ∈ Lane
then
@act1 : stage := Environment
end

event correct
any angle
when
@grd1 : stage = Intervention
@grd2 : angle ∈ STEERING_ANGLE
@grd3 : move(position ↦ angle) ∈ Lane
then
@act1 : stage := Environment
@act2 : steering_angle := angle
end

event environment
when
@grd1 : stage = Environment
then
@act1 : stage := Perception
@act3 : position := move(position ↦ steering_angle)
@act2 : image := camera(move(position ↦ steering_angle))
end

end

```

C.3 Machine m1

```

machine m1
refines m0
sees c0
variables
stage // stage of ACL system

position // The (physical) position of the vehicle
image // The image to be used as the input to the perception stage
steering_angle // The (actual) steering angle of the vehicle

```

```

desire_path // output of the perception stage
confident_score // output of the perception stage

target_position // output of the planning stage
target_steering_angle // output of the planning stage

manual_drive // Manual drive
hands_on_steering_wheel // Hands on steering signal
auditory_notification
warning_message
sensitive_monitoring_features_detected
invariants
@typeof-hands_on_steering_wheel: hands_on_steering_wheel ∈ BOOL
@typeof-manual_drive: manual_drive ∈ BOOL
@typeof-auditory_notification: auditory_notification = TRUE ⇒ manual_drive = TRUE
@typeof-warning_message: warning_message ∈ BOOL
@typeof-sensitive_monitoring_features_detected:
    sensitive_monitoring_features_detected ∈ BOOL
@hands_on_steering_wheel: hands_on_steering_wheel = FALSE ⇒ auditory_notification =
    TRUE
@sensitive_monitoring_features_detected: sensitive_monitoring_features_detected =
    FALSE ⇒ auditory_notification = TRUE
@safety-Correction: stage = Intervention ∧ warning_message = FALSE ⇒ move(position
    ↦ steering_angle) ∈ Lane

events
event INITIALISATION extends INITIALISATION
begin
@init-manual_drive: manual_drive := FALSE
@init-hands_on_steering_wheel: hands_on_steering_wheel := TRUE
@init-warning_message: warning_message := FALSE
@init-auditory_notification: auditory_notification := FALSE
@init-sensitive_monitoring_features_detected: sensitive_monitoring_features_detected
    := TRUE
end

event perception extends perception
end

event decision extends decision
end

event control extends control
begin
@act3: warning_message := bool(move(position ↦ target_steering_angle) ∉ Lane)
end

```

event accept **refines** accept

when

@grd1: stage = Intervention

@grd2: warning_message = FALSE

@grd3: manual_drive = FALSE

then

@act1: stage := Environment

end

event correct **extends** correct

when

@grd4: manual_drive = TRUE \vee warning_message = TRUE

end

event environment **extends** environment

end

event hands_on_wheel

when

@grd1: hands_on_steering_wheel = FALSE

then

@act1: hands_on_steering_wheel := TRUE

@act2: auditory_notification := bool(sensitive_monitoring_features_detected = FALSE)

end

event hands_off_wheel

when

@grd1: hands_on_steering_wheel = TRUE

then

@act1: hands_on_steering_wheel := FALSE

@act2: auditory_notification := TRUE

@act3: manual_drive := TRUE

end

event detect_sensitive_monitoring_features

when

@grd1: sensitive_monitoring_features_detected = FALSE

then

@act1: sensitive_monitoring_features_detected := TRUE

@act2: auditory_notification := bool(hands_on_steering_wheel = FALSE)

end

event lost_sensitive_monitoring_features

when

@grd1: sensitive_monitoring_features_detected = TRUE

then

@act1: sensitive_monitoring_features_detected := FALSE

```
@act2: auditory_notification := TRUE
@act3: manual_drive := TRUE
end

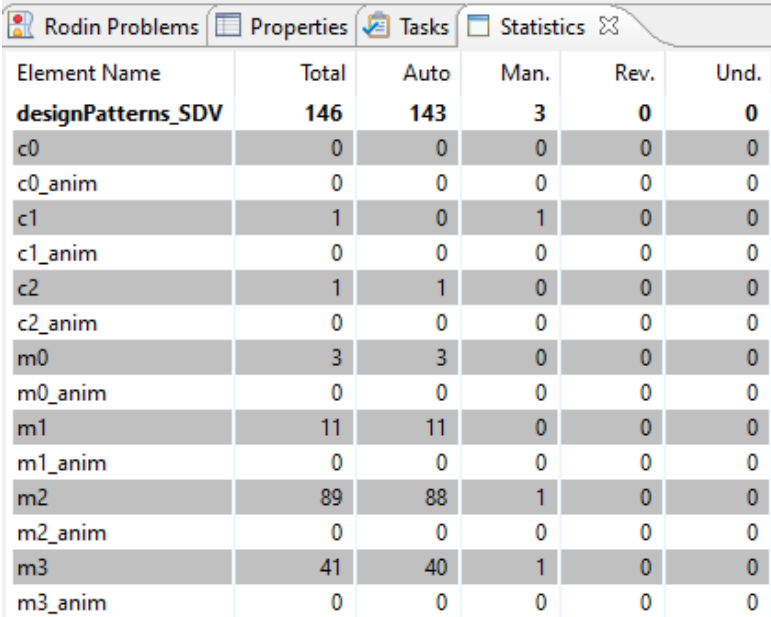
event autonomos_drive
when
  @grd1: manual_drive = TRUE
  @grd2: hands_on_steering_wheel = TRUE
  @grd3: sensitive_monitoring_features_detected = TRUE
then
  @act1: auditory_notification := FALSE
  @act2: manual_drive := FALSE
end

end
```


Appendix D

Modelling Patterns for SDV Systems

In this appendix, we present the complete Event-B version of the modelling patterns for the Self-Driving Vehicle (SDV) system, as outlined in Chapter 6. The majority of the proof obligations in these patterns were verified either automatically using Rodin provers or with the assistance of additional external prover plug-ins, such as SMT solvers (as illustrated in Figure D.1).



Element Name	Total	Auto	Man.	Rev.	Und.
designPatterns_SDV	146	143	3	0	0
c0	0	0	0	0	0
c0_anim	0	0	0	0	0
c1	1	0	1	0	0
c1_anim	0	0	0	0	0
c2	1	1	0	0	0
c2_anim	0	0	0	0	0
m0	3	3	0	0	0
m0_anim	0	0	0	0	0
m1	11	11	0	0	0
m1_anim	0	0	0	0	0
m2	89	88	1	0	0
m2_anim	0	0	0	0	0
m3	41	40	1	0	0
m3_anim	0	0	0	0	0

FIGURE D.1: Modelling Patterns Prover Statistics.

The complete Event-B models ¹ are presented in the following sections.

¹The Event-B models for developing patterns of SDV systems are available as a Rodin archive at https://drive.google.com/drive/folders/1D7rVAJKCEh_-rVr9hKZZYdFDCxAP8r3F.

D.1 Abstract Level

D.1.1 Context c0

```

context c0
sets
Semi_FUNCTION /* function might be : On, OFF */
constants
POSITION /* set of positions that an SDV (physical vehicle) can move to them */
ODD /* subset of position */
init_position /* initial position of SDV */
ON /* Semi-automation system is working*/
OFF /* Semi-automation system is deactivated*/
axioms
/* Type of POSITION */
@typeof-POSITION: POSITION =  $\mathbb{N} \times \mathbb{Z}$ 
/* ODD involves a set of positions */
@typeof-ODD: ODD  $\subseteq$  POSITION
/* the initial position of an SDV is located in the ODD */
@typeof-init_position: init_position  $\in$  ODD
/* System status */
@def-system-mode: partition(Semi_FUNCTION, {ON}, {OFF})
end

```

D.1.2 Machine m0

```

machine m0 sees c0
/*
* Abstract machine (M0) aims to capture how the constraints
* imposed by the driving environment can affect the movement of an SDV.
*/
variables
/*  $\times \times \times \times$  */
* M0 ----- variables
 $\times \times \times \times$ */
/* (CA1) The (physical) position of the vehicle */
SDV_POSITION_env
/* System status is ON or OFF */
Sys_status

invariants
/* SC0: the SDV must always be located inside the limited ODD */
@safety: SDV_POSITION_env  $\in$  ODD
@ALC_mode: Sys_status  $\in$  Semi_FUNCTION

events

```



```

event INITIALISATION
  begin
    /* (F1) shows the current position of an SDV */
    @init-position: SDV_POSITION_env := init_position
    /*initialise ALC status is ON*/
    @init-mode: Sys_status := OFF
  end

  /*
  * activation
  */
  event system_on
    where
      @grd1: Sys_status = OFF
    then
      @act1: Sys_status := ON
    end

  /*
  * deactivation
  */
  event system_off
    where
      @grd1: Sys_status = ON
    then
      @act1: Sys_status := OFF
      @reset-position: SDV_POSITION_env := init_position
    end

  /*
  * This event changes a current position of an SDV into a new position inside the ODD.
  */
  event move
    any
      new_position
    where
      /* SR0: the SDV must travel into a new position inside the ODD*/
      @grd1: new_position ∈ ODD
    then
      /* moved into a new position inside the ODD*/
      @act1: SDV_POSITION_env := new_position
    end

end

```

D.2 First Refinement

D.2.1 Context c1

context c1 extends c0

constants

/ lateral range */*

LATERAL

max_lat

min_lat

/ longitudinal range */*

LONGITUDINAL

max_lon

min_lon

*/**

** A function assumes that an SDV actuate lateral\longitudinal vehicle variables*

** to reach a new position (actuation task).*

**/*

move

axioms

@typeof-max_lat: max_lat $\in \mathbb{Z}$

@typeof-min_lat: min_lat $\in \mathbb{Z}$

@min_max_steering: min_lat < max_lat

/ these just an example of lateral range */*

@def-lateral: LATERAL = min_lat .. max_lat

@typeof-max_lon: max_lon $\in \mathbb{Z}$

@typeof-min_lon: min_lon $\in \mathbb{Z}$

@min_max_lon: min_lon < max_lon

/ these just an example of longitudinal range */*

@def-longitudinal: LONGITUDINAL = min_lon .. max_lon

*/**

** This function assumes that SDV actuates a longitudinal variable, such as speed,*

** and a lateral variable, such as steering angle , in order to reach a specific*

** position where SDV moves into multiple positions inside the limited ODD.*

**/*

@def-MOVE: move =

(

λlong \mapsto lat \mapsto speed \mapsto angle · long \mapsto lat \in

POSITION \wedge

speed \in LONGITUDINAL \wedge

```

    angle ∈ LATERAL |
    {i · i ∈ 1..3 | long + i ↦ lat + i × speed × angle}
  )
theorem @typeof-move: move ∈ POSITION × LONGITUDINAL × LATERAL →  $\mathbb{P}_1$ (
    POSITION)

end

```

D.2.2 Machine m1

machine m1 refines m0

sees c1

```

/*
 * This machine (M1) investigate how the lateral and longtiudinal variables of an SDV is
   modified
 * to reach a new position inside the limited ODD.
 */

```

variables

```

/*××××*
 * M0 ----- variables
 ××××*/
 /* (CA1) The (physical) position of the vehicle */
 SDV_POSITION_env
 /* System status is ON or OFF */
 Sys_status

/*××××*
 * M1 ----- variables
 ××××/
 /* (CA2) The (actual) lateral variable of the SDV */
 LATERAL_VAR
 /* (CA3) The (actual) longitudinal variable of the SDV */
 LONGITUDINAL_VAR

```

invariants

```

/*lateral variable must be in a defined range of lateral variable in SDV (abstract constant)*/
@typeof-lateral: LATERAL_VAR ∈ LATERAL
/*longitudinal variable must be in a defined range of longitudinal variable in SDV (abstract
  constant)*/
@typeof-longitudinal: LONGITUDINAL_VAR ∈ LONGITUDINAL

```

events

```

event INITIALISATION extends INITIALISATION
begin
  /*(F2) The current lateral variable of an SDV that is randomly initialised*/
  @init-steering_angle: LATERAL_VAR :∈ LATERAL
  /*(F3) The current longitudinal variable of an SDV that is randomly initialised */
  @init-speed: LONGITUDINAL_VAR :∈ LONGITUDINAL
end

/*
 * This event identifies lateral/longitudinal variables autonomously
 */
event auto_actuating
any
  auto_lat
  auto_lon
where
  /* System status is ON*/
  @grd1: Sys_status = ON
  /* lateral definition */
  @grd2: auto_lat ∈ LATERAL
  /* longitudinal definition */
  @grd3: auto_lon ∈ LONGITUDINAL
  /* (SR1): system's modification leads to ODD */
  @grd4: move(SDV_POSITION_env ↦ auto_lon ↦ auto_lat) ⊆ ODD
then
  @act1: LATERAL_VAR := auto_lat
  @act2: LONGITUDINAL_VAR := auto_lon
end

/*
 * This event identifies lateral/longitudinal variables manually
 */
event manual_actuating
any
  manual_lat
  manual_lon
where
  @grd1: manual_lat ∈ LATERAL
  @grd2: manual_lon ∈ LONGITUDINAL
  /*(SR2): driver's modification leads to lane */
  @grd3: move(SDV_POSITION_env ↦ manual_lon ↦ manual_lat) ⊆ ODD
then
  /* new steering depends on how a driver specified steering angle */
  @act1: LATERAL_VAR := manual_lat
  @act2: LONGITUDINAL_VAR := manual_lon
end

```

```

/*
 * A extended event moves an SDV in its ODD based on the (new) lateral and longitudinal
   variables
 * of a system or human driver, where a movement must lead into the ODD
 */
event move extends move
where
  /* new (target) position must be within set of position inside the ODD*/
  @grd2: new_position ∈ move(SDV_POSITION_env ↦ LONGITUDINAL_VAR ↦ LATERAL_VAR
    )
end

event system_on extends system_on
end

event system_off extends system_off
end

end

```

D.3 Second Refinement

D.3.1 Context c2

```

context c2 extends c1
sets

  /* set of images obtaining from the driving environment (target lane) */
  SENSING_DATA
  /* set of Perceived environmental features */
  PERCEIVED_FEATURES
  /* partition shows the different stages of semi–automation system, such as Perception,
    planning and etc*/
  Auto_STAGE

constants

  /* An abstract constant uses to show how system gets sensing data from the driving
    environment
   * (sensor of perception component)
   */
  sensor
  /* An abstract constant shows the expected results of a received sensing data
   * (detection process of perception component )
   */
  Identify_features

```

```

/*
 * An abstract constant shows how system accomplishes the OEDR task based on the sensing
   images
 * (recognition process of perception component )
 */
OEDR_task
/*
 * An abstract constant shows a accuracy of identifying Perceived features in a received
   sensing data
 * For instance, score from 0 to 100 indicates the accuracy of the identification process in the
   incoming sensing data.
 * 0 is very low accuracy, while 100 is a high accuracy
 */
Accuracy
/*
 * An abstract constant shows how system computes planning task, e.g.,
 * 1) system selects one (target) position from a set of positions obtained from the OEDR
   function
 */
specify_target_position
/*
 * An abstract constant shows how system computes planning task , e.g.,
 * 2) system identifies a change of steering angle based on the target position and a current
   steering angle.
 */
compute_lateral
/*
 * An abstract constant shows how system computes planning task , e.g.,
 * 3) system identifies a change of LONGITUDINAL based on the target position and a
   current LONGITUDINAL.
 */
compute_longitudinal

/*
 * These constants used to distinguish between different possible stages of semi–automation
   system
 */
/* perception's stage*/
Perception
/* decision's stage*/
Decision
/* control's stage */
Control
/* intervention's stage, indicates that system could ask driver to take a control of SDV*/
Intervention
/* Autonomous driving's stage, indicates that system actuate lateral and longitudinal
   variables autonomously to reach a (identified) target position */

```

AutonomousDriving

axioms

```

/* possible stages of semi-automation system */
@def-STAGE: partition(Auto_STAGE, {Perception}, {Decision}, {Control}, {
  AutonomousDriving}, {Intervention})
/* range of Confidence score from 0 to 100 */
@score: Accuracy = 0 .. 100

/*
* Perception task
* (three functions)
*/
// 1) SENSING_DATA obtains from sensor based on SDV's position
@typeof-sensor: sensor ∈ POSITION → SENSING_DATA
/* 2) function shows the expected results of a received SENSING_DATA. */
@typeof-identify-features: Identify_features ∈ (SENSING_DATA × PERCEIVED_FEATURES)
  → Accuracy
/* 3) function recognises the path (set of position ) based on the left/right lane lined, image
and confidence score */
@typeof-path_recognition: OEDR_task ∈ (SENSING_DATA × PERCEIVED_FEATURES ×
  Accuracy) → P(POSITION)

/*
* End of Perception task
*/

/*
* planning task
* (three functions)
*/
/* 1) function to specify a target position based on a set of positions obtained from OEDR
recognition */
@typeof-compute_target_position: specify_target_position ∈ P(POSITION) → POSITION
/* 2) function to compute the lateral variable to reach a specific position */
@typeof-compute_lateral: compute_lateral ∈ POSITION × LATERAL → LATERAL
/* 3) function to compute the longitudinal variable to reach a specific position */
@typeof-compute_LONGITUDINAL: compute_longitudinal ∈ POSITION × LONGITUDINAL
  → LONGITUDINAL

/*
* End of planning task
*/

/*
* This is the aim of system, which means that there is always one lateral and longitudinal
variable

```

```

* can be used to move an SDV into a position inside the limited ODD
*/
// @aim:  $\forall$  SDV_position · SDV_position  $\in$  POSITION  $\Rightarrow$ 
// ( $\exists$  SDV_lateral, SDV_longitudinal · SDV_lateral  $\in$  LATERAL  $\wedge$  SDV_longitudinal  $\in$ 
LONGITUDINAL  $\Rightarrow$ 
// move (SDV_position  $\mapsto$  SDV_longitudinal  $\mapsto$  SDV_lateral)  $\subseteq$  ODD
//)

```

end

D.3.2 Machine m2

machine m2 refines m1

sees c2

```

/*
* This machine (M2) studies how the ALC system accomplishes its autonomous operations ,
such as perception, decision–making and control,
* to identify the steering angle that autonomously moves an SDV into a position inside the
target lane
*/

```

variables

```

/* $\times\times\times\times\times$ 
* M0 ----- variables
 $\times\times\times\times\times\times$ 
/* (CA1) The (physical) position of the vehicle */
SDV_POSITION_env
/* System status is ON or OFF */
Sys_status

/* $\times\times\times\times\times$ 
* M1 ----- variables
 $\times\times\times\times\times$ 
/* (CA2) The (actual) lateral variable of the SDV */
LATERAL_VAR
/* (CA3) The (actual) longitudinal variable of the SDV */
LONGITUDINAL_VAR

/* $\times\times\times\times\times$ 
* M2 – variables
 $\times\times\times\times\times$ 
/* (F3) The image would obtain from sensor, which can be used as the input to the perception
stage*/
sensing_data_env

/*(CA4) indicates some features in a received image */

```


`generic_features`

`/*(CA4) set of positions seen as the path that navigates an SDV inside its limited ODD*/
generic_path`

`/* (CA4) score to show the accuracy of detection process for seeing image and identifying
path */
accuracy`

`/* (CA5) a new (target) position that identifies form a set of positions (path) */
selectPos`

`/* (CA5) a target steering angle that would be used to autonomously move an SDV from
current position into target position */
selectLat`

`/* (CA5) a required change of steering angle from a current steering to a new (target) steering
*/
selectLon`

`/* shows the different stages of ALC system, such as Perception, planning and etc */
stage`

`/* A boolean flag indicates that an SDV is ready to move*/
signal_flag`

invariants

`@typeof—stage: stage ∈ Auto_STAGE
@typeof_PerceivedFeatures: generic_features ∈ PERCEIVED_FEATURES
@typeof—accuracy: accuracy ∈ Accuracy
@typeof—path: generic_path ⊆ POSITION
@typeof—sel_position: selectPos ∈ POSITION
@typeof—sel_LATERAL: selectLat ∈ LATERAL
@typeof—sel_LONGITUDINAL: selectLon ∈ LONGITUDINAL
@typepf—signal_movement: signal_flag ∈ BOOL`

`/*
* The gluing inv1 invariant states that the system might be in any stage when a SDV system
* aims to identify a new steering angle (manual or autonomous)
*/
@gluing_inv1: Sys_status = ON ⇒ stage ∈ {Perception, Decision, Control, Intervention,
AutonomousDriving}`

`/*`

```

* (SR2.1): the perception module must update the sensing information according to the
  current position of an SDV.
* Consistency: The sensor always shows the sensing data for the current position of SDV
*/
@consistency: sensing_data_env = sensor(SDV_POSITION_env)

/*
* identify the perceived environmental features based on the sensing data
*/
@detecation_task: stage = Perception ⇒ sensing_data_env ∈ ran(sensor) ∧
  (sensing_data_env ↦ generic_features ↦ accuracy) ∈ dom(OEDR_task)

/*
* recognize the set of positions (path) according to the interpretation of sensing data
*/
@recognition_task: stage = Decision ⇒ generic_path ∈ ran(OEDR_task)

/*
* target position identified based on the perceived path
*/
@selPosition: stage = Decision ∧ accuracy ≥ 80 ∧ generic_path ⊆ ODD ⇒
  generic_path ∈ dom(specify_target_position)

/*
* target position identified based on the perceived path
*/
@selLatLon: stage = Control ∧ accuracy ≥ 80 ∧ generic_path ⊆ ODD ⇒
  (selectPos ↦ LATERAL_VAR) ∈ dom(compute_lateral)
  ∧
  (selectPos ↦ LONGITUDINAL_VAR) ∈ dom(compute_longitudinal)

/*
* This invariant ensures that ALC system never exceeds the defined range of steering angle
  when the new (target) steering angle is used
* to move an SDV into a new (target) position inside the target lane
*/
@Environment-consistency: stage = AutonomousDriving
  ⇒ move(SDV_POSITION_env ↦ selectLon ↦ selectLat) ⊆ ODD

```

events

event INITIALISATION **extends** INITIALISATION

begin

```

/* sensor is ready to provide an image for the initial position */
@init-image: sensing_data_env := sensor (init_position)
/* the stage of system starts with perception module */
@init-stage: stage := Perception
/* No desire path (hence empty)*/

```

```

@init-generic_path: generic_path := ∅
/* No confident score (hence zero) */
@init-accuracy: accuracy ∈ Accuracy
/* No initial selected position (hence random) */
@init-selectPos: selectPos ∈ POSITION
/* No initial selected steering angle (hence random) */
@init-selectLATERAL: selectLat ∈ LATERAL
/* No initial selected speed (hence random) */
@init-selectSpeed: selectLon ∈ LONGITUDINAL
/* No initial perceived features (hence random) */
@typeof_perFeatures: generic_features ∈ PERCEIVED_FEATURES
/* signal of movement */
@typeof_signal: signal_flag := FALSE
end

/*
* This event performs the perception task. It involves three main subtasks:
* 1) obtain sensing data from sensor based on SDV's position
* 2) identify accuracy of detection based on the given perceived features and sensing data
* 3) recognise new path (set of position ) based on the sensing data, perceived features and
  accuracy
*/
event perception
any
gf
when
@grd1: gf ∈ PERCEIVED_FEATURES
@grd2: Sys_status = ON
@grd4: stage = Perception
then
/* obtain an sensing data from sensor based on SDV's position */
@act1: sensing_data_env := sensor (SDV_POSITION_env)
/* specify some features to be detected in the sensing data */
@act2: generic_features := gf
/* identify accuracy based on the detection results of perceived \specific features from
  sensing data */
@act4: accuracy := Identify_features(sensing_data_env ↦ generic_features)
/* recognise the path (set of position ) based on the perceived features, sensing data and
  accuracy */
@act5: generic_path := OEDR_task(sensing_data_env ↦ generic_features ↦ accuracy)
/* change a stage of system to be in Decision */
@act6: stage := Decision
end

/*
* This event checks whether new path has been identified using a low or high accuracy.
* Note: if a accuracy is less than 80, we assume that a low accuracy score

```

```

*/
event violationExpectedFeatures
  where
    @grd1: Sys_status = ON
    @grd2: stage = Decision
    /*(F4): reason about the identification of environmental features */
    @grd3: accuracy < 80  $\vee$  generic_path  $\not\subseteq$  ODD
  then
    /*
    * (SR8): if the perception module identifies the desired path with a low confidence score,
    * the ALC system will issue a request to intervene.
    * Therefore, the stage of ALC system changed to be in 'Intervention'
    */
    @act1: stage := Intervention
end

/*
* This event shows how an ALC system performs the planning task in order to identify a
* target position and a change of steering angle
* when the stage of the ALC is in Decision, and the confidence score is greater than or equal
* 80
* Note: if a confidence score is more or equal 80, we assume that a high confidence score
*/
event decision
  when
    @grd1: Sys_status = ON
    @grd2: stage = Decision
    @grd3: accuracy  $\geq$  80  $\wedge$  generic_path  $\subseteq$  ODD
  then
    /* new (target) position obtained from identified path*/
    @act1: selectPos := specify_target_position(generic_path)
    /* specify lat and long based on the selected position */
    @act2: selectLat := compute_lateral(selectPos  $\mapsto$  LATERAL_VAR)
    @act3: selectLon := compute_longitudinal(selectPos  $\mapsto$  LONGITUDINAL_VAR)
    /* stage changed to be in Control */
    @act4: stage := Control
end

/*
* This event shows how ALC performs the actuation task for moving an SDV into a target
* position when the stage of the ALC is in Control.
*/
event control
  when
    @grd1: Sys_status = ON
    @grd2: stage = Control

```

```

/* selected LONGITUDINAL and steering must be in the defined range */
@grd3: selectLat ∈ LATERAL ∧ selectLon ∈ LONGITUDINAL
/* based on a change of a LONGITUDINAL and steering, the movement of an SDV must
   lead to a position inside the ODD */
@grd4: move(SDV_POSITION_env ↦ selectLon ↦ selectLat) ⊆ ODD
then
/* the stage of ALC changed to be in 'AutonomousDriving' */
@act1: stage := AutonomousDriving
end

```

event violationLimit

where

```

@grd1: Sys.status = ON
@grd2: stage = Control
/*(F5): violates physical actuator component*/
@grd3: selectLat ∉ LATERAL ∨ selectLon ∉ LONGITUDINAL ∨
move(SDV_POSITION_env ↦ selectLon ↦ selectLat) ⊈ ODD

```

then

```
@act1: stage := Intervention
```

end

/*

* This is a refined event from an abstract event auto_actuating for allowing a target steering angle (autonomous steering angle) to be used

* for moving an SDV in its target lane when the stage of the autonomous controller is in AutonomousDriving

*/

event auto_driving **refines** auto_actuating

where

```

@grd1: Sys.status = ON
@grd2: selectLat ∈ LATERAL
@grd3: selectLon ∈ LONGITUDINAL
@grd4: move(SDV_POSITION_env ↦ selectLon ↦ selectLat) ⊆ ODD
@grd5: stage = AutonomousDriving
@grd6: signal_flag = FALSE

```

then

/* setting the lateral and longitudinal variables autonomously */

```
@act1: LATERAL_VAR := selectLat
```

```
@act2: LONGITUDINAL_VAR := selectLon
```

/* ready to move */

```
@act3: signal_flag := TRUE
```

with

/* a change of lateral and longitudinal are replaced by a change identified by the autonomous operations */

```
@auto_lat: auto_lat = selectLat
```

```

@auto_lon: auto_lon = selectLon
end

/*
 * This is a refined event from an abstract event manual_actuating for covering how a human
   driver intervenes to take over control of an SDV
 * when the stage of the system is in Intervention.
 */
event manual_driving extends manual_actuating
where
  @grd4: stage = Intervention
  @grd5: signal_flag = FALSE
then
  /* ready to move */
  @act3: signal_flag := TRUE
end

/*
 * This is an extended event for using either a manual or autonomous steering angle to move
 * the SDV inside its target lane
 */
event move extends move
where
  @grd3: signal_flag = TRUE
  @grd4: stage ∈ {AutonomousDriving , Intervention}
then
  @act2: stage := Perception
  @act3: sensing_data_env := sensor (new_position)
  @act4: signal_flag := FALSE
end

event system_on extends system_on
end

/*
 * This is an extended event to reset a stage of an autonomous controller to be at
 * the perception stage when a system is switched off
 */
event system_off extends system_off
then
  /* ALC_STAGE reset into initialization value*/
  @restALC: stage := Perception
  @rest_image: sensing_data_env := sensor (init_position)
  @rest_signal_flag: signal_flag := FALSE
end
end

```

D.4 Third Refinement

D.4.1 Machine m3

machine m3 **refines** m2

sees c2

/*

* This machine (M3) studies how the awareness level of a driver can impact autonomous operations. It specifically discusses how SDV systems ensure that human drivers remain responsive when the system issues a request to intervene

*/

variables

/*××××*

* M0 ----- variables

××××*/

/* (CA1) The (physical) position of the vehicle */

SDV_POSITION_env

/* System status is ON or OFF */

Sys.status

/*××××*

* M1 ----- variables

××××*/

/* (CA2) The (actual) lateral variable of the SDV */

LATERAL_VAR

/* (CA3) The (actual) longitudinal variable of the SDV */

LONGITUDINAL_VAR

/*××××

* M2 – variables

××××*/

/* (F3) The image would obtain from sensor, which can be used as the input to the perception stage*/

sensing_data_env

/*(CA4) indicates some features in a received image */

generic_features

/*(CA4) set of positions seen as the path that navigates an SDV inside its limited ODD*/

generic_path

/* (CA4) score to show the accuracy of detection process for seeing image and identifying path */

accuracy

/* (CA5) a new (target) position that identifies form a set of positions (path) */
 selectPos

/* (CA5) a target steering angle that would be used to autonomously move an SDV from
 current position into target position */
 selectLat

/* (CA5) a required change of steering angle from a current steering to a new (target) steering
 */
 selectLon

/* shows the different stages of system, such as Perception, planning and etc */
 stage

/* A boolean flag indicates that an SDV is ready to move*/
 signal_flag

/*××××
 *M3 – variables
 ××××/

/* (CA8) indicates the driver's input to compute awareness level */
 driver_input

/* (CA7) shows the awareness level of a driver */
 awarenessLevel

/* (F6) indicates that semi-automated system issues an intervention request */
 intervRequest

invariants

/* driver either aware (TRUE) or unaware (FALSE)*/
 @typeof-**awarenessLevel**: awarenessLevel ∈ **BOOL**

/* DMS uses the driver's input to compute the awareness level of a driver */
 @type-**driverInput**: driver_input ∈ **BOOL**

/* semi-system may issue a request to intervene*/
 @typeof-**intervention**: intervRequest ∈ **BOOL**

@**awarenesslevel**: awarenessLevel = TRUE ⇒ driver_input = TRUE

@**driverAware**: awarenessLevel = TRUE ⇒ stage ∈ {Perception , Decision , Control ,
 Intervention , AutonomousDriving}

@**interventionRequest**: intervRequest = TRUE ⇒ stage = Intervention

@**intervCases**: intervRequest = TRUE

⇒

accuracy < 80 ∨

generic_path ∉ ODD ∨

selectLat ∉ LATERAL ∨


```

selectLon  $\notin$  LONGITUDINAL  $\vee$ 
move(SDV_POSITION_env  $\mapsto$  selectLon  $\mapsto$  selectLat)  $\notin$  ODD

```

events

event INITIALISATION **extends** INITIALISATION

begin

/ No initial driver's input (hence FALSE)*/*

@init-driverInput: driver_input := FALSE

/ initial awareness level is FALSE, means a driver is unaware of driving task */*

@init-awareness_level: awarenessLevel := FALSE

/ initial intervention request is FALSE, means the system does not issue a request to intervene */*

@init-InterRequest: intervRequest := FALSE

end

event DMS_driver_input_detect

any d.input */* human monitoring feature*/*

when

@grd1: d.input = FALSE

@grd3: awarenessLevel = FALSE

then

@act1: driver_input := TRUE

@act2: awarenessLevel := TRUE

end

event DMS_driver_input_remove

when

@grd2: driver_input = TRUE

@grd3: awarenessLevel = TRUE

then

@act1: driver_input := FALSE

@act2: awarenessLevel := FALSE

end

event perception **extends** perception

when

@grd5: awarenessLevel = TRUE

end

event violationExpectedFeatures **extends** violationExpectedFeatures

where

@grd4: awarenessLevel = TRUE

@grd5: intervRequest = FALSE

then

@act2: intervRequest := TRUE

end

```

event decision extends decision
when
  @grd4: awarenessLevel = TRUE
end

```

```

event control extends control
when
  @grd5: awarenessLevel = TRUE
  @grd6: intervRequest = FALSE
end

```

```

event violationLimit extends violationLimit
where
  @grd4: awarenessLevel = TRUE
  @grd5: intervRequest = FALSE
then
  @act2: intervRequest := TRUE
end

```

```

event auto_driving extends auto_driving
when
  @grd7: awarenessLevel = TRUE
  @grd8: intervRequest = FALSE
end

```

```

event manual_driving extends manual_driving
when
  @grd6: awarenessLevel = TRUE
  @grd7: intervRequest = TRUE
then
  @act4: intervRequest := FALSE
end

```

```

event auto_move refines move
any
  new_position
where
  // inherited elements
  @grd1: new_position ∈ ODD
  @grd2: new_position ∈ move(SDV_POSITION_env ↦ LONGITUDINAL_VAR ↦
    LATERAL_VAR)
  @grd3: signal_flag = TRUE
  /* new guards*/

```

```

@grd4: stage = AutonomousDriving
@grd5: awarenessLevel = TRUE
/* no intervention request */
@grd6: intervRequest = FALSE

then
// inherited elements
@act1: SDV_POSITION_env := new_position
@act2: stage := Perception
@act3: sensing_data_env := sensor(new_position)
@act4: signal_flag := FALSE
end

event manual_move refines move
any
  new_position // inherited element
where
  @grd1: new_position ∈ ODD // inherited element
  @grd2: new_position ∈ move(SDV_POSITION_env ↦ LONGITUDINAL_VAR ↦
    LATERAL_VAR) // inherited element
  @grd3: signal_flag = TRUE // inherited element
  /* new guards */
  @grd4: stage = Intervention
  @grd5: awarenessLevel = TRUE
  @grd6: intervRequest = TRUE

then
  @act1: SDV_POSITION_env := new_position // inherited element
  @act2: stage := Perception // inherited element
  @act3: sensing_data_env := sensor(new_position) // inherited element
  @act4: signal_flag := FALSE // inherited element
  /* new action */
  @act5: intervRequest := FALSE
end

event system_on extends system_on
then
  /* reset variables of this machine */
// @rst-driverFeature: generic_driverFeature := ∅
  @rst-driverInput: driver_input := FALSE
  @rst-awareness_level: awarenessLevel := FALSE
  @rst-InterRequest: intervRequest := FALSE
end

event system_off extends system_off
then
  /* reset variables of this machine */

```

```
// @rst-driverFeature: generic_driverFeature := ∅  
@rst-driverInput: driver_input := FALSE  
@rst-awareness_level: awarenessLevel := FALSE  
@rst-InterRequest: intervRequest := FALSE  
end  
end
```

Appendix E

Modelling LKA and DMS Functions in the ALC System

In this appendix, we present the complete Event-B version of the application of the modelling patterns for the Lane Keeping Assist (LKA) and Driver Monitoring System (DMS) functions in the Automated Lane Centring (ALC) system, as outlined in Chapter 7. The majority of the proof obligations in modelling these functions were verified either automatically using Rodin provers or with the assistance of additional external prover plug-ins, such as SMT solvers (as illustrated in Figure E.1).

Element Name	Total	Auto	Manual	Rev.	Und.
LKA_DMS	194	191	3	0	0
c0	0	0	0	0	0
c0_anim	0	0	0	0	0
c1	1	0	1	0	0
c1_anim	0	0	0	0	0
c2	1	1	0	0	0
c2_anim	0	0	0	0	0
m0	3	3	0	0	0
m0_anim	0	0	0	0	0
m1	7	7	0	0	0
m1_anim	0	0	0	0	0
m2	126	125	1	0	0
m2_anim	0	0	0	0	0
m3	56	55	1	0	0
m3_anim	0	0	0	0	0

FIGURE E.1: Modelling LKA and DMS functions, prover statistics

The complete Event-B models ¹ are presented in the following sections.

¹The Event-B models for developing design patterns of SDV systems are available as a Rodin archive at https://drive.google.com/drive/folders/1D7rVAJKCEh_-rVr9hKZZYdFDCxAP8r3F.

E.1 Abstract Level

E.1.1 Context c0

```

context c0
sets
  ALC_FUNCTION /* function might be : On, OFF */
constants
  POSITION /* set of positions that an SDV can move to them */
  Lane /* subset of position */
  init_position /* initial position of SDV */
  ON /* ALC is working*/
  OFF /* ALC is deactivated*/
axioms
  /* Type of POSITION */
  @typeof-POSITION: POSITION =  $\mathbb{N} \times \mathbb{Z}$ 
  /* Lane involves a set of positions */
  @typeof-Lane: Lane  $\subseteq$  POSITION
  /* the initial position of an SDV is located in the Lane */
  @typeof-init_position: init_position  $\in$  Lane
  /* System status */
  @def-ALC-mode: partition(ALC_FUNCTION, {ON}, {OFF})
end

```

E.1.2 Machine m0

```

machine m0 sees c0
/*
 * Abstract machine (M0) aims to capture the main functionality of a system as 'the ALC
 * system moves an SDV in the middle of its desired (target) lane
 */
variables
/* $\times \times \times$ */
 * M0 ----- variables
 $\times \times \times$ */
/* (CA1) The (physical) position of the vehicle */
SDV_POSITION_env
/* System status is ON or OFF */
ALC_Status

invariants

/* SC0: the SDV must always be located inside the target lane */
@safety: SDV_POSITION_env  $\in$  Lane
/* System status might be either ON or OFF */
@ALC.mode: ALC_Status  $\in$  ALC_FUNCTION

```

events**event** INITIALISATION**begin**

/* (F1) shows the current position of an SDV */

@init-position: SDV_POSITION_env := init_position

/*initialise ALC status is ON*/

@init-mode: ALC_Status := OFF

end

/*

* Activation_Function

*/

event ALC_ON**where**

@grd1: ALC_Status = OFF

then

@act1: ALC_Status := ON

end

/*

* Deactivation_Function

*/

event ALC_OFF**where**

@grd1: ALC_Status = ON

then

@act1: ALC_Status := OFF

@reset-position: SDV_POSITION_env := init_position

end

/*

* This event changes a current position of an SDV into a new position inside the target lane.

*/

event move**any**

/* SR0: the SDV must travel into a new position inside the target lane */

new_position

where

/* A new position inside the target lane*/

@grd1: new_position ∈ Lane

then

/* moved into a new position inside the target lane */

@act1: SDV_POSITION_env := new_position

end**end**

E.2 First Refinement

E.2.1 Context c1

context c1 **extends** c0

constants

/ steering range */*

STEERING_ANGLE

max_steering

min_steering

/ range of acceptable change to physical steering angle*/*

STEERING_ANGLE_CHANGE

max_steering_constraint

min_steering_constraint

/ A function assumes that an SDV actuate target steering angle to reach a target position (actuation task) */*

move

axioms

@typeof-max_steering: max_steering $\in \mathbb{Z}$

@typeof-min_steering: min_steering $\in \mathbb{Z}$

@min_max_steering: min_steering < max_steering

/ these just an example of steering range */*

@def-STEERING_ANGLE: STEERING_ANGLE = min_steering .. max_steering

@typeof-max_steering_constraint: max_steering_constraint $\in \mathbb{Z}$

@typeof-min_steering_constraint: min_steering_constraint $\in \mathbb{Z}$

@min_max_steering_change: min_steering_constraint < max_steering_constraint

@def-STEERING_ANGLE_CHANGE: STEERING_ANGLE_CHANGE = min_steering_constraint .. max_steering_constraint

*/**

** function to assume that SDV actuates a steering angle into a specific position where SDV moves*

** into multiple positions inside the lane to reach a specific position*

**/*

@def-MOVE: move = (λ long \mapsto lat \mapsto angle \cdot long \mapsto lat \in POSITION \wedge angle \in STEERING_ANGLE | $\{i \cdot i \in 1..3 \mid$ long + i \mapsto lat + i \times angle})

theorem @typeof-move: move \in POSITION \times STEERING_ANGLE $\rightarrow \mathbb{P}_1$ (POSITION)

end

E.2.2 Machine m1


```

machine m1 refines m0
sees c1
/*
 * This machine (M1) investigate how the steering angle of an SDV is modified to reach a new
   position inside the target lane.
 */
variables
/*×××××*
 * M0 ----- variables
×××××*/
SDV_POSITION_env
ALC_Status
/*×××××*
 * M1 ----- variables
×××××/
/* (CA2) The (actual) steering angle of the vehicle */
SDV_STEERING_ANGLE_env
invariants
/* Steering angle must be in a defined range of steering angle (abstract constant) */
@typeof-steering_angle: SDV_STEERING_ANGLE_env ∈ STEERING_ANGLE
events
event INITIALISATION extends INITIALISATION
begin
/* (F2) The current steering angle of an SDV that is randomly initialized */
@init-steering_angle: SDV_STEERING_ANGLE_env :∈ STEERING_ANGLE
end
/*
 * This event identifies a steering angle autonomously
 *
 */
event ALC_actuating
any
steering_angle_change
where
@grd1: ALC_Status = ON
/*steering angle change definition*/
@grd2: steering_angle_change ∈ STEERING_ANGLE_CHANGE
/* new steering angle in defined range*/
@grd3: steering_angle_change + SDV_STEERING_ANGLE_env ∈ STEERING_ANGLE
/* (SR1.1): new change will lead into the lane */
@grd4: move(SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env +
steering_angle_change)) ⊆ Lane
then
/* replace old steering */
@act1: SDV_STEERING_ANGLE_env := (SDV_STEERING_ANGLE_env +
steering_angle_change)
end

```

```

/*
 * This event identifies a steering angle manually
 */
event Manual_actuating
any
manual_steering_angle
where
@grd1: manual_steering_angle ∈ STEERING_ANGLE
/*(SR1.2): driver steering leads to lane */
@grd2: move(SDV_POSITION_env ↦ manual_steering_angle) ⊆ Lane
then
/*replace old steering*/
@act1: SDV_STEERING_ANGLE_env := manual_steering_angle
end

/*
 * A extended event moves an SDV in its target lane based the (new) steering angle of ALC
   or human driver.
 * where a movement must lead into the lane
 */
event move extends move
where
/* new (target) position must be within set of position inside the lane*/
@grd2: new_position ∈ move(SDV_POSITION_env ↦ SDV_STEERING_ANGLE_env)
end

event ALC_ON extends ALC_ON
end
event ALC_OFF extends ALC_OFF
end

end

```

E.3 Second Refinement

E.3.1 Context c2

```

context c2 extends c1
sets
/* set of images obtained from the driving environment (target lane) */
IMAGE
/* partition shows the different stages of the ALC system, such as Perception, planning and
   etc*/
STAGE
/* set of left lane lines */
LeftLane

```

```

/* set of right lane lines */
RightLane
constants
/* An abstract constant uses to show how the autonomous controller gets the images of
   SDV's positions in the driving environment (sensor of perception task) */
camera
/* An abstract constant shows the expected results of a received image */
Seen_image
/* An abstract constant shows how ALC accomplishes the OEDR task based on the sensing
   images (perception task)
   * For example, the desired path (set of positions) would identify from a received image
   */
OEDR.task
/* An abstract constant shows an accuracy of identifying left/right lane lines in a received
   image
   * For instance, a score from 0 to 100 indicates the accuracy of the identification process of left
   and right lane lines in the incoming image.
   * 0 is very low accuracy, while 100 is a high accuracy
   */
Confidence.score
/*
   * An abstract constant shows how AC computes planning task, e.g., ALC selects one (target)
   position from a set of position (desired path)
   */
target_position
/*
   * An abstract constant shows how AC computes planning task, e.g., ALC identifies a change
   of steering angle based on the target position and a current steering angle
   */
target_steering_angle

/*
   * These constants are used to distinguish between different possible stages of the ALC
   system
   */
Perception /* perception's stage*/
Decision /* decision's stage*/
Control /* control's stage */
Intervention /* intervention's stage indicates ALC could ask the driver to take control of
   SDV*/
AutonomousDriving /* Autonomous driving's stage, indicates ALC actuate the steering
   angle autonomously to reach a (identified) target position */

axioms
/* possible stages of ALC system */
@def-STAGE: partition(STAGE, {Perception}, {Decision}, {Control}, {
   AutonomousDriving}, {Intervention})

```

```

/* range of Confidence score from 0 to 100 */
@score: Confidence_score = 0 .. 100
/*
 * Perception task
 * (three functions)
 */
// 1) image obtains from camera based on SDV's position
@typeof-camera: camera ∈ POSITION → IMAGE
/* 2) function shows the expected results of a received image. */
@typeof-image-seen: Seen_image ∈ (IMAGE × LeftLane × RightLane) →
    Confidence_score
/* 3) function recognises the desired path (set of position ) based on the left/right lane lined,
    image and confidence score */
@typeof-path-recognition: OEDR_task ∈ (IMAGE × LeftLane × RightLane ×
    Confidence_score) → IP(POSITION)
/*
 * End of Perception task
 */
/*
 * planning task
 * (two functions)
 */
/* 1) function to specify a target position based on a set of positions (desired path) */
@typeof-compute_target_position: target_position ∈ IP(POSITION) → POSITION
/* 2) function to compute the change of steering angle to reach a specific position */
@typeof-compute_target_steering_angle: target_steering_angle ∈ POSITION ×
    STEERING_ANGLE → STEERING_ANGLE_CHANGE
/*
 * End of the planning task
 */
/*
 * This is the aim of the ALC system, which means that there is always one steering angle
 * can be used to move an SDV into a position inside the target lane
 */
@aim: ∀ SDV_position · SDV_position ∈ POSITION ⇒
    (∃ SDV_steering · SDV_steering ∈ STEERING_ANGLE ⇒
        move (SDV_position ↦ SDV_steering) ⊆ Lane
    )

```

end

E.3.2 Machine m2

machine m2 refines m1

sees c2

/*

```

* This machine (M2) studies how the ALC system accomplishes its autonomous operations,
  such as perception, decision–making and control,
* to identify the steering angle that autonomously moves an SDV into a position inside the
  target lane.
*/

```

variables

```

/*××××*
* M0 – variables
××××*/
SDV_POSITION_env
ALC_Status
/*××××*
* M1 – variables
××××/
SDV_STEERING_ANGLE_env
/×××*
* M2 – variables
××××/
/* (F3) The image would obtain from the camera, which can be used as the input to the
  perception stage */
IMAGE_env
/* indicates left lane lines in a received image */
leftLanePoints
/* indicates right lane lines in a received image */
rightLanePoints
/* (CA4) set of positions seen as the desired path */
desirePath
/* (CA5) score to show the accuracy of the detection process for seeing images and
  identifying the desired path */
confidenceScore
/* (CA6) a new (target) position that identifies form a set of positions (desired path) */
targetPosition
/* (CA7) a target steering angle that would be used to autonomously move an SDV from the
  current position into the target position */
targetSteeringAngle
/* (F5) a required change of steering angle from current steering to a new (target) steering */
steeringAngleChange
/* shows the different stages of the ALC system, such as Perception, planning and etc */
stage
/* A boolean flag indicates that an SDV is ready to move*/
signal

```

invariants

```

@typeof–stage: stage ∈ STAGE
@typeof–desire_path: desirePath ⊆ POSITION
@typeof–perceived_confident: confidenceScore ∈ Confidence_score

```

```

@typeof_leftLanePoints: leftLanePoints ∈ LeftLane
@typeof_rightLanePoints: rightLanePoints ∈ RightLane
@typeof_target_position: targetPosition ∈ POSITION
@typeof_target_steering_angle: targetSteeringAngle ∈ STEERING_ANGLE
@typeof_current_steering_angle_change: steeringAngleChange ∈
  STEERING_ANGLE_CHANGE
@typeof_signal_movement: signal ∈ BOOL

/* The gluing inv1 invariant states that the ALC system might be at any stage when an SDV
  system aims to identify a new steering angle (manual or autonomous)*/
@gluing_inv1: ALC_Status = ON ⇒ stage ∈ STAGE

/*(SR2.1): The camera always shows the image of the current position*/
@consistency: IMAGE_env = camera(SDV_POSITION_env)

/* (SR2.2): based on the detected left/right lane lines and confidence scores, the perception
  component identifies the desired path*/
@perceivedImage: stage = Perception ⇒ IMAGE_env ∈ ran(camera) ∧
  (IMAGE_env ↦ leftLanePoints ↦ rightLanePoints ↦ confidenceScore) ∈ dom(
    OEDR_task)

/*
  * before changing the stage of ACL from perception to Decision, the desired path must be
  identified
  */
@perceptionTask: stage = Decision ⇒ desirePath ∈ ran(OEDR_task)

/* (SR2.5) and (SR2.6): emphasise that the decision-making component must compute a
  target position and a required change of steering angle according to the set of positions (
  desired path) and a current steering angle of an SDV. Therefore, invariant targetPosition
  ensures that a target (new) position will obtain from the desired path, i.e., targetPosition
  := compute_target_position(desirePath). In addition, invariant changeOfSA ensures that
  a required change of steering angle is computed according to the target position and the
  current steering angle of an SDV, i.e., steeringAngleChange := target_steering_angle(
  targetPosition → SDV_STEERING_ANGLE_env). Note: confidenceScore ≥ 80 denotes
  that a system identifies a correct path (assumption) */
@targetPosition: stage = Decision ∧ confidenceScore ≥ 80 ⇒
  desirePath ∈ dom(target_position)
@changeSteer: stage = Control ∧ confidenceScore ≥ 80 ⇒
  (targetPosition ↦ SDV_STEERING_ANGLE_env) ∈ dom(target_steering_angle)

/* (SR2.7): indicates that a target steering angle must not actuate the physical steering angle
  of an SDV too quickly; therefore, invariant change_ofSteeringAngle ensures that the
  decision-making component identifies a required change of the steering angle in a
  defined range that prevents actuating the steering of an SDV too quickly, i.e.,
  steeringAngleChange ∈ min_steering_constraint .. max_steering_constraint.*/

```

```
@newSteer: stage = Control  $\wedge$  confidenceScore  $\geq$  80  $\Rightarrow$  steeringAngleChange  $\in$  ran(
  target_steering_angle)
```

```
/*This invariant ensures that a control module of the ALC system always provides a new (
  target) steering angle in a defined range of power steering system in SDV*/
```

```
@Control-consistency: stage = Control  $\Rightarrow$  targetSteeringAngle  $\in$  STEERING_ANGLE
```

```
/* This invariant ensures that the ALC system never exceeds the defined range of steering
  angle when the new (target) steering angle is used to move an SDV into a new (target)
  position inside the target lane*/
```

```
@Environment-consistency: stage = AutonomousDriving
 $\Rightarrow$  move(SDV_POSITION_env  $\mapsto$  targetSteeringAngle)  $\subseteq$  Lane
```

events

event INITIALISATION **extends** INITIALISATION

begin

```
/* Camera is ready to provide an image for the initial position */
```

```
@init-image: IMAGE_env := camera (init_position)
```

```
/* the stage of ALC system starts with the perception module */
```

```
@init-stage: stage := Perception
```

```
/* No desire path (hence empty)*/
```

```
@init-desire_path: desirePath :=  $\emptyset$ 
```

```
/* No confident score (hence zero) */
```

```
@init-confident_score: confidenceScore := 0
```

```
/* No initial target position (hence random) */
```

```
@init-target_position: targetPosition  $\in$  POSITION
```

```
/* No initial target steering angle (hence random) */
```

```
@init-target_steering_angle: targetSteeringAngle  $\in$  STEERING_ANGLE
```

```
/* NO initial amount change in current steering angle (hence random) */
```

```
@init-change_steering_angle: steeringAngleChange  $\in$  STEERING_ANGLE_CHANGE
```

```
/* No initial left lane points (hence random)*/
```

```
@typeof_leftLanePoints: leftLanePoints  $\in$  LeftLane
```

```
/* No initial right lane points (hence random)*/
```

```
@typeof_rightLanePoints: rightLanePoints  $\in$  RightLane
```

```
/* signal of movement*/
```

```
@typeof_signal: signal := FALSE
```

end

```
/*
```

```
* This event performs the perception task. It involves three main subtasks:
```

```
* 1) obtain an image from the camera based on SDV's position
```

```
* 2) identify confidence score based on the detection results of a received image as the left/
  right lane lines
```

```
* 3) recognise the desired path (set of positions) based on the left/right lane lines, image and
  confidence score
```

```
*/
```

```

event perception
  any
  leftLane /* given left lane */
  rightLane /* given right lane */
  when
  @grd1: leftLane  $\subseteq$  LeftLane
  @grd2: rightLane  $\subseteq$  RightLane
  @grd3: ALC_Status = ON
  @grd4: stage = Perception
  then
  /* obtain an image from camera based on SDV's position */
  @act1: IMAGE_env := camera (SDV_POSITION_env)
  @act2: leftLanePoints  $\in$  LeftLane
  @act3: rightLanePoints  $\in$  RightLane
  /* SR2.3: identify confidence score based on the detection results of a received image as the
     left/right lane lines */
  @act4: confidenceScore := Seen_image (IMAGE_env  $\mapsto$  leftLanePoints  $\mapsto$ 
    rightLanePoints)
  /* recognise the desired path (set of position ) based on the left/right lane lines, image and
     confidence score */
  @act5: desirePath := OEDR_task (IMAGE_env  $\mapsto$  leftLanePoints  $\mapsto$  rightLanePoints  $\mapsto$ 
    confidenceScore)
  /* change a stage of ALC to be in Decision */
  @act6: stage := Decision
end

/*
* This event checks whether the desired path has been identified using a low or high
  confidence score.
* Note: if a confidence score is less than 80, we assume that a low confidence score
*/
event lowConfidenceScore.interven
  where
  @grd1: ALC_Status = ON
  @grd2: stage = Decision
  /* (F4): the ALC system may determine the desired path with a low confidence score */
  @grd3: confidenceScore < 80
  then
  /* (SR2.4): if the perception module identifies the desired path with a low confidence score,
     the ALC system will issue a request to intervene. Therefore, the stage of the ALC system
     changed to be in Intervention */
  @act1: stage := Intervention
end

```



```
/* This event shows how an ALC system performs the planning task in order to identify a
   target position and a change of steering angle when the stage of the ALC is in Decision,
   and the confidence score is greater than or equal to 80. Note: if a confidence score is more
   or equal to 80, we assume that a high confidence score*/
```

```
event decision
```

```
when
```

```
@grd1: ALC_Status = ON
```

```
@grd2: stage = Decision
```

```
/* denotes a high confidence score */
```

```
@grd3: confidenceScore ≥ 80
```

```
then
```

```
/* new (target) position obtained from a set of positions (desired path) */
```

```
@act1: targetPosition := target_position(desirePath)
```

```
/* a change of steering angle computed based on the identified (target) position and current
   steering angle of SDV */
```

```
@act2: steeringAngleChange := target_steering_angle(targetPosition ↦
   SDV_STEERING_ANGLE_env)
```

```
/* the stage of ALC changed to be in 'Control' */
```

```
@act3: stage := Control
```

```
end
```

```
/* This event shows how ALC performs the actuation task for moving an SDV into a target
   position when the stage of the ALC is in Control.*/
```

```
event control
```

```
when
```

```
@grd1: ALC_Status = ON
```

```
@grd2: stage = Control
```

```
/* a required change of current steering must be in the defined range of power steering
   system in SDV */
```

```
@grd3: SDV_STEERING_ANGLE_env + steeringAngleChange ∈ STEERING_ANGLE
```

```
/* based on a change of steering angle, the movement of an SDV must lead to a position
   inside the target lane */
```

```
@grd4: move(SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env +
   steeringAngleChange) ) ⊆ Lane
```

```
then
```

```
/* target steering angle of ALC is computed */
```

```
@act1: targetSteeringAngle := steeringAngleChange + SDV_STEERING_ANGLE_env
```

```
/* the stage of ALC changed to be in 'AutonomousDriving'*/
```

```
@act2: stage := AutonomousDriving
```

```
end
```

```
/*
```

```
* This event covers how ALC may change a target steering angle to satisfy the defined range
   (minimum steering)
```

```
* of the power steering system in the SDV when the stage of the ALC is in Control.
```

```
*/
```

```
event correct_exceeding_min_steering
```

```

when
@grd1: ALC_Status = ON
@grd2: stage = Control
/*
* ALC presents a change of steering angle that violates the minimum steering of the power
  steering system in SDV.
*/
@grd3: SDV_STEERING_ANGLE_env + steeringAngleChange < min_steering
/*
* the minimum steering of the power steering system moves SDV into a new position
  inside the target lane
*/
@grd4: move(SDV_POSITION_env ↦ min_steering) ⊆ Lane
then
/* if the control module of ALC exceeds the steering angle range of an SDV, the ALC
  system will issue a request to intervene; therefore, the stage of ALC changed to be in '
  Intervention'*/
@act1: stage := Intervention
/* If a target steering angle exceeds the minimum steering angle, then the control module
  will modify the target steering angle to be the minimum steering angle*/
@act2: targetSteeringAngle := min_steering
end

/* This event covers how ALC may change a target steering angle to satisfy the defined
  range (maximum steering) of the power steering system in the SDV when the stage of the
  autonomous controller is in Control*/
event correct_exceeding_max_steering
when
@grd1: ALC_Status = ON
@grd2: stage = Control
/*ALC presents a change of steering angle that violates the maximum steering of the power
  steering system in SDV*/
@grd3: SDV_STEERING_ANGLE_env + steeringAngleChange > max_steering
/* the maximum steering of the power steering system moves SDV into a new position
  inside the target lane*/
@grd4: move(SDV_POSITION_env ↦ max_steering) ⊆ Lane
then
/* if the control module of ALC exceeds the steering angle range of an SDV, the ALC
  system will issue a request to intervene; therefore, the stage of ALC changed to be in '
  Intervention'*/
@act1: stage := Intervention
/* If a target steering angle exceeds the maximum steering angle, then the control module
  will modify the target steering angle to be the maximum steering angle*/
@act2: targetSteeringAngle := max_steering
end
/*

```

```

* This event detects a wrong movement outside the target lane when the ALC system may
  change a target steering to a minimum or maximum steering
*/
event correct_out_of_lane
where
@grd1 : ALC_Status = ON
@grd2 : stage = Control
/* when a change of steering angle in a defined range of steering angle but moving SDV
  outside the lane*/
@grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange ∈ STEERING_ANGLE ⇒
  move(SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env + steeringAngleChange)) ⊈
  Lane
then
/* The stage of ALC changed to be 'Intervention'*/
@act1 : stage := Intervention
/*
* Keep the current steering as the target steering
* because the previous steering is safer than this new steering << assumption >>
*/
@act2 : targetSteeringAngle := SDV_STEERING_ANGLE_env
end

/*This is a refined event from an abstract event ALC_actuating for allowing a target steering
  angle (autonomous steering angle) to be used for moving an SDV in its target lane when
  the stage of the autonomous controller is in AutonomousDriving*/
event ALC_actuating refines ALC_actuating
where
@grd1 : ALC_Status = ON
@grd2 : steeringAngleChange ∈ STEERING_ANGLE_CHANGE
@grd3 : steeringAngleChange + SDV_STEERING_ANGLE_env ∈ STEERING_ANGLE
/* (SR): the ALC system must actuate a steering angle to reach a target position that keeps
  an SDV inside the target lane */
@grd4 : move(SDV_POSITION_env ↦ (SDV_STEERING_ANGLE_env +
  steeringAngleChange)) ⊆ Lane
@grd5 : stage = AutonomousDriving
@grd6 : steeringAngleChange + SDV_STEERING_ANGLE_env = targetSteeringAngle
@grd7 : signal = FALSE
then
/* Change the current steering based on the steering change of ALC system */
@act1 : SDV_STEERING_ANGLE_env := (SDV_STEERING_ANGLE_env +
  steeringAngleChange)
/* ready to move */
@act2 : signal := TRUE
with
/* a change of steering angle is replaced by a change identified by the ALC system */
@steering_angle_change : steering_angle_change = steeringAngleChange
end

```

```

/* This is a refined event from an abstract event Manual_actuating for covering how a
   human driver intervenes to take over control of an SDV when the stage of the ALC is in
   Intervention */
event Manual_actuating extends Manual_actuating
where
  @grd3: stage = Intervention
  @grd4: signal = FALSE
then
  /* ready to move */
  @act2: signal := TRUE
end

/*
 * This is an extended event for using either a manual or autonomous steering angle to move
 * The SDV inside its target lane
 */
event move extends move
where
  @grd3: signal = TRUE
  @grd4: stage ∈ {AutonomousDriving , Intervention}
then
  @act2: stage := Perception
  @act3: IMAGE_env := camera (new_position)
  @act4: signal := FALSE
end

event ALC_ON extends ALC_ON
end

/*
 * This is an extended event to reset a stage of an autonomous controller to be at
 * the perception stage when a system is switched off
 */
event ALC_OFF extends ALC_OFF
then
  /* ALC_STAGE reset into initialisation value*/
  @restALC: stage := Perception
  @rest-image: IMAGE_env := camera (init_position)
  @rest_signal: signal := FALSE
end
end

```

E.4 Third Refinement

E.4.1 Machine m3

machine m3 **refines** m2

sees c2

/*

* This machine (M3) investigates how the awareness level of a driver involves in the autonomous operations of a system in order to

* ensure that a human driver is a fallback option when the ALC system may issue a request to intervene.

*/

variables

/*XXXXXXXXXX

* M0 – variables

XXXXXXXXXX*/

SDV_POSITION_env

ALC_Status

/*XXXXXXXXXX

* M1 – variables

XXXXXXXXXX*/

SDV_STEERING_ANGLE_env

/XXXXXXXXXX*

* M2 – variables

XXXXXXXXXX*/

IMAGE_env

leftLanePoints

rightLanePoints

desirePath

confidenceScore

targetPosition

targetSteeringAngle

steeringAngleChange

stage

signal

/*XXXXXXXXXX

* M3 – variables,

XXXXXXXXXX*/

/* (CA8) shows the awareness level of a driver */

awarenessLevel

/* (F6, F7) indicates that the ALC system issues an intervention request */

warningMessage

/* (CA9) indicates whether a driver puts their hands on the steering wheel inside the SDV or not */

handsOnSteeringWheel

/* (CA10) indicates whether a driver provides a sensitive monitoring feature or not */

sensitiveMonitoringFeature

invariants

@typeof-*awarenessLevel*: *awarenessLevel* ∈ BOOL

@typeof-*warning*: *warningMessage* ∈ BOOL

@typeof-*handsOnSteeringWheel*: *handsOnSteeringWheel* ∈ BOOL

@typeof-*sensitiveMonitoringFeature*: *sensitiveMonitoringFeature* ∈ BOOL

/* (SR3.2): This invariant ensures that the awareness level of a driver computes based on the detection of the hands-On steering wheel and the sensitive monitoring feature */

@*awarenesslevel*: *awarenessLevel* = TRUE ⇒ (*handsOnSteeringWheel* = TRUE ∧ *sensitiveMonitoringFeature* = TRUE)

/* (SR3.1): This invariant ensures that a driver will be aware of driving tasks in any stages of ALC, such as perception, planning and etc */

@*driver_aware*: *awarenessLevel* = TRUE ⇒ *stage* ∈ {*Perception* , *Decision* , *Control* , *Intervention* , *AutonomousDriving*}

/*(SR3.5)This invariant means that the ALC system will send an intervention request if the autonomous controller's stage in Intervention*/

@*send_req*: *warningMessage* = TRUE ⇒ *stage* = *Intervention*

/* (SR3.5) Because we have two intervention scenarios in the previous machine, the intervention request will send as follows

* 1) when the ALC system used a low confidence score to identify the desired path, OR

* 2) when the ALC system attempts to exceed the define steering range of the power steering system

* 3) when the ALC system moves to position outside the lane

*/

@*interv_cases*: *warningMessage* = TRUE

⇒

confidenceScore < 80 ∨

SDV_STEERING_ANGLE_env + *steeringAngleChange* ∉ *STEERING_ANGLE* ∨

move(*SDV_POSITION_env* ↦ (*SDV_STEERING_ANGLE_env* + *steeringAngleChange*))

∉ *Lane*

events

event *INITIALISATION* extends *INITIALISATION*

begin

/*initial awareness level is FALSE, which means a driver is unaware of driving task*/

@*init-*awareness_level**: *awarenessLevel* := FALSE

/*initial warning message is FALSE, means the ALC system does not issue a request to intervene*/

@*init-*warning**: *warningMessage* := FALSE

/*initial detection of hands-on steering wheel is FALSE, means driver does not put hands on steering wheel*/

@*init-*handsOnSteeringWheel**: *handsOnSteeringWheel* := FALSE

```
/*initial detection of sensitive monitoring feature is FALSE, means driver does not provide
sensitive monitoring feature*/
```

```
@init-sensitiveMonitoringFeature: sensitiveMonitoringFeature := FALSE
```

```
end
```

```
/* This event abstractly captures a change of the hands-on steering wheel when a driver
puts their hands on the steering wheel inside the SDV.*/
```

```
event DMS_hands_on_wheel
```

```
where
```

```
@grd1: ALC_Status = ON
```

```
@grd2: awarenessLevel = FALSE
```

```
@grd3: handsOnSteeringWheel = FALSE
```

```
then
```

```
@act1: handsOnSteeringWheel := TRUE
```

```
/* compute based on both human monitoring features */
```

```
@act3: awarenessLevel := bool (sensitiveMonitoringFeature = TRUE)
```

```
end
```

```
/*This event abstractly captures a change of the sensitive monitoring feature when a driver
provides their sensitive monitoring features*/
```

```
event DMS_detect_sensitiveMonitoringFeature
```

```
where
```

```
@grd1: ALC_Status = ON
```

```
@grd2: awarenessLevel = FALSE
```

```
@grd3: sensitiveMonitoringFeature = FALSE
```

```
then
```

```
@act1: sensitiveMonitoringFeature := TRUE
```

```
/* compute based on both human monitoring features */
```

```
@act3: awarenessLevel := bool (handsOnSteeringWheel = TRUE)
```

```
end
```

```
/*This is a refined event from the ALC_OFF event for covering a possible change of the
hands-on steering wheel. Note; the human monitoring features, such as the hands-on
steering wheel, are considered as one of the preconditions for activating the ALC system
; therefore, the system immediately switches off and starts again from the initialisation
states when a driver removes their hands from the steering wheel*/
```

```
event DMS_hands_off_wheel
```

```
extends ALC_OFF
```

```
where
```

```
@grd3: awarenessLevel = TRUE
```

```
@grd4: handsOnSteeringWheel = TRUE
```

```
then
```

```
@act3: awarenessLevel := FALSE
```

```
@act4: warningMessage := FALSE
```

```
@act5: handsOnSteeringWheel := FALSE
```

```
@act6: sensitiveMonitoringFeature := FALSE
```

```
end
```

```

event DMS_lost_sensitiveMonitoringFeature
extends ALC_OFF
where
  @grd3: awarenessLevel = TRUE
  @grd4: sensitiveMonitoringFeature = TRUE
then
  @act3: awarenessLevel := FALSE
  @act4: warningMessage := FALSE
  @act5: sensitiveMonitoringFeature := FALSE
  @act6: handsOnSteeringWheel := FALSE

end

event ALC_ON extends ALC_ON
then
  /* reset variables of this machine */
  @reset-awariness_level: awarenessLevel := FALSE
  @reset-warining: warningMessage := FALSE
  @reset-handsOnSteeringWheel: handsOnSteeringWheel := FALSE
  @reset-sensitiveMonitoringFeature: sensitiveMonitoringFeature := FALSE
end

event ALC_OFF extends ALC_OFF
then
  /* reset variables of this machine */
  @reset-awariness_level: awarenessLevel := FALSE
  @reset-warining: warningMessage := FALSE
  @reset-handsOnSteeringWheel: handsOnSteeringWheel := FALSE
  @reset-sensitiveMonitoringFeature: sensitiveMonitoringFeature := FALSE
end

/*The awareness level of a driver must be TRUE to enable the perception event*/
event perception extends perception
where
  @grd5: awarenessLevel = TRUE
end

/*The awareness level of a driver must be TRUE to enable the low_cofidance_score event
Warning message changes to TRUE, which denotes the ALC system issue a request to
intervene*/
event lowConfidenceScore_interven extends lowConfidenceScore_interven
where
  @grd4: awarenessLevel = TRUE
  @grd5: warningMessage = FALSE
then
  @act2: warningMessage := TRUE
end

```



```

/*Awareness level of a driver must be TRUE to enable the decision event*/
event decision extends decision
where
  @grd4: awarenessLevel = TRUE
end

/*The awareness level of a driver must be TRUE to enable the control event Warning
message must be FALSE to ensure that a system can keep working with no need for
driver help*/
event control extends control
where
  @grd5: awarenessLevel = TRUE
  @grd6: warningMessage = FALSE
end

/*Awareness level of a driver must be TRUE to enable the correct_steering_max event
Warning message must be TRUE which denotes the ALC system issue a request to
intervene*/
event correct_exceeding_max_steering
extends correct_exceeding_max_steering
where
  @grd5: awarenessLevel = TRUE
  @grd6: warningMessage = FALSE
then
  @act3: warningMessage := TRUE
end

/*Awareness level of a driver must be TRUE to enable the correct_steering_min event
Warning message must be TRUE which denotes the ALC system issue a request to
intervene*/
event correct_exceeding_min_steering
extends correct_exceeding_min_steering
where
  @grd5: awarenessLevel = TRUE
  @grd6: warningMessage = FALSE
then
  @act3: warningMessage := TRUE
end

/*Awareness level of a driver must be TRUE to enable the correct_steering_min event
Warning message must be TRUE which denotes the ALC system issue a request to
intervene*/
event correct_out_of_lane extends correct_out_of_lane
where
  @grd5: awarenessLevel = TRUE
  @grd6: warningMessage = FALSE

```

```

then
  /* the stage of ALC changed to be in 'Intervention'*/
  @act3: warningMessage := TRUE
end

/*Awareness level of a driver must be TRUE to enable the Auto_driving event Warning
message must be FALSE to ensure that a system can keep working with no need for
driver help*/
event ALC_actuating extends ALC_actuating
where /* driver is aware*/
  @grd8: awarenessLevel = TRUE
  /* no request to intervene*/
  @grd9: warningMessage = FALSE
end

/*Awareness level of a driver must be TRUE to enable the Manual_driving event Warning
message must be TRUE to enable this event. In any cases of intervention, a driver will be
receptive and change Warning message to be FALSE */
event Manual_actuating extends Manual_actuating
where
  @grd5: awarenessLevel = TRUE
  @grd6: warningMessage = TRUE
then
  @act3: warningMessage := FALSE
end

/*This event assumes that a driver would provide a steering angle manually when the ALC
system is in Intervention*/
event manual_movement refines move
any
  new_position
where
  @grd1: new_position ∈ Lane
  @grd2: new_position ∈ move(SDV_POSITION_env ↦ SDV_STEERING_ANGLE_env) //
    inherited element
  @grd4: signal = TRUE
  @grd5: stage = Intervention
  /* driver is already react*/
  @grd6: awarenessLevel = TRUE
  @grd7: warningMessage = FALSE

then
  @act1: SDV_POSITION_env := new_position
  @act3: stage := Perception
  @act4: IMAGE_env := camera(new_position)
  @act5: signal := FALSE
end

```

```
/*This event assumes that an SDV applies a steering angle autonomously.*/  
event auto_movement refines move  
any  
  new_position  
where  
  @grd1: new_position ∈ Lane  
  @grd2: new_position ∈ move(SDV_POSITION_env ↦ SDV_STEERING_ANGLE_env) //  
    inherited element  
  @grd4: signal = TRUE  
  @grd5: stage = AutonomousDriving  
  @grd6: awarenessLevel = TRUE  
  @grd7: warningMessage = FALSE  
then  
  @act1: SDV_POSITION_env := new_position  
  @act3: stage := Perception  
  @act4: IMAGE_env := camera(new_position)  
  @act5: signal := FALSE  
end  
  
end
```


Appendix F

Modelling LKA, DMS and ACC Functions in the ALC System

In this appendix, we present the complete Event-B version of the application of the modelling patterns for the Lane Keeping Assist (LKA), Driver Monitoring System (DMS) and Adaptive Cruise Control (ACC) functions in the Automated Lane Centring (ALC) system, as outlined in Chapter 8. The majority of the proof obligations in modelling these functions were verified either automatically using Rodin provers or with the assistance of additional external prover plug-ins, such as SMT solvers (as illustrated in Figure F.1).

Element Name	Total	Auto	Man.	Rev.	Und.
def_ALC_cas...	372	359	13	0	0
c0	0	0	0	0	0
c0_anim	0	0	0	0	0
c1	1	0	1	0	0
c1_anim	0	0	0	0	0
c2	1	1	0	0	0
c2_anim	0	0	0	0	0
m0	3	3	0	0	0
m0_anim	0	0	0	0	0
m1	17	17	0	0	0
m1_anim	0	0	0	0	0
m2	220	208	12	0	0
m2_anim	0	0	0	0	0
m3	61	61	0	0	0
m3_anim	0	0	0	0	0

FIGURE F.1: Modelling LKA, DMS and ACC functions, prover statistics

The complete Event-B models ¹ are presented in the following sections.

¹The Event-B models for developing design patterns of SDV systems are available as a Rodin archive at https://drive.google.com/drive/folders/1D7rVAJKCEh_-rVr9hKZZYdFDCxAP8r3F.

F.1 Abstract Level

F.1.1 Context c0

```

context c0
sets
  ALC_FUNCTION
constants
  POSITION
  Lane
  init_position
  ON
  OFF
axioms
  @typeof-POSITION: POSITION =  $\mathbb{N} \times \mathbb{Z}$ 
  @typeof-Lane: Lane  $\subseteq$  POSITION
  @typeof-init_position: init_position  $\in$  Lane
  @def-ALC-mode: partition(ALC_FUNCTION, {ON}, {OFF})
end

```

F.1.2 Machine m0

```

machine m0 sees c0
variables
  SDV_POSITION_env
  ALC_Status
invariants
  @safety: SDV_POSITION_env  $\in$  Lane
  @ALC_mode: ALC_Status  $\in$  ALC_FUNCTION
events
  event INITIALISATION
    begin
    @init-position: SDV_POSITION_env := init_position
    @init-mode: ALC_Status := OFF
    end
  event ALC_ON
    where
    @grd1: ALC_Status = OFF
    then
    @act1: ALC_Status := ON
    end
  event ALC_OFF
    where
    @grd1: ALC_Status = ON
    then
    @act1: ALC_Status := OFF

```

```

@reset—position: SDV_POSITION_env := init_position
end
event move
  any
  new_position
  where
  @grd1: new_position ∈ Lane
  then
  @act1: SDV_POSITION_env := new_position
end
end

```

F.2 First Refinement

F.2.1 Context c1

```

context c1 extends c0
constants
  STEERING_ANGLE
  max_steering
  min_steering

  STEERING_ANGLE_CHANGE
  max_steering_constraint
  min_steering_constraint

  /* Speed range*/
  SPEED
  max_speed
  min_speed
  /* A function assumes that an SDV actuate target steering angle to reach a target position (
    actuation task) */
  move

axioms
  @typeof—max_steering: max_steering ∈ ℤ
  @typeof—min_steering: min_steering ∈ ℤ
  @min_max_steering: min_steering < max_steering
  @def—STEERING_ANGLE: STEERING_ANGLE = min_steering .. max_steering

  @typeof—max_speed: max_speed ∈ ℕ1
  @typeof—min_speed: min_speed ∈ ℕ1
  @min_max_speed: min_speed < max_speed
  /* these just an example of steering range */
  @def—speed: SPEED = min_speed .. max_speed

```

```

@typeof-max_steering_constraint: max_steering_constraint ∈ ℤ
@typeof-min_steering_constraint: min_steering_constraint ∈ ℤ
@min_max_steering_change: min_steering_constraint < max_steering_constraint
@def-STEERING_ANGLE_CHANGE: STEERING_ANGLE_CHANGE = min_steering_constraint
    .. max_steering_constraint

/*
* This function assumes that SDV actuates a specific speed and steering angle in order to
  reach a specific
* position where SDV moves into multiple positions.
*/
@def-MOVE: move =
(
  λ long ↦ lat ↦ speed ↦ angle · long ↦ lat ∈
  POSITION ∧
  speed ∈ SPEED ∧
  angle ∈ STEERING_ANGLE |
  {i · i ∈ 1..3 | long + i ↦ lat + i × speed × angle}
)
theorem @typeof-move: move ∈ POSITION × SPEED × STEERING_ANGLE → ℙ1(
  POSITION)
end

```

F.2.2 Machine m1

```

machine m1 refines m0
sees c1
variables
  SDV_POSITION_env
  ALC_Status
  SDV_STEERING_ANGLE_env
  /* (CA2) (CA3) The (actual) speed of the SDV */
  SDV_SPEED_env
  /* target speed*/
  ACC_target_speed
invariants
  @typeof-steering_angle: SDV_STEERING_ANGLE_env ∈ STEERING_ANGLE
  /* Speed must be in a defined range of speed (abstract constant) */
  @typeof-speed: SDV_SPEED_env ∈ SPEED
  /* ACC target speed*/
  @typeof-targetSpeed: ACC_target_speed ∈ SPEED

events
event INITIALISATION extends INITIALISATION
begin
  /* (F2) The current speed and steering angle of an SDV that is randomly initialised */
  @init-steering_angle: SDV_STEERING_ANGLE_env :∈ STEERING_ANGLE

```



```

@init—speed: SDV_SPEED_env :∈ SPEED
/* random initialised*/
@init—targetSpeed: ACC.target_speed :∈ SPEED
end
/*
* This event identifies speed and steering angle autonomously
*
*/
event ALC_actuating
any
/* possible change of a current steering angle by ALC */
steering_angle_change
/* possible change of a current speed by ALC*/
sp
where
/* System status is ON*/
@grd1: ALC_Status = ON
/*steering angle change definition */
@grd2: steering_angle_change ∈ STEERING_ANGLE_CHANGE
/* new steering angle of ALC computed according to the current steering and a steering
change that ALC might require */
@grd3: steering_angle_change + SDV_STEERING_ANGLE_env ∈ STEERING_ANGLE
/* new speed definition */
@grd4: sp ∈ min_speed .. ACC.target_speed ∧ sp ≤ max_speed
/* (SR1): new change of steering angle will lead into the lane */
@grd5: move(SDV_POSITION_env ↦ sp ↦ (SDV_STEERING_ANGLE_env +
steering_angle_change) ) ⊆ Lane
then
/* Change a SDV's steering based on the current steering and the required steering change
of ALC */
@act1: SDV_STEERING_ANGLE_env := (SDV_STEERING_ANGLE_env +
steering_angle_change)
/* Change a SDV's speed by ALC */
@act2: SDV_SPEED_env := sp
end
/*
* This event identifies speed and steering angle manually
*/
event Manual_actuating
any
/* possible change of a current steering angle by human driver */
manual_steering_angle
manual_speed
where
@grd1: manual_steering_angle ∈ STEERING_ANGLE
@grd2: manual_speed ∈ SPEED
@grd3: move(SDV_POSITION_env ↦ manual_speed ↦ manual_steering_angle) ⊆ Lane

```

```

then
  /* new steering depends on how a driver specified steering angle */
  @act1 : SDV_STEERING_ANGLE_env := manual_steering_angle
  @act2 : SDV_SPEED_env := manual_speed
end

/*
 * A refined event moves an SDV in its target lane based the (new) steering angle and speed
   of ALC
 * where a movement must lead into the lane
 */
event auto_move refines move
any
  new_position
  sp
  steer
where
  @grd0 : ALC_Status = ON
  @grd1 : new_position ∈ Lane
  @grd2 : sp ∈ SPEED
  @grd3 : steer ∈ STEERING_ANGLE
  /* new (target) position must be within set of position inside the lane */
  @grd4 : new_position ∈ move(SDV_POSITION_env ↦ sp ↦ steer)
then
  @act1 : SDV_POSITION_env := new_position
  @act2 : SDV_SPEED_env := sp
  @act3 : SDV_STEERING_ANGLE_env := steer
end

/*
 * An extended event manual_moves moves an SDV in its target lane based the (new)
   steering angle and speed of human driver.
 * where a movement must lead into the lane
 */
event manual_move extends move
where
  /* new (target) position must be within set of position inside the lane */
  @grd2 : new_position ∈ move(SDV_POSITION_env ↦ SDV_SPEED_env ↦
    SDV_STEERING_ANGLE_env)
end

/*
 * Driver specifies a target speed when the ALC system starts working.
 */
event ALC_ON extends ALC_ON
any
  /* Driver may specify a target speed */

```

```

tsp
where
  @grd2: tsp ∈ SPEED
then
  @act2: ACC_target_speed := tsp
end

event ALC_OFF extends ALC_OFF
end

end

```

F.3 Second Refinement

F.3.1 Context c2

```

context c2 extends c1
sets
  IMAGE
  /* radar_pints to capture the distance and speed of leading vehicle */
  RADAR_READING
  STAGE
  LeftLane
  RightLane
constants
  camera
  /*
  * An abstract constant uses to show how autonomous controller gets the radar points from
  * multiple positions
  * in the driving environment.
  */
  radar
  /* Leading vehicle set*/
  LEADING_VEHICLE
  Seen_image
  /*
  * An abstract constant shows the expected results of a received radar readings
  */
  Seen_radar_reading

  OEDR_task
  Confidence_score
  target_position
  target_steering_angle
  target_speed

```

Perception
 Decision
 Control
 Intervention
 AutonomousDriving
 SAFE_DISTANCE
 SPEED_REDUCER
 speed_reduced

axioms

```

@def-STAGE: partition(STAGE, {Perception}, {Decision}, {Control}, {
  AutonomousDriving}, {Intervention})
/* define leading vehicle */
@Zero_or_more: LEADING_VEHICLE = {x · x ∈ Lane | {x}} ∪ {∅}

@score: Confidence_score = 0 .. 100
@typeof-camera: camera ∈ POSITION → IMAGE
@typeof-radar: radar ∈ POSITION → RADAR_READING
@typeof-image-seen: Seen_image ∈ (IMAGE × LeftLane × RightLane) →
  Confidence_score
@typeof-radar-seen: Seen_radar_reading ∈ (RADAR_READING) ↔ Lane
@typeof-path_recognition: OEDR_task ∈ ( LeftLane × RightLane × Confidence_score ×
  LEADING_VEHICLE) → P(POSITION)

@typeof-compute_target_position: target_position ∈ P(POSITION) → POSITION
@typeof-compute_target_steering_angle: target_steering_angle ∈ POSITION ×
  STEERING_ANGLE → STEERING_ANGLE_CHANGE
@typeof-compute_targer_speed: target_speed ∈ ( POSITION × SPEED ×
  LEADING_VEHICLE) → SPEED

@aim: ∀ SDV_position · SDV_position ∈ POSITION ⇒
(∃ SDV_speed, SDV_steering · SDV_steering ∈ STEERING_ANGLE ∧ SDV_speed ∈ SPEED ⇒
  move (SDV_position ↦ SDV_speed ↦ SDV_steering) ⊆ Lane
)
/*Safe distance*/
@distance: SAFE_DISTANCE ∈ ℕ1
@rd: SPEED_REDUCER ∈ ℕ1
@mitigation_speed_reduce: speed_reduced ∈ SPEED → min_speed .. SPEED_REDUCER
@reduced_speed_range: SPEED_REDUCER ≤ max_speed
end

```

F.3.2 Machine m2**machine m2 refines m1****sees c2**

/*

```

* This machine (M2) studies how the ALC system accomplishes its autonomous operations ,
  such as perception, decision–making and control,
* to identify the steering angle and speed that autonomously moves an SDV into a position
  inside the target lane
*/

```

variables

SDV_POSITION_env

ALC_Status

SDV_STEERING_ANGLE_env

SDV_SPEED_env

ACC_target_speed

```
/* × × × × *
```

```
* M2 ----- variables
```

```
× × × × */
```

IMAGE_env

```
/* Radar points variable */
```

RADAR_reading_env

leftLanePoints

rightLanePoints

desirePath

confidenceScore

```
/* Leading vehicle variable */
```

leadingVehicleSet

targetPosition

targetSteeringAngle

steeringAngleChange

targetSpeed

distance

stage

signal

invariants

@typeof–stage: stage ∈ STAGE

@typeof–desire_path: desirePath ⊆ POSITION

@typeof–perceived_confident: confidenceScore ∈ Confidence_score

@typeof_leftLanePoints: leftLanePoints ∈ LeftLane

@typeof_rightLanePoints: rightLanePoints ∈ RightLane

@typeof_leadingV: leadingVehicleSet ∈ LEADING_VEHICLE

@t: leadingVehicleSet ≠ ∅ ⇒ (∃ x · leadingVehicleSet = {x})

@typeof–target_position: targetPosition ∈ POSITION

```

@typeof-target_steering_angle: targetSteeringAngle ∈ STEERING_ANGLE
@typeof-current_steering_angle_change: steeringAngleChange ∈
    STEERING_ANGLE_CHANGE
@typeof-target_speed: targetSpeed ∈ SPEED

@typeof-target_distance: distance ∈ ℤ

@typeof-signal_movement: signal ∈ BOOL

@gluing_inv1: ALC_Status = ON ⇒ stage ∈ STAGE

/*
*(SR): The camera and radar always obtain the sensing data (image, radar points) of the
current position
*/
@consistency_1: IMAGE_env = camera(SDV_POSITION_env)
@consistency_2: RADAR_reading_env = radar(SDV_POSITION_env)

/*
*(SR): based on the detected left/right lane lines, confidence score and potential leading
vehicle, the perception component identifies the desired path
*/
@perceivedSensingData: stage = Perception ⇒ IMAGE_env ∈ ran(camera) ∧
(leftLanePoints ↦ rightLanePoints ↦ confidenceScore ↦ leadingVehicleSet) ∈ dom(
    OEDR_task)

/*
*(SR5): before changing the stage of ACL from perception to Decision, the desired path must
be identified
*/
@perceptionTask: stage = Decision ⇒ desirePath ∈ ran(OEDR_task)

@targetPosition: stage = Decision ∧ confidenceScore ≥ 80 ⇒
    desirePath ∈ dom(target_position)
@changeOfSteerSp: stage = Control ∧ confidenceScore ≥ 80 ⇒
    (targetPosition ↦ SDV_STEERING_ANGLE_env) ∈ dom(target_steering_angle) ∧
    (targetPosition ↦ SDV_SPEED_env ↦ leadingVehicleSet) ∈ dom(target_speed)

@change_ofSteeringAngle: stage = Control ∧ confidenceScore ≥ 80 ⇒
    steeringAngleChange ∈ ran(target_steering_angle)

@Control-consistency1: stage = Control ⇒ targetSteeringAngle ∈ STEERING_ANGLE
@Control-consistency2: stage = Control ⇒ targetSpeed ∈ SPEED

```

```
@Environment-consistency: stage = AutonomousDriving
  ⇒ move(SDV_POSITION_env ↦ targetSpeed ↦ targetSteeringAngle) ⊆ Lane
```

events

event INITIALISATION **extends** INITIALISATION

begin

```
@init-image: IMAGE_env := camera (init_position)
@init-radarReading: RADAR_reading_env := radar (init_position)
@init-stage: stage := Perception
@init-desire_path: desirePath := ∅
@init-confident_score: confidenceScore := 0
@typeof_leftLanePoints: leftLanePoints ∈ LeftLane
@typeof_rightLanePoints: rightLanePoints ∈ RightLane
@typeof_leadingVS: leadingVehicleSet := ∅
@init-target_position: targetPosition ∈ POSITION
@init-target_steering_angle: targetSteeringAngle ∈ STEERING_ANGLE
@init-change_steering_angle: steeringAngleChange ∈ STEERING_ANGLE_CHANGE
@init-target_speed: targetSpeed ∈ SPEED
@init-distance: distance := 0
@typeof_signal: signal := FALSE
```

end

event perception

any

```
leftLane /* given left lane */
rightLane /* given right lane */
```

when

```
@grd1: leftLane ⊆ LeftLane
@grd2: rightLane ⊆ RightLane
@grd3: ALC_Status = ON
@grd5: stage = Perception
```

then

```
@act1: IMAGE_env := camera (SDV_POSITION_env)
@act2: RADAR_reading_env := radar (SDV_POSITION_env)
@act3: leftLanePoints ∈ LeftLane
@act4: rightLanePoints ∈ RightLane
```

```
@act5: confidenceScore := Seen_image(IMAGE_env ↦ leftLanePoints ↦
  rightLanePoints)
```

/*

* In ACC, we randomly identify the leading vehicle based on the a received radar points

* It could be either any position inside the Lane or empty

*/

/* leading vehicle*/

```

@act6: leadingVehicleSet := Seen_radar_reading[{{RADAR_reading_env}}]
/*
* Here we show the predicted path that the SDV suppose
*/
@act7: desirePath := OEDR_task( leftLanePoints  $\mapsto$  rightLanePoints  $\mapsto$  confidenceScore
 $\mapsto$  leadingVehicleSet)
/* change a stage of ALC to be in Decision */
@act8: stage := Decision
end

```

event lowConfidenceScore_interven

```

where
@grd1: ALC_Status = ON
@grd2: stage = Decision
@grd3: confidenceScore < 80
then
@act1: stage := Intervention
end

```

event decision

```

when
@grd1: ALC_Status = ON
@grd2: stage = Decision
@grd3: confidenceScore  $\geq$  80
then
@act1: targetPosition := target_position(desirePath)
@act2: steeringAngleChange := target_steering_angle(targetPosition  $\mapsto$ 
SDV_STEERING_ANGLE_env)
/* a change of speed computed based on the identified (target) position, ACC target speed
and leading vehicle */
@act3: targetSpeed := target_speed(targetPosition  $\mapsto$  ACC_target_speed  $\mapsto$ 
leadingVehicleSet)
@act4: stage := Control
end

```

event control

```

when
@grd1: ALC_Status = ON
@grd2: stage = Control
@grd3: SDV_STEERING_ANGLE_env + steeringAngleChange  $\in$  STEERING_ANGLE
@grd4: move(SDV_POSITION_env  $\mapsto$  targetSpeed  $\mapsto$  (SDV_STEERING_ANGLE_env +
steeringAngleChange))  $\subseteq$  Lane
then
@act1: targetSteeringAngle := steeringAngleChange + SDV_STEERING_ANGLE_env
@act2: stage := AutonomousDriving
end

```



```

event correct_exceeding_min_steering
when
  @grd1 : ALC_Status = ON
  @grd2 : stage = Control
  @grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange < min_steering
then
  @act1 : stage := Intervention
  @act2 : targetSteeringAngle := min_steering
end

event correct_exceeding_max_steering
when
  @grd1 : ALC_Status = ON
  @grd2 : stage = Control
  @grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange > max_steering
then
  @act1 : stage := Intervention
  @act2 : targetSteeringAngle := max_steering
end

event correct_out_of_lane
where
  @grd1 : ALC_Status = ON
  @grd2 : stage = Control
  @grd3 : SDV_STEERING_ANGLE_env + steeringAngleChange ∈ min_steering .. max_steering
    ∧
    move(SDV_POSITION_env ↦ targetSpeed ↦ (SDV_STEERING_ANGLE_env +
      steeringAngleChange)) ∉ Lane
then
  @act1 : stage := Intervention
  @act2 : targetSteeringAngle := SDV_STEERING_ANGLE_env
end

/*
 * This is a refined event from an abstract event ALC_actuating for allowing a target steering
 * angle (autonomous steering angle) to be used
 * for moving an SDV in its target lane when the stage of the autonomous controller is in
 * AutonomousDriving
 */
event ALC_actuating_with_LV refines ALC_actuating
any
  // new position as longitudinal and lateral
  SDV_lon SDV_lat
  // leading vehicle as longitudinal and lateral
  LV_lon LV_lat

```

where

```

@mainGuard: leadingVehicleSet  $\neq \emptyset$ 
// coordinates as lon and lat
@grd1: targetPosition = SDV_lon  $\mapsto$  SDV_lat
@grd2: leadingVehicleSet = {LV_lon  $\mapsto$  LV_lat}
/* System status is ON*/
@grd3: ALC_Status = ON
/*steering angle change definition*/
@grd4: steeringAngleChange  $\in$  STEERING_ANGLE_CHANGE
/* new steering angle of ALC computed according to the current steering and a steering
change that ALC might require*/
@grd5: steeringAngleChange + SDV_STEERING_ANGLE_env  $\in$  STEERING_ANGLE
/* new speed definition*/
@grd6: targetSpeed  $\in$  min_speed .. ACC_target_speed
/* (SR3): the ALC system must actuate a steering angle to reach a target position that keeps
an SDV inside the target lane*/
@grd8: move(SDV_POSITION_env  $\mapsto$  targetSpeed  $\mapsto$  (SDV_STEERING_ANGLE_env +
steeringAngleChange))  $\subseteq$  Lane
@grd9: stage = AutonomousDriving
@grd10: steeringAngleChange + SDV_STEERING_ANGLE_env = targetSteeringAngle
@grd11: signal = FALSE

```

then

```

/* Change the current steering based on the steering change of ALC system*/
@act1: SDV_STEERING_ANGLE_env := (SDV_STEERING_ANGLE_env +
steeringAngleChange)
/* Change a SDV's speed by ALC*/
@act2: SDV_SPEED_env := targetSpeed
/* ready to move*/
@act3: signal := TRUE
// distance to leading
@act4: distance := LV_lon - SDV_lon

```

with

```

/* a change of steering angle is replaced by a change identified by the ALC system*/
@steering_angle_change: steering_angle_change = steeringAngleChange
/* a change of speed is replaced by a change identified by the ALC system*/
@sp: sp = targetSpeed

```

end

event ALC_actuating_without_LV **refines** ALC_actuating

where

```

@mainGuard: leadingVehicleSet =  $\emptyset$ 
/* System status is ON*/
@grd3: ALC_Status = ON
/*steering angle change definition*/
@grd4: steeringAngleChange  $\in$  STEERING_ANGLE_CHANGE

```

```

/* new steering angle of ALC computed according to the current steering and a steering
   change that ALC might require */
@grd5: steeringAngleChange + SDV_STEERING_ANGLE_env ∈ STEERING_ANGLE
/* new speed definition */
@grd6: targetSpeed ∈ min_speed .. ACC_target_speed
/* (SR3): the ALC system must actuate a steering angle to reach a target position that keeps
   an SDV inside the target lane */
@grd8: move(SDV_POSITION_env ↦ targetSpeed ↦ (SDV_STEERING_ANGLE_env +
   steeringAngleChange)) ⊆ Lane
@grd9: stage = AutonomousDriving
@grd10: steeringAngleChange + SDV_STEERING_ANGLE_env = targetSteeringAngle
@grd11: signal = FALSE
then
/* Change the current steering based on the steering change of ALC system */
@act1: SDV_STEERING_ANGLE_env := (SDV_STEERING_ANGLE_env +
   steeringAngleChange)
/* Change a SDV's speed by ALC */
@act2: SDV_SPEED_env := targetSpeed
/* ready to move */
@act3: signal := TRUE
/* distance */
@act4: distance := 0
with
/* a change of steering angle is replaced by a change identified by the ALC system */
@steering_angle_change: steering_angle_change = steeringAngleChange
/* a change of speed is replaced by a change identified by the ALC system */
@sp: sp = targetSpeed
end

event Manual_actuating extends Manual_actuating
where
@grd4: stage = Intervention
@grd5: signal = FALSE
then
/* ready to move */
@act3: signal := TRUE
end

event safe_distance_violation
where
@grd1: distance < SAFE_DISTANCE
@grd2: signal = TRUE
then
/* the stage of ALC changed to be in 'Intervention' */

```

```

@act1 : stage := Intervention
/* function to assume a system would reduce the speed*/
@act2 : targetSpeed := speed_reduced(SDV_SPEED_env)
@rest_distance : distance := 0
end

/* auto move whit leading vehicle */
event auto_move_with_LV refines auto_move
any
LV_lon
LV_lat
target_SDV_lon
target_SDV_lat
where
@grd0 : ALC_Status = ON
@grd1 : targetPosition ∈ Lane
@grd2 : targetSpeed ∈ SPEED
@grd3 : targetSteeringAngle ∈ STEERING_ANGLE
/* new (target) position must be within set of position inside the lane */
@grd4 : targetPosition ∈ move(SDV_POSITION_env ↦ targetSpeed ↦
    targetSteeringAngle)
@SDV : targetPosition = target_SDV_lon ↦ target_SDV_lat
@grd5 : signal = TRUE
@grd6 : stage = AutonomousDriving
@grd7 : LV_lon ↦ LV_lat ∈ Lane
/* new position keep safe distance */
@safe : SAFE_DISTANCE > LV_lon – target_SDV_lon
then
@act1 : SDV_POSITION_env := targetPosition
@act2 : SDV_SPEED_env := targetSpeed
@act3 : SDV_STEERING_ANGLE_env := targetSteeringAngle
@act4 : stage := Perception
@act5 : IMAGE_env := camera(targetPosition)
@act6 : RADAR_reading_env := radar(targetPosition)
@act7 : signal := FALSE
/* remove the old detecting of leading vehicle */
@act8 : leadingVehicleSet := {LV_lon ↦ LV_lat}
@reset_distance : distance := 0
with
/* a change of steering angle is replaced by a change identified by the ALC system */
@new_position : new_position = targetPosition
/* a change of speed is replaced by a change identified by the ALC system */
@sp : sp = targetSpeed
@steer : steer = targetSteeringAngle
end

```

```

/* auto move without leading vehicle */
event auto_move_without_LV refines auto_move
where
  @grd0: ALC_Status = ON
  @grd1: targetPosition ∈ Lane
  @grd2: targetSpeed ∈ SPEED
  @grd3: targetSteeringAngle ∈ STEERING_ANGLE
  /* new (target) position must be within set of position inside the lane */
  @grd4: targetPosition ∈ move(SDV_POSITION_env ↦ targetSpeed ↦
    targetSteeringAngle)
  @grd5: signal = TRUE
  @grd6: stage = AutonomousDriving
then
  @act1: SDV_POSITION_env := targetPosition
  @act2: SDV_SPEED_env := targetSpeed
  @act3: SDV_STEERING_ANGLE_env := targetSteeringAngle
  @act4: stage := Perception
  @act5: IMAGE_env := camera(targetPosition)
  @act6: RADAR_reading_env := radar(targetPosition)
  @act7: signal := FALSE
  /* remove the old detecting of leading vehicle */
  @act8: leadingVehicleSet := ∅
  @reset-distance: distance := 0
with
  /* a change of steering angle is replaced by a change identified by the ALC system */
  @new_position: new_position = targetPosition
  /* a change of speed is replaced by a change identified by the ALC system */
  @sp: sp = targetSpeed
  @steer: steer = targetSteeringAngle
end

event manual_move extends manual_move
where
  @grd0: ALC_Status = ON
  @grd3: signal = TRUE
  @grd4: stage = Intervention
then
  @act2: stage := Perception
  @act3: IMAGE_env := camera(new_position)
  @act4: RADAR_reading_env := radar(new_position)
  @act5: signal := FALSE
  /* remove the old detecting of leading vehicle */
  @act6: leadingVehicleSet := ∅
  @reset-distance: distance := 0
end

event ALC_ON extends ALC_ON

```

```

end
/*
 * This is an extended event to reset a stage of an autonomous controller to be at
 * the perception stage when a system is switched off
 */
event ALC_OFF extends ALC_OFF
then
  /* ALC_STAGE reset into initialization value*/
  @restALC: stage := Perception
  @rest-image: IMAGE.env := camera (init_position)
  @rest-radarPoints: RADAR_reading_env := radar (init_position)
  @rest_signal: signal := FALSE
  @reset-distance: distance := 0
end
end

```

F.4 Third Refinement

F.4.1 Machine m3

```

machine m3 refines m2
sees c2
/*
 * This machine (M3) investigates how the awareness level of a driver involves in the
 * autonomous operations of a system in order to
 * ensure that a human driver is a fallback option when the ALC system may issue a request to
 * intervene.
 */
variables

SDV_POSITION_env
ALC_Status
SDV_STEERING_ANGLE_env
SDV_SPEED_env
ACC_target_speed
IMAGE.env
RADAR_reading_env
leftLanePoints
rightLanePoints
desirePath
confidenceScore
leadingVehicleSet
targetPosition
targetSteeringAngle
steeringAngleChange
targetSpeed

```

```

distance
stage
signal
/*××××*
* M3 ----- variables
××××*/
awarenessLevel
warningMessage
handsOnSteeringWheel
sensitiveMonitoringFeature
invariants

@typeof-awarenessLevel: awarenessLevel ∈ BOOL
@typeof-warining: warningMessage ∈ BOOL
@typeof-handsOnSteeringWheel: handsOnSteeringWheel ∈ BOOL
@typeof-sensitiveMonitoringFeature: sensitiveMonitoringFeature ∈ BOOL

@compute_awarenesslevel: awarenessLevel = TRUE ⇒ (handsOnSteeringWheel = TRUE ∧
    sensitiveMonitoringFeature = TRUE)

@driver_Aware: awarenessLevel = TRUE ⇒ stage ∈ {Perception , Decision , Control ,
    Intervention , AutonomousDriving}

@send_interventionRequest: warningMessage = TRUE ⇒ stage = Intervention

@intervention_cases: warningMessage = TRUE
    ⇒
    confidenceScore < 80 ∨
    SDV_STEERING_ANGLE_env + steeringAngleChange ∉ STEERING_ANGLE ∨
    targetSpeed ∈ SPEED ∨
    move(targetPosition ↦ targetSpeed ↦ (SDV_STEERING_ANGLE_env +
    steeringAngleChange)) ∉ Lane ∨
    distance < SAFE_DISTANCE

events

event INITIALISATION extends INITIALISATION
begin
    @init-awareness_level: awarenessLevel := FALSE
    @init-warining: warningMessage := FALSE
    @init-handsOnSteeringWheel: handsOnSteeringWheel := FALSE
    @init-sensitiveMonitoringFeature: sensitiveMonitoringFeature := FALSE
end

event DMS_hands_on_wheel
where

```

```

@grd1: ALC_Status = ON
@grd2: awarenessLevel = FALSE
@grd3: handsOnSteeringWheel = FALSE
then
@act1: handsOnSteeringWheel := TRUE
/* compute based on both human monitoring features */
@act3: awarenessLevel := bool (sensitiveMonitoringFeature = TRUE)
end

event DMS_detect_sensitiveMonitoringFeature
where
@grd1: ALC_Status = ON
@grd2: awarenessLevel = FALSE
@grd3: sensitiveMonitoringFeature = FALSE
then
@act1: sensitiveMonitoringFeature := TRUE
@act3: awarenessLevel := bool (handsOnSteeringWheel = TRUE)
end

event DMS_hands_off_wheel extends ALC_OFF
where
@grd2: awarenessLevel = TRUE
@grd3: handsOnSteeringWheel = TRUE
then
@act2: awarenessLevel := FALSE
@act3: warningMessage := FALSE
@act4: handsOnSteeringWheel := FALSE
@act5: sensitiveMonitoringFeature := FALSE
end

event DMS_lost_sensitiveMonitoringFeature extends ALC_OFF
where
@grd3: awarenessLevel = TRUE
@grd4: sensitiveMonitoringFeature = TRUE
then
@act3: awarenessLevel := FALSE
@act4: warningMessage := FALSE
@act5: sensitiveMonitoringFeature := FALSE
@act6: handsOnSteeringWheel := FALSE
end

event ALC_ON extends ALC_ON
then
@reset-awareness_level: awarenessLevel := FALSE
@reset-waring: warningMessage := FALSE
@reset-handsOnSteeringWheel: handsOnSteeringWheel := FALSE
@reset-sensitiveMonitoringFeature: sensitiveMonitoringFeature := FALSE

```


end

event ALC_OFF **extends** ALC_OFF

then

@reset-*awareness_level*: *awarenessLevel* := FALSE

@reset-*waring*: *warningMessage* := FALSE

@reset-*handsOnSteeringWheel*: *handsOnSteeringWheel* := FALSE

@reset-*sensitiveMonitoringFeature*: *sensitiveMonitoringFeature* := FALSE

end

event perception **extends** perception

where

@grd6: *awarenessLevel* = TRUE

end

event lowConfidenceScore_interven **extends** lowConfidenceScore_interven

where

@grd4: *awarenessLevel* = TRUE

@grd5: *warningMessage* = FALSE

then

@act2: *warningMessage* := TRUE

end

event decision **extends** decision

where

@grd4: *awarenessLevel* = TRUE

end

event control **extends** control

where

@grd5: *awarenessLevel* = TRUE

@grd6: *warningMessage* = FALSE

end

event correct_exceeding_min_steering **extends** correct_exceeding_min_steering

where

@grd4: *awarenessLevel* = TRUE

@grd5: *warningMessage* = FALSE

then

@act3: *warningMessage* := TRUE

end

event correct_exceeding_max_steering **extends** correct_exceeding_max_steering

where

```

    @grd4: awarenessLevel = TRUE
    @grd5: warningMessage = FALSE
then
    @act3: warningMessage := TRUE
end

event correct_out_of_lane extends correct_out_of_lane
where
    @grd4: awarenessLevel = TRUE
    @grd5: warningMessage = FALSE
then
    @act3: warningMessage := TRUE
end

event ALC_actuating_with_LV extends ALC_actuating_with_LV
where
    /* driver is aware*/
    @grd12: awarenessLevel = TRUE
    /* no request to intervene*/
    @grd13: warningMessage = FALSE
end

event ALC_actuating_without_LV extends ALC_actuating_without_LV
where
    /* driver is aware*/
    @grd12: awarenessLevel = TRUE
    /* no request to intervene*/
    @grd13: warningMessage = FALSE
end

/*
 * Awareness level of a driver must be TRUE to enable the Manual actuating event
 * Warning message must be TRUE to enable this event. In any cases of intervention,
 * a driver will be receptive and change Warning message to be FALSE
 */
event Manual_actuating extends Manual_actuating
where
    @grd6: awarenessLevel = TRUE
    @grd7: warningMessage = TRUE
then
    @act4: warningMessage := FALSE
end

event safe_distance_violation extends safe_distance_violation
where
    @grd3: awarenessLevel = TRUE
    @grd4: warningMessage = FALSE

```

```
then
  @act3: warningMessage := TRUE
end

event auto_move_without_LV extends auto_move_without_LV
where
  @grd8: awarenessLevel = TRUE
  @grd9: warningMessage = FALSE
end

event auto_move_with_LV extends auto_move_with_LV
where
  @grd8: awarenessLevel = TRUE
  @grd9: warningMessage = FALSE
end

event manual_move extends manual_move
where
  /* driver is already react, see manual actuating*/
  @grd6: awarenessLevel = TRUE
  @grd7: warningMessage = FALSE
end

end
```


Appendix G

Intervention Timing Pattern

This appendix presents the complete Event-B models¹ for the intervention timing pattern, as discussed in Chapter 9.

G.1 Layer 4: Driver Reactions in Modelling Patterns

In this section, we present the complete Event-B version of the intervention timing pattern, as outlined in section 9.1 of Chapter 9. The majority of the proof obligations in modelling these functions were verified either automatically using Rodin provers or with the assistance of additional external prover plug-ins, such as SMT solvers (as illustrated in Figure G.1).

An Event-B machine for generic driver reactions is presented in the following section.

G.1.1 Intervention Timing Pattern

```
machine m0
/*
 * Models the time constraints when the automated system issues a request to intervene.
 */
variables
/*Red flag denotes a system enters a hazardous driving event and waits a fallback driver to
  react */
redFlag
/*Indicates any time of a system */
time
/*Trigger/hazard time when automated system issues a request to intervene*/
```

¹The Event-B models for developing design patterns of SDV systems are available as a Rodin archive at https://drive.google.com/drive/folders/1D7rVAJKCEh_-rVr9hKZZYdFDCxAP8r3F.

Element Name	Total	Auto	Manual	Rev.	Und.
m0	22	22	0	0	0
inv1	2	2	0	0	0
inv2	2	2	0	0	0
inv3	2	2	0	0	0
inv4	0	0	0	0	0
inv5	0	0	0	0	0
inv6	3	3	0	0	0
waiting_re...	5	5	0	0	0
alerting	4	4	0	0	0
alarm_state	4	4	0	0	0
INITIALISA...	7	7	0	0	0
request	6	6	0	0	0
tick	3	3	0	0	0
notify	3	3	0	0	0
intervene	3	3	0	0	0

FIGURE G.1: Intervention timing pattern, prover statistics.

requestTime

/*Sounding an alarm */

alarmFlag

/*Time waiting for a response from a fallback human before the alarm is sounding*/

alarmTime

/*Time when a fallback human may react to a request to intervene before the auditory notification is sounding*/

reactionTime

invariants

@inv1: time $\in \mathbb{N}$

@inv2: requestTime $\in \mathbb{N}$

@inv3: alarmTime $\in \mathbb{N}$

@inv4: redFlag $\in \text{BOOL}$

@inv5: alarmFlag $\in \text{BOOL}$

@inv6: reactionTime $\in \mathbb{N}$

/*

* This emphasizes that a system gives the fallback human a limited time to respond if the intervention request is sent (redFlag = TRUE).

* The limited time is included in the sum of alarm time and the time of issuing a request to intervene. Precisely, the current time of

* a system (time) can move beyond the time of issuing a request to intervene (requestTime) to the alert time (|alarmTime|), i.e., the waiting

* time can be written as (time \leq requestTime + alarmTime) where a current time of a system is equal to the time of issuing a request to

* intervene (time = requestTime).

*/

```

@waiting_response: redFlag = TRUE  $\wedge$  alarmFlag = FALSE  $\Rightarrow$  requestTime  $\leq$  time  $\wedge$  time
     $\leq$  alarmTime

/*
 * This invariant ensures that the alarm will be triggered if the system exceeds the waiting time
 * where the tick_took_intervene event blocks time progression until the alarm event triggered
    the alarm.
 */
@alerting: alarmFlag = TRUE  $\Rightarrow$  time  $\geq$  alarmTime

// if alarm then red flag is on
@alarm_state: alarmFlag = TRUE  $\Rightarrow$  redFlag = TRUE

/* alarm is a delay pattern from the request
 * intervene is a deadline pattern from the request.
 */

events
event INITIALISATION
begin
    /* none—deterministic assignment of a system time */
    @init—time: time := 0
    /* no start time of trigger event, hence zero */
    @init—requestTime: requestTime := 0
    @init—alarmTime: alarmTime := 0
    /*No initialization of human reaction time; hence is zero*/
    @init—reactionTime: reactionTime := 0
    @init—flagEvent: redFlag := FALSE
    @init—alarm: alarmFlag := FALSE
end

/* A trigger event indicates the entrance of a hazardous event when a system waits for a
    response before an alarm is raised */
event request
any
    /* Maximum time of a system waiting for a response before raises an alert*/
    duration
when
    /*Any time is given for waiting human's response*/
    @grd1: duration  $\in$   $\mathbb{N}_1$ 
    /*No intervention request and alarm is OFF*/
    @grd2: redFlag = FALSE  $\wedge$  alarmFlag = FALSE
then
    /*Specify a time of waiting for a driver before the alarm sounds*/
    @act1: alarmTime := time + duration
    /*Update a time of issuing a request to intervene*/
    @act2: requestTime := time

```

```

/*Update a flag of issuing a request to intervene*/
@act3: redFlag := TRUE
/*No reaction from human yet*/
@act4: reactionTime := 0
end

/*
 * tick_tock schedules the time progression associated with the alarm property
 */
event tick
where
/*Work only if a system issued a request to intervene*/
@flag_intervene: redFlag = TRUE
/*System time doesn't reach a alert time yet*/
@no_alarm: redFlag = TRUE  $\wedge$  alarmFlag = FALSE  $\Rightarrow$  time  $\leq$  alarmTime
/*System time arrives on alert time, so alarm must be operating*/
@alarmOn: (time = alarmTime  $\wedge$  redFlag = TRUE)  $\Rightarrow$  alarmFlag = TRUE
then
/* Increment timer */
@act1: time := time + 1
end

/*
 * The alarm event is 'sounding' when a driver does not react and the counter (time) reaches (
   alert time)
 */
event notify
where
/*System issues a request to intervene, while an alarm is not sounding */
@grd1: alarmFlag = FALSE  $\wedge$  redFlag = TRUE
/*System time equal to or has moved beyond alert time*/
@timeAlarm: time  $\geq$  alarmTime
then
/*Update value of alarm*/
@act1: alarmFlag := TRUE
end

/*
 * A fallback human reacts before sounding the auditory notification
 */
event intervene
when
/*Automated system issues a request to intervene, while an alarm is 'Off'*/
@grd1: redFlag = TRUE
/*Possible values of system time when a driver may react*/
@grd2: time < alarmTime

```



```

then
  /*Update a driver reaction time*/
  @receivedReaction : reactionTime := time
  /*Update values of flag and alarm: human did react*/
  @updateflag : redFlag := FALSE
end

end

```

G.2 ALC-Layer 4: Driver Reactions in the ALC System

In this section, we present the complete Event-B version of the application of the intervention timing pattern to the Automated Lane Centring (ALC) system, as outlined in section 9.2 of Chapter 9. The majority of the proof obligations in modelling these functions were verified either automatically using Rodin provers or with the assistance of additional external prover plug-ins, such as SMT solvers (as illustrated in Figure G.2).

Element Name	Total	Auto	Man.	Rev.	Und.
m4	69	69	0	0	0
typeof-audiotoryNo...	0	0	0	0	0
typeof-time	4	4	0	0	0
typeof-sentTime	11	11	0	0	0
typeof-receivedTime	8	8	0	0	0
typeof-alarm	11	11	0	0	0
waiting_response	17	17	0	0	0
alerting	17	17	0	0	0
INITIALISATION	7	7	0	0	0
lowConfidenceScor...	4	4	0	0	0
correct_exceeding_...	4	4	0	0	0
correct_exceeding_...	4	4	0	0	0
correct_out_of_lane	4	4	0	0	0
safe_distance_violati...	4	4	0	0	0
tick_took_intervene	3	3	0	0	0
response_before_ala...	3	3	0	0	0
response_after_alarm	3	3	0	0	0
alert	2	2	0	0	0
auto_move_with_LV	3	3	0	0	0
auto_move_without...	3	3	0	0	0
manual_move	5	5	0	0	0
DMS_hands_on_wh...	0	0	0	0	0
DMS_detect_sensitiv...	0	0	0	0	0
DMS_hands_off_wh...	5	5	0	0	0
DMS_lost_sensitive...	5	5	0	0	0
ALC_ON	5	5	0	0	0
ALC_OFF	5	5	0	0	0
perception	0	0	0	0	0
decision	0	0	0	0	0
control	0	0	0	0	0
ALC_actuating_with...	0	0	0	0	0
ALC_actuating_with...	0	0	0	0	0

FIGURE G.2: Prover statistics for modelling driver reactions in ALC system

An Event-B machine for driver reactions in the ALC system is presented in the following section.

G.2.1 Machine m4

machine m4 refines m3

sees c2

```
/*
 * This machine (M4) models the time constraints when an ALC system issues a request to
   intervene. For instance, ALC system may wait for
 * the driver's reaction, the time when an ALC system may raise the alarm and wait as well, or
   a time when a ALC releases the control of an SDV.
 */
```

variables

```
/*××××*
```

* M0 ----- variables

```
××××*/
```

```
/* (CA1) The (physical) position of the vehicle */
```

[SDV_POSITION_env](#)

```
/* System status is ON or OFF */
```

[ALC_Status](#)

```
/*××××*
```

* M1 ----- variables

```
××××*/
```

```
/* (CA2) The (actual) steering angle of the vehicle */
```

[SDV_STEERING_ANGLE_env](#)

```
/* (CA3) The (actual) speed of the SDV */
```

[SDV_SPEED_env](#)

```
/* target speed*/
```

[ACC_target_speed](#)

```
/*××××*
```

* M2 ----- variables

```
××××*/
```

```
/* (F3) The image would obtain from camera, which can be used as the input to the
   perception stage */
```

[IMAGE_env](#)

[RADAR_reading_env](#)

```
/* indicates left lane lines in a received image */
```

[leftLanePoints](#)

```
/* indicates right lane lines in a received image */
```

[rightLanePoints](#)

```
/* (CA4) set of positions seen as the desired path */
```

[desirePath](#)

```
/* (CA5) score to show the accuracy of detection process for seeing image and identifying
   desired path */
```

[confidenceScore](#)

[leadingVehicleSet](#)

```

/* (CA6) a new (target) position that identifies form a set of positions (desired path) */
targetPosition
/* (CA7) a target steering angle that would be used to autonomously move an SDV from
current position into target position */
targetSteeringAngle
/* (F5) a required change of steering angle from a current steering to a new (target) steering
*/
steeringAngleChange
/* target speed*/
targetSpeed
/*distance */
distance
/* shows the different stages of ALC system, such as Perception, planning and etc */
stage
/* A boolean flag indicates that an SDV is ready to move*/
signal

/*××××*
* M3 ----- variables
××××*/
/* (CA8) shows the awareness level of a driver */
awarenessLevel
/* (F6,F7) indicates that ALC system issues an intervention request */
warningMessage
/* (CA9) indicates whether a driver puts their hands on steering wheel inside the SDV or not
*/
handsOnSteeringWheel
/* (CA10) indicates whether a driver provides a sensitive monitoring feature or not */
sensitiveMonitoringFeature

/*××××××××××
* M4 ----- variables
××××××××××*/

/*
* (F8) Sounding an alarm, might be On(TRUE) or Off(FALSE)
*/
auditoryNotification

/*
* This indicates a current time of a system
*/
time

/*
* This indicates when a system issues a request to intervene (warningMessage_achf = TRUE
)

```

```
*/
```

```
sentRequest
```

```
/*
```

```
* Time when a driver may react to a request to intervene before the auditory notification is sounding.
```

```
*/
```

```
driverReact
```

```
/*
```

```
* Time waiting for a response from a driver before the the auditory notification is sounding
```

```
*/
```

```
alertTime
```

invariants

```
/*
```

```
* Alarm might be On (TRUE) or Off (False) when a driver ignores \reacts to an intervention request
```

```
*/
```

```
@typeof-auditoryNotification: auditoryNotification ∈ BOOL
```

```
/*
```

```
* current time of ALC system
```

```
*/
```

```
@typeof-time: time ∈ ℕ
```

```
/*
```

```
* Time of issuing a request to intervene
```

```
*/
```

```
@typeof-sentTime: sentRequest ∈ ℕ
```

```
/*
```

```
* Time when a driver may react
```

```
*/
```

```
@typeof-receivedTime: driverReact ∈ ℕ
```

```
/*
```

```
* Alarm time after ending waiting of a driver if a driver does not react
```

```
*/
```

```
@typeof-alarm: alertTime ∈ ℕ
```

```
/*
```

```
* (SR18):
```

```
* This emphasises that a system gives the driver a limited time to respond if the intervention request is sent (|warningMessage.achf = TRUE|).
```

```

* The limited time is included in the sum of alert time and the time of issuing a request to
  intervene. Precisely, the current time of a system ( $|time|$ ) can move beyond
* the time of issuing a request to intervene ( $|sentRequest\_tm|$ ) to the alert time ( $|alertTime|$ ),i.
  e., the waiting time can be written as  $|time \leq sentRequest\_tm + alertTime|$ 
* where a current time of a system is equal to the time of issuing a request to intervene ( $|time$ 
   $= sentRequest\_tm|$ ).
*/
@waiting_response: warningMessage = TRUE  $\wedge$  auditoryNotification = FALSE  $\Rightarrow$ 
    sentRequest  $\leq$  time  $\wedge$ 
    time  $\leq$  alertTime

/* (SR19)
* the invariant  $|@alerting|$  ensures that the auditory notification will be sent if the system
  exceeds the waiting time
* where the  $|tick\_took\_intervene|$  event blocks time progression until the  $|alert|$  event sent
  the auditory notification.
*/
@alerting: auditoryNotification = TRUE  $\wedge$  warningMessage = TRUE  $\Rightarrow$  time  $\geq$  alertTime

```

events

event INITIALISATION **extends** INITIALISATION

begin

```

/* alarm is OFF*/
@init-auditoryNotification: auditoryNotification := FALSE
    /* time variables initialized*/
/* any current time of a system*/
@init-_intervTime: time  $\in$   $\mathbb{N}$ 
/*No initialization of sending intervention request time; hence is zero*/
@init-sentTime: sentRequest := 0
/*No initialization of human reaction time; hence is zero*/
@init-receivedTime: driverReact := 0
/* No initialization of alert time; hence is zero */
@init-alertTime: alertTime := 0

```

end

```

/*
* These events specify a sending time when the ALC system issues a request to intervene
* (warningMessage_achf = TRUE)
*/

```

event lowConfidenceScore_interven **extends** lowConfidenceScore_interven

any

```

/* maximum time of a system waiting for a response before raises an alert */
duration

```

when

```

/*Time of waiting for a reaction from a driver*/

```

```

@grd6: duration  $\in \mathbb{N}_1$ 
@grd7: warningMessage = FALSE  $\wedge$  auditoryNotification = FALSE
then
  /* 1) Specify a time of waiting for a driver before the alarm sounds */
  @act3: alertTime := duration + time
  /* 2) Update a time of issuing a request to intervene*/
  @act4: sentRequest := time
end

event correct_exceeding_min_steering extends correct_exceeding_min_steering
any
  /* maximum time of a system waiting for a response before raises an alert */
  duration
when
  /*Time of waiting for a reaction from a driver*/
  @grd6: duration  $\in \mathbb{N}_1$ 
  @grd7: warningMessage = FALSE  $\wedge$  auditoryNotification = FALSE
then
  /* 1) Specify a time of waiting for a driver before the alarm sounds */
  @act4: alertTime := duration + time
  /* 2) Update a time of issuing a request to intervene*/
  @act5: sentRequest := time
end

event correct_exceeding_max_steering extends correct_exceeding_max_steering
any
  /* maximum time of a system waiting for a response before raises an alert */
  duration
when
  /*Time of waiting for a reaction from a driver*/
  @grd6: duration  $\in \mathbb{N}_1$ 
  @grd7: warningMessage = FALSE  $\wedge$  auditoryNotification = FALSE
then
  /* 1) Specify a time of waiting for a driver before the alarm sounds */
  @act4: alertTime := duration + time
  /* 2) Update a time of issuing a request to intervene*/
  @act5: sentRequest := time
end

event correct_out_of_lane extends correct_out_of_lane
any
  /* maximum time of a system waiting for a response before raises an alert */
  duration
when
  /*Time of waiting for a reaction from a driver*/

```

```

@grd6: duration  $\in \mathbb{N}_1$ 
@grd7: warningMessage = FALSE  $\wedge$  auditoryNotification = FALSE
then
  /* 1) Specify a time of waiting for a driver before the alarm sounds */
  @act4: alertTime := duration + time
  /* 2) Update a time of issuing a request to intervene*/
  @act5: sentRequest := time
end

event safe_distance_violation extends safe_distance_violation
any
  /* maximum time of a system waiting for a response before raises an alert */
  duration
when
  /*Time of waiting for a reaction from a driver*/
  @grd5: duration  $\in \mathbb{N}_1$ 
  @grd6: warningMessage = FALSE  $\wedge$  auditoryNotification = FALSE
then
  /* 1) Specify a time of waiting for a driver before the alarm sounds */
  @act4: alertTime := duration + time
  /* 2) Update a time of issuing a request to intervene*/
  @act5: sentRequest := time
end

/*
* The tick_tock schedules the time progression associated with the alarm property
*/
event tick_tock_intervene
where
  /* works only if a system issued a request to intervene */
  @grd1: warningMessage = TRUE
  /* Increment duration is inside the intervention timer */
  @grd2: warningMessage = TRUE  $\wedge$  auditoryNotification = FALSE  $\Rightarrow$  time  $\leq$  alertTime
  /*Time is reach alert time, auditory notification must be operating*/
  @grd3: (time = alertTime  $\wedge$  warningMessage = TRUE)  $\Rightarrow$  auditoryNotification = TRUE
then
  /* accept increment time */
  @act1: time := time + 1
end

/*
* A driver reacts before sounding the auditory notification
*/
event response_before_alarm extends Manual_actuating
when

```

```

/*ALC issues a request to intervene, while an alarm is OFF*/
@grd8: warningMessage = TRUE ∧ auditoryNotification = FALSE
/* (SR20): Possible values of current time when a driver may react */
@grd9: time ≥ sentRequest ∧ time < alertTime
then
/* update a driver reaction time*/
@upd–receivedTime: driverReact := time
end

/*
* A driver reacts after sounding the auditory notification
*/
event response_after_alarm extends Manual_actuating
when
/*ALC issues a request to intervene, while an alarm is On */
@grd8: warningMessage = TRUE ∧ auditoryNotification = TRUE
/*(SR∀): System has already raised an auditory notification */
@grd9: time ≥ alertTime
then
@upd–receivedTime: driverReact := time
/* rest an alarm*/
@up–alarm–state: auditoryNotification := FALSE
end

/*
* The alarm event is on when a driver does not react and the counter (time) reaches (alerttime
)
*/
event alert
when
/* ALC issues a request to intervene, while an alarm is OFF */
@grd1: auditoryNotification = FALSE ∧ warningMessage = TRUE
/*current time equal to or has moved beyond alert time */
@grd2: time ≥ alertTime
then
/* update value of auditory notification*/
@act1: auditoryNotification := TRUE
end

event auto_move_with_LV extends auto_move_with_LV
any
duration

```



```

/* A duration that the ALC system may take to complete its operations and moves an SDV
into a new position */

```

where

```

/* duration must be a positive number*/

```

```

@duration: duration  $\in \mathbb{N}_1$ 

```

then

```

@act9: time := time + duration

```

end

event auto_move_without_LV **extends** auto_move_without_LV

any

duration

```

/* A duration that the ALC system may take to complete its operations and moves an SDV
into a new position */

```

where

```

/* duration must be a positive number*/

```

```

@duration: duration  $\in \mathbb{N}_1$ 

```

then

```

@act9: time := time + duration

```

end

event manual_move **extends** manual_move

then

```

/* reset time*/

```

```

@upd-sendTime: sentRequest := 0

```

```

@upd-receivedTime: driverReact := 0

```

```

@upd-alertTime: alertTime := 0

```

```

/* rest variables of modelling driver reactions*/

```

```

@alram-of: auditoryNotification := FALSE

```

end

event DMS_hands_on_wheel **extends** DMS_hands_on_wheel

end

event DMS_detect_sensitiveMonitoringFeature **extends**

DMS_detect_sensitiveMonitoringFeature

end

event DMS_hands_off_wheel **extends** DMS_hands_off_wheel

then

```

/* reset time*/

```

```

@upd-sendTime: sentRequest := 0

```

```

@upd-receivedTime: driverReact := 0

```

```

@upd-alertTime: alertTime := 0

```

```
    /* rest variables of modelling driver reactions*/
    @alarm-of: auditoryNotification := FALSE
end

event DMS_lost_sensitiveMonitoringFeature extends DMS_lost_sensitiveMonitoringFeature
then
    /* reset time*/
    @upd-sendTime: sentRequest := 0
    @upd-receivedTime: driverReact := 0
    @upd-alertTime: alertTime := 0
    /* rest variables of modelling driver reactions*/
    @alarm-of: auditoryNotification := FALSE
end

event ALC_ON extends ALC_ON
then
    /* reset time*/
    @upd-sendTime: sentRequest := 0
    @upd-receivedTime: driverReact := 0
    @upd-alertTime: alertTime := 0
    /* rest variables of modelling driver reactions*/
    @alarm-of: auditoryNotification := FALSE
end

event ALC_OFF extends ALC_OFF
then
    /* reset time*/
    @upd-sendTime: sentRequest := 0
    @upd-receivedTime: driverReact := 0
    @upd-alertTime: alertTime := 0
    /* rest variables of modelling driver reactions*/
    @alarm-of: auditoryNotification := FALSE
end

event perception extends perception
end

event decision extends decision
end

event control extends control
end

event ALC_actuating_with_LV extends ALC_actuating_with_LV
end
```

```
event ALC_actuating_without_LV extends ALC_actuating_without_LV  
end
```

```
end
```


References

- [1] Asim Abdulkhaleq and Stefan Wagner. Experiences with applying stpa to software-intensive systems in the automotive domain. *STAMP Conference at MIT*, 2013.
- [2] Asim Abdulkhaleq, Stefan Wagner, and Nancy Leveson. A comprehensive safety engineering approach for software-intensive systems based on stpa. *Procedia Engineering*, 128:2–11, 2015.
- [3] Asim Abdulkhaleq, Daniel Lammering, Stefan Wagner, Jürgen Röder, Norbert Balbierer, Ludwig Ramsauer, Thomas Raste, and Hagen Boehmert. A systematic approach based on stpa for developing a dependable architecture for fully automated driving vehicles. *Procedia Engineering*, 179:41–51, 2017.
- [4] Asim Abdulkhaleq, Stefan Wagner, Daniel Lammering, Hagen Boehmert, and Pierre Blueher. Using stpa in compliance with iso 26262 for developing a safe architecture for fully automated vehicles. *arXiv preprint arXiv:1703.03657*, 2017.
- [5] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.
- [6] Jean-Raymond Abrial and Thai Son Hoang. Using design patterns in formal methods: An event-b approach. In *Theoretical Aspects of Computing-ICTAC 2008: 5th International Colloquium, Istanbul, Turkey, September 1-3, 2008. Proceedings 5*, pages 1–2. Springer, 2008.
- [7] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in event-b. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.
- [8] National Highway Traffic Safety Administration. Automated driving systems 2.0: A vision for safety, 2017. URL https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/13069a-ads2.0_090617_v9a_tag.pdf.

- [9] Michael Aeberhard, Sebastian Rauch, Mohammad Bahram, Georg Tanzmeister, Julian Thomas, Yves Pilat, Florian Homm, Werner Huber, and Nico Kaempchen. Experience, results and lessons learned from automated driving on germany's highways. *IEEE Intelligent transportation systems magazine*, 7(1):42–57, 2015.
- [10] Comma ai 'OpenPilot'. an open source driver assistance system, Sean (2021/03/22). URL <https://github.com/commaai/openpilot/>.
- [11] Fahad Alotaibi. Improving trustworthiness of self-driving systems. In *Rigorous State-Based Methods: 7th International Conference, ABZ 2020, Ulm, Germany, May 27–29, 2020, Proceedings 7*, pages 405–408. Springer, 2020.
- [12] Fahad Alotaibi, Thai Son Hoang, and Michael Butler. High-level rigorous template for analysing safety properties of self-driving vehicle systems. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1643–1648. IEEE, 2022.
- [13] Fahad Alotaibi, Thai Son Hoang, and Michael Butler. A rigorous iterative analysis approach for capturing the safety requirements of self-driving vehicle systems. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1697–1702, 2023. .
- [14] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Meireles Paixão, Filipe Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, pages 1–34, 2020.
- [15] Gerrit Bagschik, Andreas Reschka, Torben Stolte, and Markus Maurer. Identification of potential hazardous events for an unmanned protective vehicle. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 691–697. IEEE, 2016.
- [16] Lisanne Bainbridge. Ironies of automation. In *Analysis, design and evaluation of man-machine systems*, pages 129–135. Elsevier, 1983.
- [17] Stephan Baumgart, Joakim Froberg, and Sasikumar Punnekkat. Can stpa be used for a system-of-systems? experiences from an automated quarry site. In *2018 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–8. IEEE, 2018.
- [18] Christopher Becker, Larry Yount, Shane Rozen-Levy, John Brewer, et al. Functional safety assessment of an automated lane centering system. Technical report, United States. Department of Transportation. National Highway Traffic Safety ... , 2018.
- [19] Chaima Bensaci, Youcef Zennir, Denis Pomorski, Fares Innal, and Yiliu Liu. Distributed vs. hybrid control architecture using stpa and ahp-application to an autonomous mobile multi-robot system. *Int. Journal of Safety and Security Engin.*, 11: 1–12, 2021.

- [20] Pete Bigelow. Why level 3 automated technology has failed to take hold. *Automotive News*, 21, 2019.
- [21] Myra Blanco, Jon Atwood, Holland M Vasquez, Tammy E Trimble, Vikki L Fitchett, Joshua Radlbeck, Gregory M Fitch, Sheldon M Russell, Charles A Green, Brian Cullinane, et al. Human factors evaluation of level 2 and level 3 automated driving concepts. Technical report, National Highway Traffic Safety Admin., 2014.
- [22] Barry W Boehm and Philip N. Papaccio. Understanding and controlling software costs. *IEEE transactions on software engineering*, 14(10):1462–1477, 1988.
- [23] Chris Bogdiukiewicz, Michael Butler, Thai Son Hoang, Martin Paxton, James Snook, Xanthippe Waldron, and Toby Wilkinson. Formal development of policing functions for intelligent systems. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 194–204. IEEE, 2017.
- [24] Rajiv Bongirwar. Leveraging systems theoretic process analysis (stpa) for efficient iso 26262 compliance. Technical report, SAE Technical Paper, 2021.
- [25] John B Bowles. The new sae fmeca standard. In *Annual Reliability and Maintainability Symposium. 1998 Proceedings. International Symposium on Product Quality and Integrity*, pages 48–53. IEEE, 1998.
- [26] Michael Butler. Chapter 8 modelling guidelines for discrete control systems, July 2009. URL <http://www.deploy-project.eu/pdf/D15-D6.1-Advances-in-Methodological-WPs..pdf>.
- [27] Michael Butler and Jerome Falampin. An approach to modelling and refining timing properties in b. In: *Refinement of Critical Systems (RCS)*, 2002.
- [28] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini. Logic-based technologies for multi-agent systems: a systematic literature review. *Autonomous Agents and Multi-Agent Systems*, 35(1):1, 2021.
- [29] Dominique Cansell, Dominique Méry, and Joris Rehm. Time constraint patterns for event b development. In *B 2007: Formal Specification and Development in B: 7th International Conference of B Users, Besançon, France, January 17-19, 2007. Proceedings 7*, pages 140–154. Springer, 2006.
- [30] Li Chen, Tutian Tang, Zhitian Cai, Yang Li, Penghao Wu, Hongyang Li, Jianping Shi, Junchi Yan, and Yu Qiao. Level 2 autonomous driving on a single device: Diving into the devils of openpilot. *arXiv preprint arXiv:2206.08176*, 2022.
- [31] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-end autonomous driving: Challenges and frontiers. *arXiv preprint arXiv:2306.16927*, 2023.

- [32] Amit Chougule, Vinay Chamola, Aishwarya Sam, Fei Richard Yu, and Biplab Sikdar. A comprehensive review on limitations of autonomous driving and its impact on accidents and collisions. *IEEE Open Journal of Vehicular Technology*, 2023.
- [33] Amit Chougule, Vinay Chamola, Aishwarya Sam, Fei Richard Yu, and Biplab Sikdar. A comprehensive review on limitations of autonomous driving and its impact on accidents and collisions. *IEEE Open Journal of Vehicular Technology*, 2023.
- [34] Andy Christensen, Andrew Cunningham, Jerry Engelman, Charles Green, Charles Kawashima, Steve Kiger, Danil Prokhorov, Levasseur Tellis, Barbara Wendling, and Frank Barickman. Key considerations in the development of driving automation systems. In *24th enhanced safety vehicles conference. Gothenburg, Sweden*, 2015.
- [35] John Colley and Michael Butler. A formal, systematic approach to stpa using event-b refinement and proof. *21st Safety Critical System Symposium*, 2013.
- [36] Stephen Conner, Lisa Underwood, Minal Shah, Thom Sabo, Clayton Holstun, Brian Canfield, and Curt Voss. Designing a long-life, page-wide print-head. In *NIP & Digital Fabrication Conference*, pages 332–335. Society for Imaging Science and Technology, 2015.
- [37] Kenneth James Williams Craik. *The nature of explanation*, volume 445. CUP Archive, 1952.
- [38] Frank Crawley and Brian Tyler. *HAZOP: Guide to best practice*. Elsevier, 2015.
- [39] Louise A Dennis and Berndt Farwer. Gwendolen: A bdi language for verifiable agents. In *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Society for the Study of Artificial Intelligence and Simulation of Behaviour*, pages 16–23, 2008.
- [40] Dana Dghaym, Thai Son Hoang, Stephen R Turnock, Michael Butler, Jon Downes, and Ben Pritchard. An stpa-based formal composition framework for trustworthy autonomous maritime systems. *Safety science*, 136:105139, 2021.
- [41] Catherine Dinsmoor, Michael Mei, Brian Wong, Ayush Agrawal, and Grant Levene. Software requirements specification (srs) lane management system 2. *Technical Report. Michigan State Uni*, 2016.
- [42] Adel Dokhanchi, Heni Ben Amor, Jyotirmoy V Deshmukh, and Georgios Fainekos. Evaluating perception systems for autonomous vehicles using quality temporal logic. In *International Conference on Runtime Verification*, pages 409–416. Springer, 2018.

- [43] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.
- [44] Daniel Reyes Duran, Elliot Robinson, Andrew J Kornecki, and Janusz Zalewski. Safety analysis of autonomous ground vehicle optical systems: Bayesian belief networks approach. In *2013 Federated Conference on Computer Science and Information Systems*, pages 1419–1425. IEEE, 2013.
- [45] Ellen Edmonds. American trust in autonomous vehicles slips, May 2018. URL <https://newsroom.aaa.com/2018/05/aaa-american-trust-autonomous-vehicles-slips/>.
- [46] Alaa El Khatib, Chaojie Ou, and Fakhri Karray. Driver inattention detection in the context of next-generation autonomous vehicles design: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4483–4496, 2019.
- [47] Clifton A Ericson and Clifton Ll. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, volume 1, pages 1–9, 1999.
- [48] Raymond C Ezeoma, Ameze Big-Alabo, and Bartholomew Ogbonna. Model predictive control of autonomous vehicles to enhance driving performance and safety. *IUP Journal of Electrical & Electronics Engineering*, 16(2), 2023.
- [49] Lucas ER Fernandes, Vinicius Custodio, Gleifer V Alves, and Michael Fisher. A rational agent controlling an autonomous vehicle: Implementation and formal verification. *arXiv preprint arXiv:1709.02557*, 2017.
- [50] Michael Fisher, Louise Dennis, and Matt Webster. Verifying autonomous systems. *Communications of the ACM*, 56(9):84–93, 2013.
- [51] Cody Harrison Fleming, Melissa Spencer, John Thomas, Nancy Leveson, and Chris Wilkinson. Safety assurance in nextgen and complex transportation systems. *Safety science*, 55:173–187, 2013.
- [52] Lex Fridman. Human-centered autonomous vehicle systems: Principles of effective shared autonomy. *arXiv preprint arXiv:1810.01835*, 2018.
- [53] Lex Fridman, Daniel E Brown, Michael Glazer, William Angell, Spencer Dodd, Benedikt Jenik, Jack Terwilliger, Julia Kindelsberger, Li Ding, Sean Seaman, et al. Mit autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. *arXiv preprint arXiv:1711.06976*, 1, 2017.
- [54] Ivo Friedberg, Kieran McLaughlin, Paul Smith, David Lavery, and Sakir Sezer. Stpa-safesec: Safety and security analysis for cyber-physical systems. *Journal of information security and applications*, 34:183–196, 2017.

- [55] Warren Gilchrist. Modelling failure modes and effects analysis. *International Journal of Quality & Reliability Management*, 1993.
- [56] Samineh C Gillmore and Nathan L Tenhundfeld. The good, the bad, and the ugly: Evaluating tesla’s human factors in the wild west of self-driving cars. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, pages 67–71. SAGE Publications Sage CA: Los Angeles, CA, 2020.
- [57] Ileana Maria Greca and Marco Antonio Moreira. Mental models, conceptual models, and modelling. *International journal of science education*, 22(1):1–11, 2000.
- [58] Rânik Guidolini, Lucas G Scart, Luan FR Jesus, Vinicius B Cardoso, Claudine Badue, and Thiago Oliveira-Santos. Handling pedestrians in crosswalks using deep neural networks in the iara autonomous car. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [59] Abhishek Gupta, Alagan Anpalagan, Ling Guan, and Ahmed Shaharyar Khwaja. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, 10:100057, 2021.
- [60] Anthony Hall. Seven myths of formal methods. *IEEE software*, 7(5):11–19, 1990.
- [61] Akihiro Hata, Keijiro Araki, Shigeru Kusakabe, Yoichi Omori, and Hsin-Hung Lin. Using hazard analysis stamp/stpa in developing model-oriented formal specification toward reliable cloud service. In *2015 International Conference on Platform Technology and Service*, pages 23–24. IEEE, 2015.
- [62] Richard Hawkins. 1.2 – identifying hazardous system behaviour, practical guidance – automotive, Seen (2021/07/10). URL <https://www.york.ac.uk/media/assuring-autonomy/bodyofknowledgestructure/section1imagesanddocs/1.2%20automotive%20practical%20guidance%20Richard%20Hawkins.pdf>.
- [63] Thai Son Hoang. An introduction to the event-b modelling method. *Industrial Deployment of System Engineering Methods*, pages 211–236, 2013.
- [64] Thai Son Hoang, Dana Dghaym, Colin Snook, and Michael Butler. A composition mechanism for refinement-based methods. In *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 100–109. IEEE, 2017.
- [65] Thai Son Hoang, Naoto Sato, Tomoyuki Myosin, Michael Butler, Yuichiroh Nakagawa, and Hideto Ogawa. Policing functions for machine learning systems. In *Workshop on Verification and Validation of Autonomous Systems: Satellite Workshop of Floc.*, 2018.
- [66] Jeroen H Hogema, Sjoerd C De Vries, Jan BF Van Erp, and Raymond J Kiefer. A tactile seat for direction coding in car driving: Field evaluation. *IEEE Transactions on Haptics*, 2(4):181–188, 2009.

- [67] Hommes and Qi Van Eikema. Review and assessment of the iso 26262 draft road vehicle-functional safety. Technical report, SAE Technical Paper, 2012.
- [68] Giles Howard, Michael Butler, John Colley, and Vladimiro Sassone. Formal analysis of safety and security requirements of critical systems supported by an extended stpa methodology. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 174–180. IEEE, 2017.
- [69] Giles Howard, Michael Butler, John Colley, and Vladimiro Sassone. A methodology for assuring the safety and security of critical infrastructure based on stpa and event-b. *International Journal of Critical Computer-Based Systems*, 9(1-2):56–75, 2019.
- [70] Henry Alexander Ignatious, Manzoor Khan, et al. An overview of sensors in autonomous vehicles. *Procedia Computer Science*, 198:736–741, 2022.
- [71] Shantanu Ingle and Madhuri Phute. Tesla autopilot: semi autonomous driving, an uptick for future autonomy. *International Research Journal of Engineering and Technology*, 3(9):369–372, 2016.
- [72] ISO. Road vehicles – functional safety. *International Organization for Standardization*, 2018.
- [73] ISO. Pas 21448-road vehicles-safety of the intended functionality. *International Organization for Standardization*, 2019.
- [74] Ruochen Jiao, Hengyi Liang, Takami Sato, Junjie Shen, Qi Alfred Chen, and Qi Zhu. End-to-end uncertainty-based mitigation of adversarial attacks to automated lane centering. *arXiv preprint arXiv:2103.00345*, 2021.
- [75] Maryam Kamali, Louise A Dennis, Owen McAree, Michael Fisher, and Sandor M Veres. Formal verification of autonomous vehicle platooning. *Science of computer programming*, 148:88–106, 2017.
- [76] Maryam Kamali, Sven Linker, and Michael Fisher. Modular verification of vehicle platooning with respect to decisions, space and time. In *International Workshop on Formal Techniques for Safety-Critical Systems*, pages 18–36. Springer, 2018.
- [77] Davis King. Dlib c++ library. Access on: <http://dlib.net>, 2012.
- [78] OM Kirovskii and VA Gorelov. Driver assistance systems: analysis, tests and the safety case. iso 26262 and iso pas 21448. *IOP Conference Series: Materials Science and Engineering*, 534(1), 2019.
- [79] Philip Koopman. Challenges in autonomous vehicle validation: Keynote presentation abstract. In *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles*, pages 3–3, 2017.

- [80] Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety*, 4(1):15–24, 2016.
- [81] Philip Koopman, Uma Ferrell, Frank Fratrick, and Michael Wagner. A safety standard approach for fully autonomous vehicles. In *International Conference on Computer Safety, Reliability, and Security*, pages 326–332. Springer, 2019.
- [82] Neil Lerner, James Jenness, Emanuel Robinson, Timothy Brown, Carryl Baldwin, Robert E Llaneras, et al. Crash warning interface metrics. Technical report, United States. National Highway Traffic Safety Administration, 2011.
- [83] Michael Leuschel and Michael Butler. Prob: an automated analysis toolset for the b method. *International Journal on Software Tools for Technology Transfer*, 10(2): 185–203, 2008.
- [84] N. Leveson and J.P. Thomas. Stpa handbook. https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf, 2018.
- [85] Nancy Leveson, Mirna Daouk, Nicolas Dulac, and Karen Marais. Applying stamp in accident analysis. In *NASA Conference Publication*, pages 177–198. NASA; 1998, 2003.
- [86] Nancy G Leveson. *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.
- [87] Xiaodong Liu, Hongji Yang, and Hussein Zedan. Formal methods for the re-engineering of computing systems: a comparison. In *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC’97)*, pages 409–414. IEEE, 1997.
- [88] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR)*, 52(5):1–41, 2019.
- [89] Haneet Singh Mahajan, Thomas Bradley, and Sudeep Pasricha. Application of systems theoretic process analysis to a lane keeping assist system. *Reliability Engineering & System Safety*, 167:177–183, 2017.
- [90] Amel Mammam and Marc Frappier. Modeling of a speed control system using event-b. In *International Conference on Rigorous State-Based Methods*, pages 367–381. Springer, 2020.
- [91] Frederick Hirsch Marcellus Buchheit. A short introduction into trustworthiness., March 2018. URL https://www.iiconsortium.org/news/joi-articles/2018-Sept-JoI_A_Short_Introduction_into_Trustworthiness-TTG.pdf.
- [92] Thomas C McKelvey. How to improve the effectiveness of hazard and operability analysis. *IEEE Transactions on Reliability*, 37(2):167–170, 1988.

- [93] Natasha Merat, A Hamish Jamson, Frank CH Lai, Michael Daly, and Oliver MJ Carsten. Transition to manual: Driver behaviour when resuming control from a highly automated vehicle. *Transportation research part F: traffic psychology and behaviour*, 27:274–282, 2014.
- [94] Rhiannon Michelmore, Marta Kwiatkowska, and Yarin Gal. Evaluating uncertainty quantification in end-to-end autonomous driving control. *arXiv preprint arXiv:1811.06817*, 2018.
- [95] John A Michon. A critical view of driver behavior models: what do we know, what should we do? In *Human behavior and traffic safety*, pages 485–524. Springer, 1985.
- [96] Christopher A Miller and Raja Parasuraman. Designing for flexible interaction between humans and automation: Delegation interfaces for supervisory control. *Human factors*, 49(1):57–75, 2007.
- [97] General Motors. Self-driving safety report, Sean (2020-05-27). URL <https://www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf>.
- [98] General Motors. Self-driving safety report, Sean (2020-05-27). URL <https://www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf>.
- [99] Huansheng Ning, Rui Yin, Ata Ullah, and Feifei Shi. A survey on hybrid human-artificial intelligence for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2021.
- [100] National Transportation Safety Board (NTSB). Preliminary report hwy18mh010, Seen (2020/01/22). URL <https://www.nts.gov/Pages/PageNotFound.aspx?requestUrl=https://www.nts.gov/investigations/AccidentReports/Reports/HWY18MH010-%20prelim.pdf>.
- [101] Society of Automotive Engineers (SAE). Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems., Sean (2019/11/22). URL https://www.sae.org/standards/content/j3016_201806/.
- [102] O’Kane. Tesla starts using in-car camera for autopilot driver monitoring., Sean (2021-05-27). URL <https://www.theverge.com/2021/5/27/22457430/tesla-in-car-camera-driver-monitoring-system>.
- [103] D.G. Padmavathi and M. Shanmugapriya. A survey of attacks, security mechanisms and challenges in wireless sensor networks. *arXiv preprint arXiv:0909.0576*, 2009.
- [104] Yunxian Pan, Qinyu Zhang, Yifan Zhang, Xianliang Ge, Xiaoqing Gao, Shiyan Yang, and Jie Xu. Lane-change intention prediction using eye-tracking technology: A systematic review. *Applied Ergonomics*, 103:103775, 2022.

- [105] Raja Parasuraman, Thomas B Sheridan, and Christopher D Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, 30(3):286–297, 2000.
- [106] Darsh Parekh, Nishi Poddar, Aakash Rajpurkar, Manisha Chahal, Neeraj Kumar, Gyanendra Prasad Joshi, and Woong Cho. A review on autonomous vehicles: Progress, methods and challenges. *Electronics*, 11(14):2162, 2022.
- [107] David Lorge Parnas and Jan Madey. Functional documents for computer systems. *Science of Computer programming*, 25(1):41–61, 1995.
- [108] Haapanen Pentti and Helminen Atte. Failure mode and effects analysis of software-based automation systems. *VTT Industrial Systems, STUK-YTO-TR*, 190:190, 2002.
- [109] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, Vitor AM Jorge, Mara Abel, Raj Madhavan, Angela Locoro, Paulo Goncalves, Marcos E Barreto, Maki Habib, et al. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.
- [110] Yi Qi, Philippa Ryan Conmy, Wei Huang, Xingyu Zhao, and Xiaowei Huang. A hierarchical hazop-like safety analysis for learning-enabled systems. *arXiv preprint arXiv:2206.10216*, 2022.
- [111] Yi Qi, Yi Dong, Xingyu Zhao, and Xiaowei Huang. Stpa for learning-enabled systems: A survey and a new method. *arXiv preprint arXiv:2302.10588*, 2023.
- [112] Millie Radovic. Tech talk: Untangling the 5 levels of drone autonomy, last accessed 2022/11/25. URL <https://droneii.com/project/drone-autonomy-levels>.
- [113] Anand S Rao and Michael Wooldridge. Foundations of rational agency. In *Foundations of rational agency*, pages 1–10. Springer, 1999.
- [114] Stefan Riedmaier, Thomas Ponn, Dieter Ludwig, Bernhard Schick, and Frank Diermeyer. Survey on scenario-based safety assessment of automated vehicles. *IEEE access*, 8:87456–87477, 2020.
- [115] Highway safety research & communications (HSRC). Top safety picks by hsrc., last accessed 2022/06/22. URL <https://www.iihs.org/iihs/ratings/TSP-List>.
- [116] Asieh Salehi Fathabadi, Colin Snook, Dana Dghaym, Thai Son Hoang, Fahad Alotaibi, and Michael Butler. Designing critical systems using hierarchical stpa and event-b. In *Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings*, pages 220–237. Springer, 2023. URL https://doi.org/10.1007/978-3-031-33163-3_17.

- [117] Mohammad Reza Sarshogh and Michael Butler. Specification and refinement of discrete timing properties in event-b. *Electronic Communications of the EASST*, 36, 2011.
- [118] Stuart S Shapiro. Privacy risk analysis based on system control structures: Adapting system-theoretic process analysis for privacy engineering. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 17–24. IEEE, 2016.
- [119] Thomas B Sheridan and William L Verplank. Human and computer control of undersea teleoperators. Technical report, Massachusetts Inst of Tech Cambridge Man-Machine Systems Lab, 1978.
- [120] Tereza Soukupova and Jan Cech. Eye blink detection using facial landmarks. In *21st computer vision winter workshop, Rimske Toplice, Slovenia*, 2016.
- [121] Sardar Muhammad Sulaman, Armin Beer, Michael Felderer, and Martin Höst. Comparison of the fmea and stpa safety analysis methods—a case study. *Software quality journal*, 27(1):349–387, 2019.
- [122] Liangliang Sun, Yan-Fu Li, and Enrico Zio. Comparison of the hazop, fmea, fram and stpa methods for the hazard analysis of automatic emergency brake systems. *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, 2021.
- [123] John Thomas and Nancy Leveson. Generating formal model-based safety requirements for complex, software-and human-intensive systems. *Safety-Critical Systems Club*, 2013.
- [124] Stephen Thomas and Dirk Vandenberg. Harnessing uncertainty in autonomous vehicle safety. *Journal of System Safety*, 55(2):23–29, 2019.
- [125] Daniel Tokody, Imre János Mezei, and György Schuster. An overview of autonomous intelligent vehicle systems. In *Vehicle and Automotive Engineering*, pages 287–307. Springer, 2017.
- [126] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Sim-ataw: Simulation-based adversarial testing framework for autonomous vehicles. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 283–284, 2018.
- [127] Cumhur Erkan Tuncali, Georgios Fainekos, Danil Prokhorov, Hisahiro Ito, and James Kapinski. Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Transactions on Intelligent Vehicles*, 5(2): 265–280, 2019.
- [128] Ben Upcroft, Michael Moser, Alexei Makarenko, David Johnson, Ashod Donikian, Alen Alempijevic, Robert Fitch, William Uther, El Grötli, J Biermeyer,

- et al. Darpa urban challenge technical paper: Sydney-berkeley driving team. *University of Sydney*, 2007.
- [129] Marialena Vagia and Ørnulf Jan Rødseth. A taxonomy for autonomous vehicles for different transportation modes. *Journal of Physics: Conference Series*, 1357(1), 2019.
- [130] Qi Van Eikema Hommes et al. Assessment of safety standards for automotive electronic control systems. Technical report, United States. National Highway Traffic Safety Administration, 2016.
- [131] Volvo. Tips for using pilot assist car functions, volvo support., Sean (2021/03/22). URL <https://www.volvocars.com/uk/support/topics/use-your-car/car-functions/tips-for-using-pilot-assist>.
- [132] Jun Wang, Li Zhang, Yanjun Huang, and Jian Zhao. Safety of autonomous vehicles. *Journal of advanced transportation*, 2020, 2020.
- [133] LLC Waymo. On the road to fully self-driving. *Waymo Safety Report*, pages 1–43, 2017.
- [134] J. R. Wilson and A. Rutherford. Mental models, theory and application in human factors. *Human Factors*, 31(6):617–634, 1989.
- [135] Jeannette M Wing. A specifier’s introduction to formal methods. *Computer*, 23(9):8–22, 1990.
- [136] Sanaz Yeganefard and Michael Butler. Structuring functional requirements of control systems to facilitate refinement-based formalisation. *Electronic Communications of the EASST*, 46, 2012.
- [137] Sanaz Yeganefard, Michael Butler, and Abdolbaghi Rezazadeh. Evaluation of a guideline by formal modelling of cruise control system in event-b. In *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, pages 182–191, 2010.
- [138] William Young and Nancy Leveson. Systems thinking for safety and security. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 1–8, 2013.
- [139] William Young and Nancy G Leveson. An integrated approach to safety and security based on systems theory. *Communications of the ACM*, 57(2):31–35, 2014.
- [140] Zhiding Yu and Cha Zhang. Image based static facial expression recognition with multiple deep network learning. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pages 435–442, 2015.

-
- [141] Lihua Zhao, Ryutaro Ichise, Tatsuya Yoshikawa, Takeshi Naito, Toshiaki Kakinami, and Yutaka Sasaki. Ontology-based decision making on uncontrolled intersections and narrow roads. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 83–88. IEEE, 2015.
- [142] Chenyang Zhu, Michael Butler, and Corina Cirstea. Formalizing hierarchical scheduling for refinement of real-time systems. *Science of Computer Programming*, 189:102390, 2020.
- [143] Wenhao Zong, Changzhu Zhang, Zhuping Wang, Jin Zhu, and Qijun Chen. Architecture design and implementation of an autonomous vehicle. *IEEE access*, 6: 21956–21970, 2018.