



BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN

University of Applied Sciences

**Beuth Hochschule für Technik Berlin**

Fachbereich VII  
Studiengang Mechatronik  
Luxemburger Str. 10, 13353 Berlin  
<https://www.beuth-hochschule.de/>

**Masterarbeit zum Thema:**

*„Simulative Absicherung vernetzter  
automatisierter Fahrzeuge“*

zur Erlangung des akademischen Grades  
Master of Engineering(M. Eng.)

vorgelegt von:

**Sami Dhiab**

Matrikel-Nr: 871321

im Auftrag der **DLR**  
(Deutsches Zentrum für Luft- und Raumfahrt)



**Deutsches Zentrum  
für Luft- und Raumfahrt**

**Betreuer:** Prof. Dr.-Ing. S. Szatmári, Beuth Hochschule, Berlin

**Betreuer:** Dipl.-Inform. D. Heß, Deutsches Zentrum für Luft-  
und Raumfahrt e. V. (DLR)

**Gutachter:** Prof. Dr.-Ing. N. Lewkowicz, Beuth Hochschule, Berlin

**Tag der Abgabe:** 6. April 2021

# **Simulative Absicherung vernetzter automatisierter Fahrzeuge**

**Sami Dhiab**

## **Abstrakt**

Automatisierte Fahrzeuge(AFs<sup>1</sup>) sind eine aufstrebende Technologie mit dem Ziel die Mobilität neu zu schaffen, die Verkehrssicherheit zu steigern, die Effizienz und Nutzbarkeit des Autotransports zu verbessern, um letztendlich die Welt die heute bekannt ist, neu zu gestalten. Die größte Herausforderung in der Anschaffung von AFs ist die Gewährleistung der Sicherheit, da das Auto innerhalb kurzer Zeit kritische Entscheidungen automatisch treffen muss, um eine Bewegungsplanung in einer dynamischen Umgebung sicher zu ermöglichen. In dieser Masterarbeit wird das Thema „Simulative Absicherung vernetzter automatisierter Fahrzeuge“ untersucht.

Als erstes wird der Stand der Technik sowie Normen und Standards zur Absicherung automatisierter Fahrzeuge erforscht. Frameworks zur Erstellung und Auswertung von simulativen Testszenarien werden identifiziert und zusammengefasst. Als Lösungskonzept wird eine Kombination aus den NHTSA und PEGASUS Frameworks verwendet und im ADORe Framework implementiert. Grundlegende Testszenarien aus der NHTSA Empfehlung werden in einem Framework zum automatisierten Fahren (ADORe) angepasst und mit den zugehörigen Tools integriert. Nach dem Prinzip vom PEGASUS werden Testszenarien automatisch ausgeführt und ausgewertet. Bei der automatischen Ausführung wird das DevOps-Tool Gitlab CI/CD verwendet. Model-Checking Technik wird für die Auswertung von Testszenarien eingesetzt. Simulationsergebnisse werden analysiert und Methoden zum Debuggen werden beschrieben.

---

<sup>1</sup>Automatisierte Fahrzeuge bzw. selbstfahrende Autos: Dieser Begriff wird entlang dieser Arbeit mehrfach erwähnt, deswegen wurde AF bzw. AFs als Abkürzung verwendet(siehe Glossar für Begriffe und Akronyme)

# **Aufgabenblatt**

## **Aufgabenstellung**

- Recherche Standards und Stand der Technik für die Validierung automatisierter Fahrfunktionen
- Auswahl und Definition von Testszenarien
- Definition von Bewertungsmetriken für die Erfassung von Elementaraussagen in der Simulation
- Recherche von Regelsystemen und Logiken, sowie zugehöriger Analysetools, die z. B. eine zeitliche Verknüpfung Elementaraussagen ermöglichen
- Definition von Regeln für die Gesamtbewertung von Simulationsergebnissen

## **Software Kenntnisse**

- Software Eclipse ADORe: bietet eine modulare Softwarebibliothek und ein Toolkit für die Entscheidungsfindung, Planung, Steuerung und Simulation automatisierter Fahrzeuge.
- Eclipse SUMO: Simulation von Verkehr und Infrastruktur rund um automatisierte Fahrzeuge
- ROS (Robot Operating System): Middleware für Robotikentwicklung
- C++/Python : geforderte Programmiersprachen für das Projekt
- Gitlab CI/Cd: DevOps Automation Tool

# Inhaltsverzeichnis

## Abbildungsverzeichnis

## Tabellenverzeichnis

<b>1 Einführung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Übersicht der Beitrag . . . . .	2
1.3 Thesisstruktur . . . . .	3
<b>2 Stand der Technik</b>	<b>5</b>
2.1 Geschichte . . . . .	5
2.2 Taxonomie . . . . .	7
2.3 Begriffe . . . . .	8
2.4 Sensoren . . . . .	9
2.5 Hardware . . . . .	11
2.6 Software . . . . .	12
2.7 V2X . . . . .	14
<b>3 Sicherheitsstandards</b>	<b>17</b>
3.1 Unfälle . . . . .	17
3.2 Sicherheitsbegriffe . . . . .	19
3.3 ISO 26262 . . . . .	20
3.4 ISO/PAS 21448.1 . . . . .	20
3.5 NHTSA . . . . .	21
3.6 PEGASUS . . . . .	25
<b>4 Implementierung</b>	<b>27</b>
4.1 ADORe Framework . . . . .	27
4.1.1 ROS . . . . .	28
4.1.2 SUMO . . . . .	30
4.1.3 Gitlab . . . . .	32
4.1.4 Docker . . . . .	33
4.2 ADORe Stand & Demos . . . . .	33
4.3 Lösungsansatz . . . . .	34

4.4	Erstellung von Testszenarien . . . . .	37
4.4.1	Test Vorlage . . . . .	37
4.4.2	Parken . . . . .	41
4.4.3	Fahrspurfolgen . . . . .	41
4.4.4	Spurwechsel . . . . .	42
4.4.5	Kreuzungsnavigation . . . . .	43
4.5	Automatisierung Testszenarien . . . . .	45
4.5.1	Gitlab CI/CD Pipelines . . . . .	46
4.5.2	GitLab-YAML . . . . .	46
<b>5</b>	<b>Auswertung</b>	<b>53</b>
5.1	Model Checking . . . . .	53
5.1.1	Kripke-Struktur . . . . .	53
5.1.2	Temporale Logik . . . . .	54
5.1.3	Computation Tree Logic* . . . . .	55
5.1.4	Syntax . . . . .	55
5.2	Umsetzung . . . . .	56
5.2.1	ModelChecker.py . . . . .	56
5.2.2	MonitorClass.py . . . . .	57
5.2.3	ParseArgument . . . . .	57
5.2.4	AnalyseTopics . . . . .	57
<b>6</b>	<b>Ergebnisse</b>	<b>65</b>
6.1	Zusammenfassung . . . . .	65
6.2	Debuggen . . . . .	68
<b>7</b>	<b>Fazit</b>	<b>72</b>
7.1	Ausblick . . . . .	73
<b>Literatur</b>		<b>83</b>
<b>Glossar</b>		<b>84</b>
<b>Anhang</b>		<b>86</b>

# Abbildungsverzeichnis

2.1	Zeitplan für die Entwicklung von autonomen Fahrzeugen [11] . . . . .	6
2.2	Automatisierungsstufen nach SAE[11] . . . . .	7
2.3	Fahraufgabe [15] . . . . .	8
2.4	Sensoren Reichweite [11] . . . . .	10
2.5	Sensoren zusammengefasst[18] . . . . .	11
2.6	Software-Architektur[11] . . . . .	12
2.7	V2X Architektur[21] . . . . .	15
3.1	Tesla S Autopilot Mode, fataler Unfall in China [26] . . . . .	17
3.2	Tesla S Autopilot Mode, Ergebnis des fatalen Unfalls in Florida [28] . . . . .	18
3.3	Google Auto Waymo Unfall [29] . . . . .	18
3.4	Uber Auto Unfall [31] . . . . .	19
3.5	Vergleich zwischen ISO26262(Grün) und ISO/PAS 21448(Blau) des SW-Entwicklungsprozesses [34] . . . . .	21
3.6	Konventioneller ADS-Testzyklus [43] . . . . .	22
3.7	Testszenario Matrix [43] . . . . .	23
4.1	ROS wrapper [49] . . . . .	29
4.2	ROS Kommunikation [50] . . . . .	30
4.3	Beispiel Netzwerk Sumo [53] . . . . .	31
4.4	Beispiel simulation sumo [53] . . . . .	32
4.5	lane following [60] . . . . .	34
4.6	SUMO Karte complex . . . . .	38
4.7	OpenDrive Karte complex . . . . .	39
4.8	Auslesen der Startposition des Ego-Fahrzeugs . . . . .	40
5.1	Beispiel Kripke-Struktur [79] . . . . .	54
5.2	Beispiele CTL Formeln [82] . . . . .	55
5.3	Extraktion df in df1 und df2 . . . . .	58
5.4	Spezifikation des Systemmodells . . . . .	59
5.5	Resultat Model Checking K1 . . . . .	60
5.6	Near Goal Model . . . . .	60

5.7 Resultat Model Checking K2 . . . . .	61
5.8 Model Checking Max Velocity . . . . .	62
5.9 Sicherheitsabstand (Längs) Modell . . . . .	63
5.10 Sicherheitsabstand (seitlich) Modell . . . . .	63
5.11 Verzögerungsrate Modell . . . . .	64
6.1 CI Pipeline zum Beginn des Prozesses . . . . .	66
6.2 CI Pipeline zum Ende des Prozesses . . . . .	67
6.3 Ergebnis der Pipeline . . . . .	67
6.4 Überschreitung Timeout Jobs in CI-Pipeline . . . . .	69
6.5 Parallele Jobs in CI-Pipeline . . . . .	70
7.1 Carla Simulator [87] . . . . .	75
7.2 Aufgabe Verteilung innerhalb einer Hierarchie basierend auf der Dauer des Verhaltens [43] . . . . .	90
7.3 ODD Klassifikation [43] . . . . .	91
7.4 Einordnen (Merge) Testszenario [43] . . . . .	94
7.5 Vergleich taktische Manöver [43] . . . . .	98
7.6 Vergleich OEDR1 [43] . . . . .	99
7.7 Vergleich OEDR2 [43] . . . . .	100
7.8 Vergleich Ausfall Modi [43] . . . . .	101
7.9 waymo Fuzzing Prozess [5] . . . . .	102
7.10 PEGASUS methode for assessment of highly automated driving function (HAD-F) [10] . . . . .	103
7.11 Pegasus, Datenbank und Szenarien extrahieren [10] . . . . .	104
7.12 Pegasus, 6 Layers Szenario [10] . . . . .	104
7.13 Pegasus, test automation tool [10] . . . . .	105
7.14 Übersicht der PEGASUS SW-Architektur inklusive der Umgebungs- simulation [10] . . . . .	105
7.15 Überblick test000_template.launch . . . . .	107
7.16 Überblick test001_parking.launch . . . . .	108
7.17 Überblick test002_lane_following.launch . . . . .	109
7.18 Überblick test003_lane_change . . . . .	110
7.19 Überblick test003_lane_change_merge . . . . .	111
7.20 Überblick test004_navigate_intersection_onramp . . . . .	112
7.21 Überblick test004_navigate_intersection_offramp . . . . .	113
7.22 Überblick test004_navigate_intersection_Roundabouts . . . . .	114
7.23 Überblick test004_navigate_intersection_lrs . . . . .	116
7.24 Überblick test004_navigate_intersection_crosswalk . . . . .	117
7.25 Überblick test004_navigate_intersection_uturn . . . . .	118
7.26 Kripke-Struktur df1 . . . . .	120
7.27 Kripke-Struktur df2 . . . . .	121

# **Tabellenverzeichnis**

7.1	NHTSA empfohlene Verhaltenskompetenzen [5] . . . . .	87
7.2	Waymo weiterentwickelte Verhaltenskompetenzen [5] . . . . .	88
7.3	Vermeidung oder Minderung häufiger Absturzszenarien [5] . . . . .	89
7.4	L4 Hochautomatisierte Fahrzeugmanöver [43] . . . . .	90
7.5	Liste 19 NHTSA OEDR [43] . . . . .	91
7.6	Perform Lane Change Test Scenarios [43] . . . . .	93
7.7	NHTSA, Waymo und ADORe Vergleich Tabelle . . . . .	106
7.8	Logik, temporale und Pfad-Operatoren [82, 81] . . . . .	119

# Listings

4.1 start demo003 . . . . .	33
4.2 start test000 . . . . .	40
4.3 rostopic traffic . . . . .	41
4.4 test002_safe_distance_formula . . . . .	42
4.5 rostopic traffic . . . . .	43
4.6 Stage unit-test . . . . .	48
4.7 Stage ci_scenarios . . . . .	49
4.8 Testszenario Skript . . . . .	49
4.9 Rosbag-record Node . . . . .	50
4.10 Terminator Node . . . . .	50
4.11 Headless Mode . . . . .	50
4.12 Testszenario Skript . . . . .	51
4.13 Stage evaluation_deploy . . . . .	51
5.1 ModelChecker.py . . . . .	57
5.2 CheckModel Methode . . . . .	59
5.3 Model Checking IN COLLISION . . . . .	59
5.4 Model Checking NEAR GOAL . . . . .	61
5.5 class Analyser: Topic 2 . . . . .	61
5.6 result max velocity . . . . .	62
5.7 Beispiel1: Auswertung einer Bag-Datei . . . . .	64
5.8 Beispiel2: Auswertung einer Bag-Datei . . . . .	64
6.1 Debuggen "test3" . . . . .	68
6.2 Debuggen . . . . .	68
6.3 Debuggen "test3" zweite Variante . . . . .	68
6.4 Global Timeout . . . . .	69
7.1 test000_template.launch . . . . .	122
7.2 json traffic . . . . .	123
7.3 Stage Build . . . . .	125
7.4 class Analyser: ParseArgument . . . . .	126
7.5 class Analyser: Topic 1 . . . . .	127
7.6 class Analyser: Topic 3 . . . . .	128
7.7 class Analyser: Topic 4 . . . . .	128
7.8 Resultat1: Auswertung einer Bag-Datei . . . . .	129

7.9 Resultat2: Auswertung einer Bag-Datei . . . . .	130
7.10Gesamtes Skript . . . . .	131

# **Kapitel 1**

# **Einführung**

Fahrerassistenzsysteme[1] in modernen Autos wie z.b. Einparkhilfe, Geschwindigkeitsbegrenzer und Spurhalteassistenz werden seit Jahren in der Automobilindustrie eingesetzt. Trotz der hoch entwickelten Funktionen ist die Sicherheit der AFs nicht gewährleistet. Die Motivation dieser Arbeit basiert auf vielen Fragen der AF-Industrie. Die größte Herausforderung in dieser Industrie bleibt die Sicherheit.

## **1.1 Motivation**

Die AF-Industrie erlebt heutzutage eine rasante Entwicklung. Jahre zuvor hat das Rennen zwischen den größten Herstellern um das ideale AF bereitzustellen, begonnen. Die automatische Routenverfolgung(Navigation)[2] wird schon in der Flugindustrie seit Jahren eingesetzt. Die Technologie scheint nicht neu zu sein, allerdings spielen die Umgebung und die Konditionen eine wichtige Rolle. Die Luftfahrt im freien Himmel ist selbstverständlich ganz anders als die Stadtfahrt. In der Luftfahrt ist der Verkehr über die Radar-Technologie verfolgbar und das Flugzeug kann die Trajektorie anhand der „automatic pilot“ Technologie[2] automatisch folgen. Im Straßenverkehr ist die Anzahl an Verkehrsteilnehmern viel höher und der Einsatz von AFs viel riskanter. Folgende Fragen werden gestellt:

- Wie sicher sind derzeitige AFs ?
- Wie ist zu beweisen, dass das System sicher genug ist??

Zur besseren Verständlichkeit der Problematik muss eine Methodik gefunden werden, um Fahrfunktionen auszuwerten. Antworten sind jedoch nur durch Tests möglich. Laut den Wissenschaftlern[3] müssten Testfahrzeuge Hundert Millionen bis Milliarden Meilen gefahren werden, um ihre Zuverlässigkeit in Bezug auf Todesfälle und Verletzungen zu demonstrieren. Selbst wenn Feldtests unter diesen Umständen hunderte von Jahren bräuchten, können keine

definitiven Rückschlüsse auf die Zuverlässigkeit der AFs demonstriert werden. Außerdem können Probefahrten keine ausreichenden Beweise an die Fahrzeugsicherheit liefern[3].

Daher sollten Wissenschaftler und Entwickler diese Technologie anders angehen und alternative Testmethoden erbringen, die realistisch, machbar und günstig für die AF-Industrie sind. Zu diesem Zweck ist die Simulation von Testszenarien eine adäquate Lösung, die kostengünstig und verfügbar ist[3]. Die Simulation kann einen größeren Raum von Fällen abdecken, die in der Realität nicht realisiert werden können.

Weltweit ist der Prozess der Zertifizierung[4] der AFs noch nicht definiert worden. Gerade forschen renommierte Forschungsinstitute im Bereich der Validierung und Verifikation der Sicherheit von AFs. Der Einsatz von Testszenarien, sowohl in virtueller (Simulation) als auch in realer Umgebung (Teststrecken), wurde empfohlen[4].

In den USA haben einige Hersteller und Entwickler wie Waymo[5] und Tesla[6] frühzeitige Freigabe erhalten, um öffentliche Tests durchführen zu können. Tesla verkauft bereits seit einigen Jahren den Autopilot weltweit. Es besteht eine große Gefahr für frühzeitige Zulassungen zum Verkauf und Testen, wenn die Sicherheit nicht vollständig nachweisbar ist. Unfälle und tödliche Konsequenzen sind dadurch entstanden, Statistiken dazu werden in einem späteren Kapitel analysiert. Zusammengefasst ist die übliche Methode zum Testen bei der Industrie FOT<sup>1</sup>[7] ineffizient und ein besserer Ansatz ist unbedingt notwendig.

## 1.2 Übersicht der Beitrag

Das Hauptziel dieser Arbeit ist eine Methodik zu finden, die ins Framework ADORE[8] implementiert wird. Zuerst wird der Stand der Technik, Standards und Normen für die Validierung automatisierter Fahrfunktionen erforscht. Danach wird ein passendes Konzept ausgewählt und dementsprechend werden Testszenarien erstellt. Erstellte Testszenarien werden automatisch ausgeführt und ausgewertet. Anhand der Gitlab-CI wird dieser Prozess automatisch laufen. Die Auswertung erfolgt anhand bestimmter Bewertungskriterien. Elementare Aussagen und Logiken werden erforscht. Die Modellprüfung<sup>2</sup> wird als Technik für die Auswertung angewendet und die „Computation Tree Logic\*“ wird als Logik eingesetzt. Als letztes werden Regeln für die Gesamtbeurteilung bestimmt. Die Ergebnisse werden analysiert und Debug-Methoden

---

<sup>1</sup>Filed operation testing

<sup>2</sup>engl. Model Checking

werden beschrieben.

Das Deutsche Zentrum für Luft und Raumfahrt[9] „DLR“ ist bereits seit Jahren Teil der Forschung im Bereich „autonomes Fahren“ und war in verschiedenen Projekten rund um das Thema involviert. Das Projekt ADORE[8] ist ein Framework zur Entwicklung von Fahrfunktionen und Entscheidungsverhalten. Das Forschungsprojekt PEGASUS[10] liefert ein Verfahren für die Verifizierung und Validierung von Fahrfunktionen. Diese Arbeit erfolgt im direkten Zusammenhang mit dem Entwicklungsteam des Projekts ADORE. Für die Durchführung dieser Arbeit ist eine Einarbeitung in dem ADORE Framework notwendig. Dieses Framework wird in einem späteren Kapitel erklärt.

## 1.3 Thesisstruktur

### Einführung

Das erste Kapitel stellt das Hauptproblem der AF-Industrie dar und erklärt, welche Wege zur Problemlösung heutzutage Teil der Forschung sind.

### Stand der Technik

In diesem Kapitel wird der Stand der Technik von AFs erläutert. Die Geschichte, die Taxonomie und die Begrifflichkeit der AF-Industrie werden erklärt. Sensoren, Hardware, Software und Kommunikationsprotokolle werden ebenfalls beschrieben.

### Sicherheitstandards

In diesem Kapitel werden Sicherheitsbegriffe erklärt und Testfälle analysiert. Normen und Standards im Zusammenhang mit der Sicherheit von AFs werden beschrieben.

### Implementierung

Das ADORe Framework und dessen Tools werden kurz beschrieben. Eine Lösung aus den gegebenen Frameworks wird implementiert, um Szenarien zu erstellen. Ausgewählte Szenarien werden manuell mit der graphischen Oberfläche erstellt. Danach werden Szenarien automatisch ausgeführt.

## **Auswertung**

Die Theorie des Model-Checking wird erklärt. Die Umsetzung von Model-Checking als Auswertungsmethodik wird detailliert beschrieben.

## **Ergebnis**

Das Ergebnis eines vollautomatischen Testverfahrens anhand der CI-Pipeline wird gezeigt. Debug-Methoden werden ebenfalls untersucht.

## **Fazit**

Die im Rahmen dieser Masterarbeit erfüllten Aufgaben werden zusammengefasst. Eine Erweiterung dieser Arbeit wird anhand vorgeschlagener Aufgaben erläutert.

## **Kapitel 2**

# **Stand der Technik**

### **2.1 Geschichte**

Der Traum vom selbstfahrenden Auto ist fast so alt wie die Automobilindustrie[11]. Die folgende Abbildung 2.1 zeigt das Zeitdiagramm zur Entwicklung autonomer Fahrzeuge.

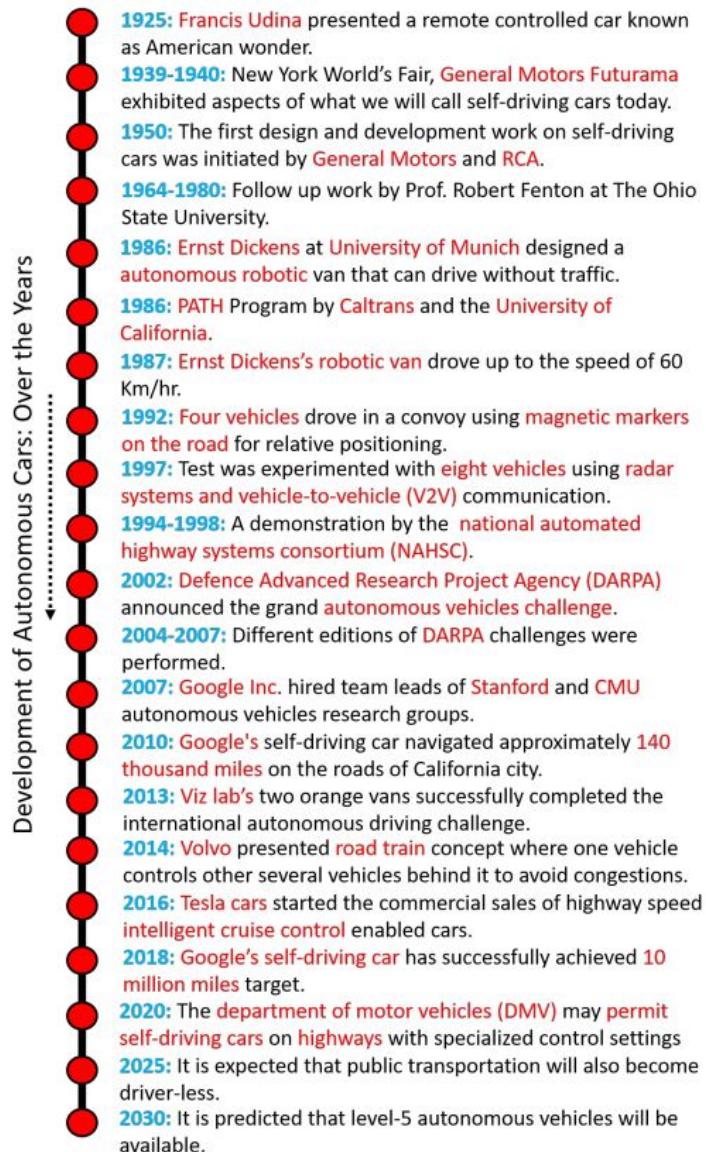


Abbildung 2.1: Zeitplan für die Entwicklung von autonomen Fahrzeugen [11]

Gemäß der Zeitachse 2.1 hat die Entwicklung von AFs bereits seit 1925 begonnen. In der Abbildung 2.1 fällt auf, dass mehrere Universitäten, Forschungsprojekte und Entwickler über viele Jahre zu der Forschung beigetragen haben. Die Top-Entwickler in dieser Branche sind Waymo und Tesla. Diese Hersteller haben unterschiedliche Ansätze für die Sicherheitsstrategie verwendet und über die Strategie online berichtet (s. Waymo Sicherheitsbericht [5] und Tesla Sicherheitsbericht [6]). Gemäß dem Zeitplan 2.1 wird die

Automatisierungsstufe 5 bis zum Jahre 2030 erreicht. Die Automatisierungsstufen werden als nächstes beschrieben.

## 2.2 Taxonomie

Die SAE<sup>1</sup>[11] hat eine Taxonomie der Fahrautomatisierung definiert, die in 6 Stufen organisiert ist. Die folgende Abbildung 2.2 stellt die Automatisierungsstufen dar[11]:

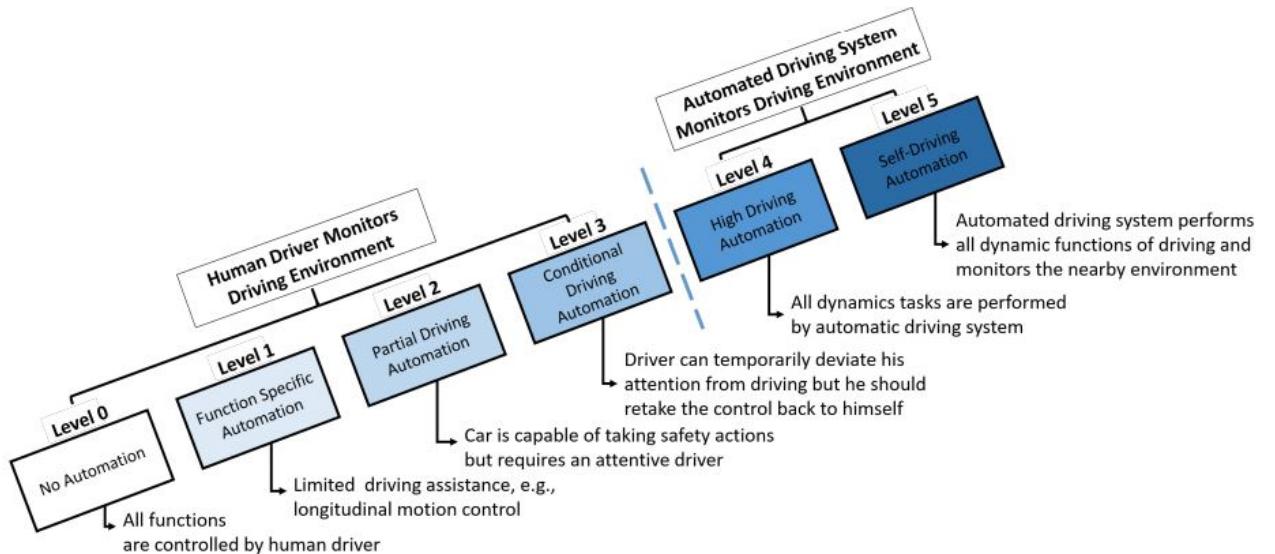


Abbildung 2.2: Automatisierungsstufen nach SAE[11]

- **Stufe 0:** Keine Automatisierung[11]: alle Fahraufgaben und Hauptsysteme werden von einem menschlichen Fahrer gesteuert(s. Abb2.2)
- **Stufe 1:** Funktionsspezifische Automatisierung[11]: Bietet begrenzte Fahrassistenz, z.B. laterale oder longitudinale Bewegungssteuerung(s. Abb2.2)
- **Stufe 2:** Teilweise Fahrautomatisierung: Mindestens zwei primäre Kontrollfunktionen werden kombiniert, um eine Aktion auszuführen. Erfordert die Aufmerksamkeit des Fahrers(s. Abb2.2).
- **Stufe 3:** Bedingte Fahrautamatik[11]: Ermöglicht begrenzte Automatisierung des Selbstfahrens, d.h. der Fahrer kann seine Aufmerksamkeit vorübergehend vom Fahren ablenken, um sich anderen Tätigkeiten

<sup>1</sup>Society of Automotive Engineers

zuzuwenden, aber die Anwesenheit des Fahrers ist immer erforderlich, um innerhalb weniger Sekunden wieder die Kontrolle zu übernehmen(s. Abb2.2)

- **Stufe 4:** Hohe Automatisierung des Fahrens[11]: Ein automatisiertes Fahrzeug übernimmt alle dynamischen Aufgaben des Fahrens, z.B. die Überwachung der Umgebung und die Bewegungssteuerung, allerdings nur in begrenztem ODD<sup>2</sup>. Dazu ist der Fahrer in der Lage, die volle Kontrolle über die sicherheitskritischen Funktionen unter bestimmten Szenarien zu ergreifen(s. Abb2.2)
- **Stufe5:** Volle Automatisierung[11]: Schließlich ist das System vollständig autonom und seine ODD ist unbegrenzt. Das bedeutet, dass es unter allen erforderlichen Bedingungen arbeiten kann. Die fünfte Stufe ist der Punkt, an dem unsere Gesellschaft einen transformatorischen Wandel erleben wird (s. Abb2.2)

## 2.3 Begriffe

In diesem Abschnitt werden fachspezifische Begriffe des autonomen Fahrens sinngemäß[12, 13] erläutert. Der erste Begriff ist „Ego“. Das Ego- bzw. das Egofahrzeug bezieht sich auf ein Fahrzeug, welches im Gegensatz zu anderen vergleichbaren Fahrzeugen oder Objekten autonom gesteuert wird [13]. Der zweite Begriff ist „Fahraufgabe<sup>3</sup>“. Die Fahraufgabe[14] ist in Abbildung2.3 dargestellt.

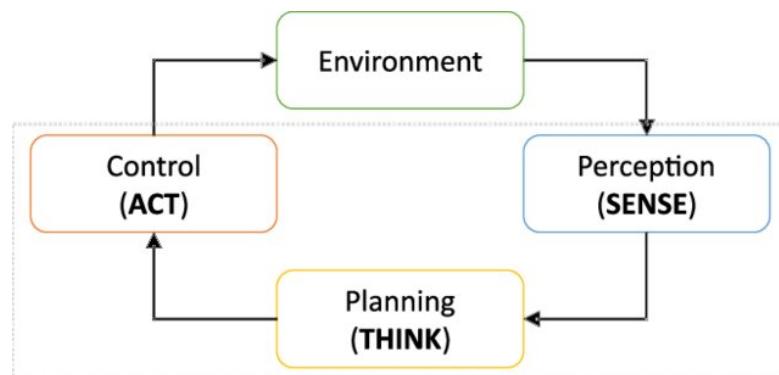


Abbildung 2.3: Fahraufgabe [15]

Die Fahraufgabe laut[14] setzt sich aus drei Teilaufgaben zusammen:

---

<sup>2</sup>Operational Design Domain

<sup>3</sup>engl. Driving Task

- 1-Perception(Sense)
- 2-Planning(Think)
- 3-Control(Act)

1- Die erste Teilaufgabe ist die Wahrnehmung<sup>4</sup>. Die Wahrnehmung bezieht sich auf das Wahrnehmen der Umgebung. Damit werden verschiedene Elemente in der Umgebung um das Ego identifiziert[14].

2- Die zweite Teilaufgabe ist die Planung<sup>5</sup>. Die Planung bezieht sich auf das Planen einer Trajektorie (von A nach B) und die langfristige Entscheidungen während der Fahrt[14].

3- Die dritte Teilaufgabe ist die Kontrolle<sup>6</sup>, wobei richtige kurzfristige Entscheidungen in Bezug auf Lenkung, Bremsen, und Beschleunigung getroffen werden müssen um die Position und Geschwindigkeit des Fahrzeugs auf der Straße zu bestimmen[14].

Der interne Mechanismus der Fahraufgabe wird im Abschnitt Software detailliert erklärt.

Objekt- und Ereigniserkennung sowie Reaktionserkennung(OEDR)<sup>7</sup> bedeutet Objekte und Ereignisse zu erkennen und angemessen darauf reagieren zu können. Operational Design Domain(ODD) beschreibt die Betriebsbedingungen, unter denen das System funktionieren soll. Es handelt sich dabei um die Umgebung, die Fahrbahnstruktur und die Tageszeit. In der Regel wird ODD vom Hersteller festgelegt[14]. Weitere Begriffe[13] befinden sich im Glossar7.1.

## 2.4 Sensoren

In diesem Abschnitt werden Sensortypen der AFs gemäß[16, 17] erläutert. Sensoren sind in zwei Kategorien unterteilt:

- Exteroceptive Sensoren: Sensoren, die die Umwelteigenschaften erfassen.
- Proprioceptive Sensoren: Sensoren, die die Eigenschaften des Ego-Fahrzeuges ermitteln.

---

<sup>4</sup>engl. Perception

<sup>5</sup>engl. Planning

<sup>6</sup>engl. Control

<sup>7</sup>engl. Object and Event Detection and Response

Die exterozeptiven Sensoren in AFs sind:

- Kamera: eignet sich hervorragend zur Erfassung von Informationen während der Fahrt. Sie zählt zu den wichtigsten Sensoren im AF. Einige Hersteller vermuten, dass eine gute Positionierung von Kameras und eine gute Bildverarbeitung für die Fahraufgabe ausreichend sind[16].
- Lidar: emittiert Lichtimpulse in die Umgebung und der reflektierte Rückstrahl wird gemessen. Durch die Messung der reflektierten Strahlen und seiner Laufzeit kann der Abstand zum Objekt eingeschätzt werden[16].
- Radar: sendet Funkwellen aus und dient dazu, Geschwindigkeit und Position zu ermitteln und kann sogar unter schlechtem Wetter sehr gut funktionieren[16].
- Ultrasonics: (auch Sonar genannt) funktioniert nach ähnlichem Prinzip wie der Radar, allerdings mit Schallwellen, welcher kleine Reichweiten messen kann und daher für das Parken sehr gut geeignet ist[16].

In der Abbildung 2.4 sind die obengenannten Sensoren anhand der Reichweite farbig dargestellt.

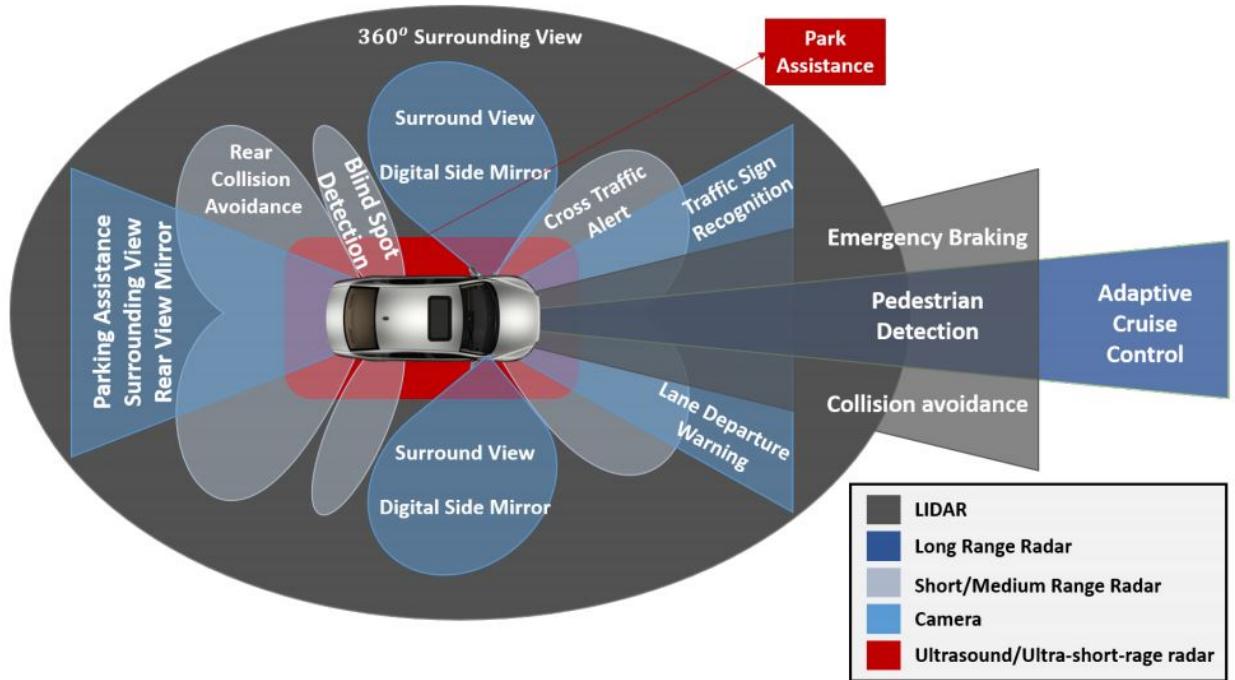


Abbildung 2.4: Sensoren Reichweite [11]

Die Propriozeptiven Sensoren in AFs sind[16]:

- GNSS<sup>8</sup>: Empfänger können die Position, die Geschwindigkeit und die Richtung des Ego-Fahrzeugs ermitteln[16].
- IMU: misst die Drehgeschwindigkeit, die Beschleunigung des Ego-Fahrzeugs[16].
- Radodometriesensor<sup>9</sup>: nimmt Raddrehzahlen auf und erfasst den Kilometerstand [16].
- GPS: eignet sich als Referenzuhr da GPS eine exakte Zeitsteuerung besitzt. Die Sensormessungen sollten mit synchronem Zeitstempel erfasst werden, um die „Sensorfusion“ korrekt zu erhalten[16].

In der Abbildung 2.5 sind die Sensoren in AF zusammengefasst[16].

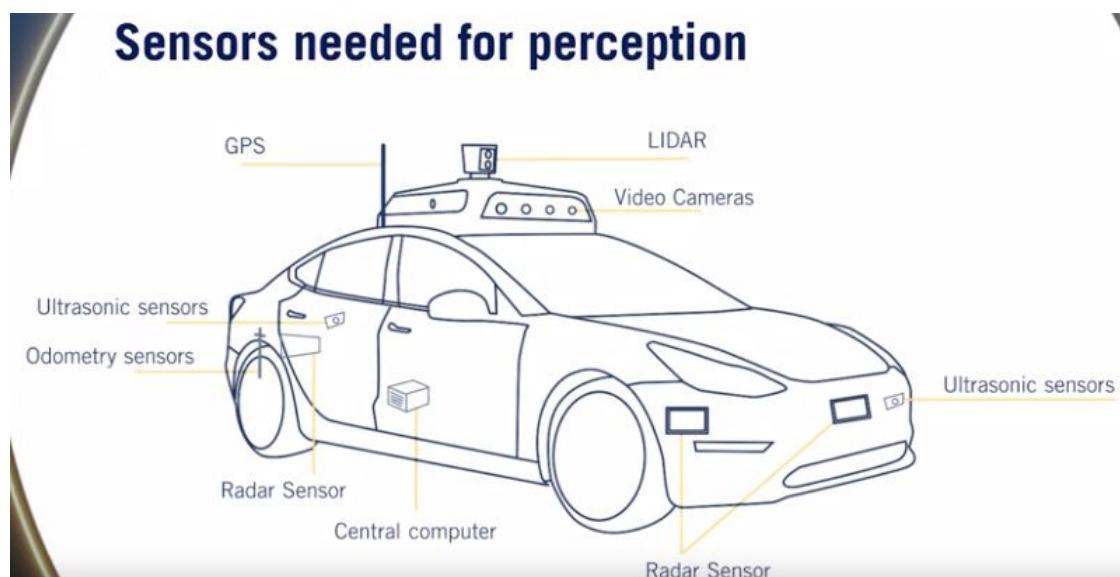


Abbildung 2.5: Sensoren zusammengefasst[18]

## 2.5 Hardware

In diesem Abschnitt werden die Hardware Anforderungen gemäß[16] beschrieben. Der wichtigste Teil der Hardware ist das zentrale Computergehirn (engl. Central Computer, s. Abbildung 2.5), das als die Hauptentscheidungseinheit des Autos ist. Es bekommt die Daten aus den Sensoren und gibt basierend darauf die Befehle zum Fahren aus. Die meisten Unternehmen

---

<sup>8</sup>Global Navigation Satellite Systems

<sup>9</sup>engl. wheel odometry sensor

entwerfen ihre eigenen Computersysteme, die den spezifischen Anforderungen ihrer Sensoren und Algorithmen entsprechen[16]. Zudem sind entwickelte Hardware Einheiten von Firmen wie Nvidia und EyeQ sehr gut auf dem Markt verbreitet und bereit zum Einsatz[16]. Jedes Computergehirn für autonomes Fahren benötigt sowohl serielle als auch parallele Computing-Module. Insbesondere für die Bild- und LIDAR-Verarbeitung zur Segmentierung, Objekterkennung und Mapping[16]. Je nach Hersteller und Kundenanforderung sind verschiedene Einheiten im Einsatz. Hierzu gehören die GPUs, FPGAs und ASICs, die parallelisierbare Rechenaufgaben (z.B. Aufgaben zur ML-Bildverarbeitung und DL-neuronale Netze) zu beschleunigen[16].

## 2.6 Software

In diesem Abschnitt wird die Software-Architektur gemäß[19] erklärt. Die Software-Architektur ist in der Abbildung 2.6 dargestellt.

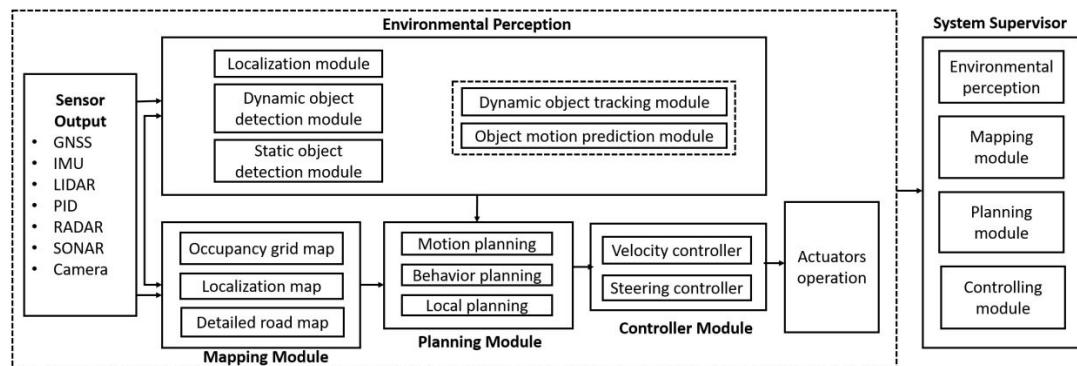


Abbildung 2.6: Software-Architektur[11]

### Sensor Output

Daten aus den oben genannten Sensoren(s. Abschnitt Sensoren) werden ständig erfasst(s. Abbildung 2.6). Die Daten der Sensorik werden in zwei Module weitergeleitet. Diese umfassen das Umgebungswahrnehmungsmodul (engl. environment perception) und das Umgebungs mappingsmodul (engl. Mapping Module)[19].

### Environment Perception

Das Umgebungswahrnehmungsmodul hat zwei Hauptaufgaben[19] (s. Abbildung 2.6):

- Identifizierung der aktuellen Position des Ego-Fahrzeugs im Raum
- Klassifizierung und Lokalisierung wichtiger Elemente der Umgebung für die Fahraufgabe

Das Umgebungswahrnehmungsmodul hat fünf Untermodule [19] (s. Abbildung 2.6):

- Lokalisierungsmodul: Aus verschiedenen Quellen(GPS, IMU und Radarometrie) wird die Ego-Position ausgegeben.
- Erkennung dynamischer Objekte: durch verschiedene Quellen (Kamera, LIDAR) werden dynamische Objekte erkannt und via 3D-Bounding-Boxen klassifiziert.
- Dynamische Objekte Tracking-Modul: Nach der Erkennung werden dynamische Objekte über die Zeit verfolgt, um die Position und den Wegverlauf zu identifizieren.
- Objekt Prediktion-Modul: Der Wegverlauf wird mit der Straßenkarte verwendet, um den zukünftigen Pfad dynamischer Objekte vorherzusagen
- Erkennung statischer Objekte: Stützt sich ebenfalls auf eine Kombination aus Kamera und LIDAR-Daten, um statische Objekte zu identifizieren

## **Mapping Module**

Das Umgebungsmappingsmodul (engl. Mapping Module) hat drei Karten, um Kollisionen zu vermeiden und Bewegungen zu planen[19] (s. Abbildung 2.6):

- Die Belegungsrasterkarte: Aus den LIDAR-Daten wird die Karte mit allen statischen Objekten erstellt. Die Karte ist mit Rasterzellen dargestellt.
- Die Lokalisierungskarte: Die Sensordaten werden mit der Karte verglichen, um die Umgebungszustandsschätzung zu verbessern und das Ego-Fahrzeug genau lokalisieren zu können.
- Die detaillierte Straßenkarte: Enthält Straßensegmente, die das Ego-Fahrzeug durchfährt. Basiert auf einer Kombination aus alten und neuen Karten-Daten, um eine aktualisierte Bewegungsplanung zu ermöglichen.

## **Planning Module**

Das Planungsmodul (engl. Planning Module) verwendet kombinierte Ausgangsdaten aus den letzten Modulen, um einen Plan durch die Umgebung zu erstellen. Das Planungsmodul hat drei Untermodule[19] (s. Abbildung 2.6):

- Bewegungsplaner: erstellt eine langfristige Planung vom aktuellen Standort bis zum Endziel.
- Verhaltensplaner: bekommt Daten aus dem Bewegungsplaner und bearbeitet kurzfristige Planung, indem Aktionen oder Manöver festgelegt sind.
- Lokale Planer: Führt sofortige oder reaktive Planung durch und definiert ein Geschwindigkeitsprofil mit Berücksichtigung der Einschränkungen durch die Umgebung. Die Umsetzung übernimmt das Steuerungsmodul.

## **Controller Module**

Das Steuerungsmodul[19] (engl. Controller Module) setzt die Planung um. Dabei wird der Lenkwinkel, die Gas- und Bremspedalstellung sowie die Gangeinstellung eingestellt, um den geplanten Weg genau zu folgen. Das Steuerungsmodul hat zwei Untermodule[19] (s. Abbildung 2.6):

- Der Longitudinalregler bzw. der Geschwindigkeitsregler (engl. velocity controller): regelt die Gaspedale, die Gänge und das Bremssystem, um die gewünschte Geschwindigkeit zu erreichen.
- Der Lateralregler bzw. der Lenkungsregler (engl. steering controller) gibt den richtigen Lenkwinkel aus, um die geplante Fahrspur beizubehalten.

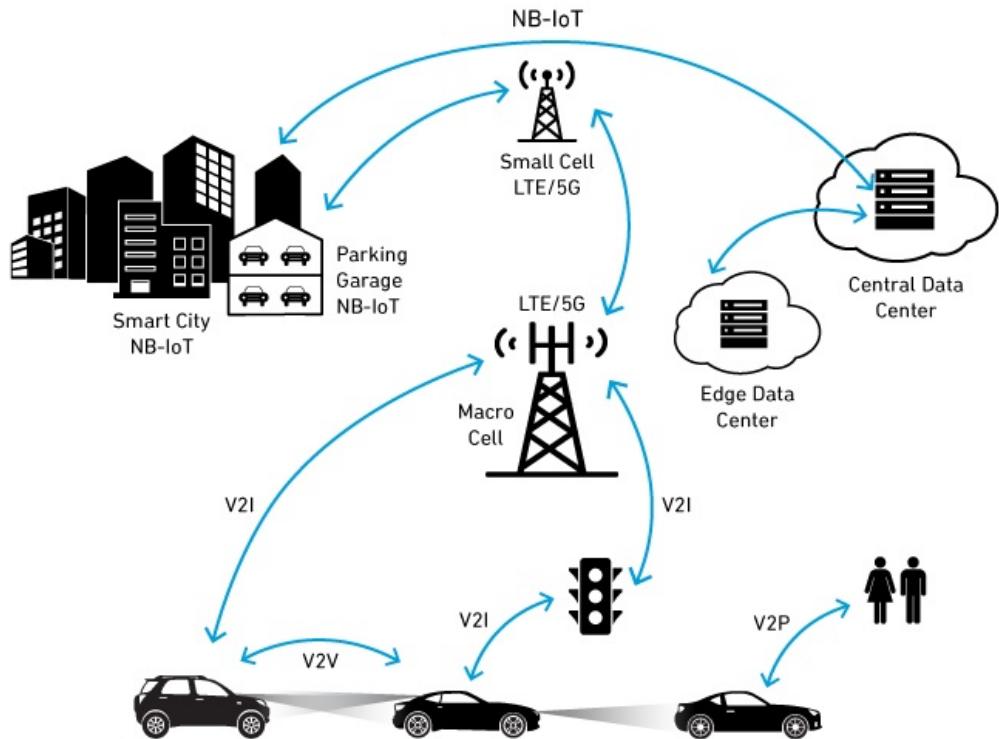
## **System Supervisor**

Das letzte Modul ist der Systembetreuer (engl. system supervisor). Der Systembetreuer[19] (s. Abbildung 2.6) überwacht kontinuierlich das gesamte Softwarepaket und die Hardware-Ausgabe, um eine geeignete Fahrfunktion sicherzustellen. Der Systembetreuer prüft Software- und Hardwarefunktionen auf Fehlern und gibt ein Feedback aus.

## **2.7 V2X**

V2X[20] ist eine Form der drahtlosen Kommunikation, bei der Daten von einem Fahrzeug zu einer Einheit oder umgekehrt übertragen werden. Die folgende Abbildung 2.7 zeigt eine Übersicht des V2X Kommunikationssystems.

## C-V2X Communications



QORVO

©2018 Qorvo, Inc.

Abbildung 2.7: V2X Architektur[21]

Wie in der Abbildung 2.7 gezeigt sind mehrere Arten dieser Kommunikation vorhanden[21]:

- Vehicle-to-infrastructure (V2i): zwischen Fahrzeuge und Infrastruktur (z.b. Ampel) um dynamische Verkehrssignalisierung zu ermöglichen.
- Vehicle-to-vehicle (V2v): zwischen Fahrzeuge um Kollision zu vermeiden.
- Vehicle-to-pedestrian (V2p): zwischen Fahrzeuge und Fußgänger um Sicherheitswarnungen zu übermitteln.
- Vehicle-to-network (v2n): zwischen Fahrzeuge und Netzwerk (z.b. LTE/5G) um Echtzeit-Verkehr, Wetter und andere Cloud-Dienste zu ermitteln[21]

V2X[20] wurde ursprünglich entwickelt, um die Verkehrssicherheit zu verbessern und die Anzahl der Unfälle zu verringern. Studien[22] haben gezeigt, dass es neben der Verkehrssicherheit auch weitere Vorteile bietet, wie z.B. das Verkehrsflussmanagement und die Stauvermeidung. Außerdem wird damit die Energie eingespart und Umweltverschmutzung reduziert. Es kann auch zum Aufbau intelligenter Verkehrssysteme beitragen und die Entwicklung neuer Formen von Autos und Transportdiensten fördern[22, 23].

# Kapitel 3

# Sicherheitsstandards

In diesem Kapitel werden Sicherheitskonzepte, Normen und Standards für das autonome Fahren gemäß[24] definiert.

## 3.1 Unfälle

Laut DMV<sup>1</sup>[25] sind seit 2014 mehrere Testfahrten mit AFs nicht erfolgreich beendet worden. Einige AFs sind mit anderen Fahrzeugen zusammengestoßen, mit Fahrradfahrern kollidiert oder andere unvorhersehbare Ereignisse sind eingetreten. Diese Unfallursachen werden in einer Datenbank festgehalten, die bei der Unfallvermeidung sehr hilfreich sein wird[25]. Im Jahr 2016 wurde der Fahrer eines Tesla[26] „Model S“ in Hanadan, China bei einem Autounfall sofort getötet. Das Auto war im Autopilot Modus und hat nicht gebremst, bevor es in einem gestoppten LKW kollidierte. Die Abbildung 3.1 zeigt die Szene direkt nach dem Unfall.



Abbildung 3.1: Tesla S Autopilot Mode, fataler Unfall in China [26]

---

<sup>1</sup>Department of motor vehicles

Die NTSB<sup>2</sup>[27] berichtete, dass in Williston Florida ein ähnlich fataler Unfall mit dem gleichen Tesla S Modell im Autopilot Mode passiert ist. Der Fahrer ist ums Leben gekommen, als das Auto mit einem großen LKW kollidierte. Die Abbildung 3.2 zeigt den Zustand des Autos nach dem Unfall[28].



Abbildung 3.2: Tesla S Autopilot Mode, Ergebnis des fatalen Unfalls in Florida [28]

Im gleichen Jahr hat Waymo<sup>3</sup>[29] einen Unfall mit einem Bus gehabt. Glücklicherweise ist niemand dadurch betroffen. Das Auto ist in der Abbildung 3.3.

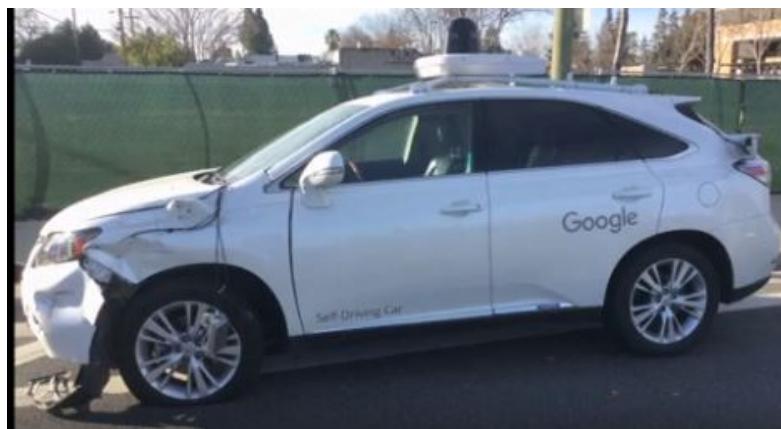


Abbildung 3.3: Google Auto Waymo Unfall [29]

Ein Jahr später hatte Cruise Chevy Bolt der Firma GM<sup>4</sup>[30] einen Unfall

---

<sup>2</sup>National Transportation Safety Board

<sup>3</sup>Google Auto

<sup>4</sup>General Motors

mit einem Motorrad, wodurch der Motorradfahrer schwer verletzt wurde. Ein autonomes Fahrzeug (Volvo XC90) der Firma Uber[31] hat einen Fußgänger in Tempe, Maricopa County, Arizona getötet. Die Abbildung 3.4 zeigt den Standort und Zustand des Autos.



Abbildung 3.4: Uber Auto Unfall [31]

Diese Unfälle sind traurige Beispiele und weisen darauf hin, dass die Absicherung AFs unersetzlich ist. Die Fahraufgabe kann auf allen Ebenen (Wahrnehmung, Planung und Kontrolle) unvorhersehbare Fehler aufweisen, die Menschenleben kosten können. Daher müssen Sicherheitsmaßnahmen und -Konzepte von Industrie und Regierung geregelt werden[24].

## 3.2 Sicherheitsbegriffe

In diesem Abschnitt sind grundlegende Sicherheitsbegriffe sowie Sicherheitsframeworks gemäß[24] definiert. Der Begriff Schaden wird verwendet, um physische Schäden an Lebewesen zu beschreiben. Der Begriff Risiko kombiniert die Wahrscheinlichkeit, dass ein Ereignis auftritt mit dem Schweregrad des Schadens, verursacht durch dieses Ereignis[24]. Sicherheit (engl. Safety), nicht zu verwechseln mit dem Sicherheitsbegriff des Schutzes (engl. Security) ist der Prozess der Vermeidung unangemessener Schadenrisiken eines Lebewesens. Der Begriff Gefahr (engl. hazard) ist die Anwesenheit einer potentiellen Quelle von unangemessenen Schadenrisiken, die eine Bedrohung der Sicherheit darstellt[24]. Zum Beispiel wenn ein Software Fehler zu einem Unfall führt, wird dieser Fehler als Gefahr bezeichnet. Eine Gefahr in AFs kann mechanisch, elektrisch oder softwarebezogen sein[24].

### **3.3 ISO 26262**

Funktionale Sicherheit[32], abgekürzt FuSa<sup>5</sup> nach ISO 26262[33], definiert die Fahrzeugsicherheit in Abwesenheit unangemessener Risikos durch die Hard- und Software Designfehlern oder Fehlverhalten in elektrischen und elektronischen Autos[33]. Es erfordert eine Gefahrenanalyse und eine Risikobewertung (HARA<sup>6</sup>) zur Bestimmung des Gefährdungsgrads des Fahrzeugs[34]. HARA identifiziert Fehler mittels FMEA<sup>7</sup>[35] und HAZOP<sup>8</sup>[36] Techniken[37]. ISO 26262 ist in zehn Teile gegliedert[38]. Eine detaillierte Beschreibung aller Teile ist hiermit nicht zielführend sondern eine Zusammenfassung wichtiger Aspekte in Bezug auf die AF-Sicherheit. ISO 26262 folgt einem V-Modell[37] für die SW<sup>9</sup>- und HW<sup>10</sup>-Entwicklungsphase (siehe Abbildung3.5). Das Prinzip der FuSa ist die Identifikation der Worst-Case-Gefahr, dann die Erstellung der Worst-Case-Anforderungen und dessen Implementierung in HW und/oder SW, damit ein System mindestens diese Worst-Case-Anforderungen erfüllen kann[37].

### **3.4 ISO/PAS 21448.1**

Die Sicherheit der beabsichtigten Funktionalität abgekürzt SOTIF<sup>11</sup> gemäß[37] befasst sich mit Fehlerursachen im Zusammenhang mit Systemeinschränkungen und vorhersehbarem Missbrauch des Systems. Die Einschränkungen können Sensorungenauigkeit, ML<sup>12</sup>-Algorithmen einschränkungen oder Akutikeinschränkungen sein. Die SOTIF in ISO/PAS 21448[39] ist eine Erweiterung der FuSa, die zusätzlich HARA-Einschränkungen der ISO26262 berücksichtigen soll[37]. Ein Vergleich zwischen ISO26262(Grün) und ISO/PAS 21448(Blau) des SW-Entwicklungsprozesses ist in der Abbildung3.5 zu sehen.

---

<sup>5</sup>Functional Safety

<sup>6</sup>Hazard Analysis and Risk Assessment

<sup>7</sup>Failure mode and effects analysis

<sup>8</sup>Hazard and operability study

<sup>9</sup>Software

<sup>10</sup>Hardware

<sup>11</sup>Safety of Intended Functionality

<sup>12</sup>Maschinelles Lernen

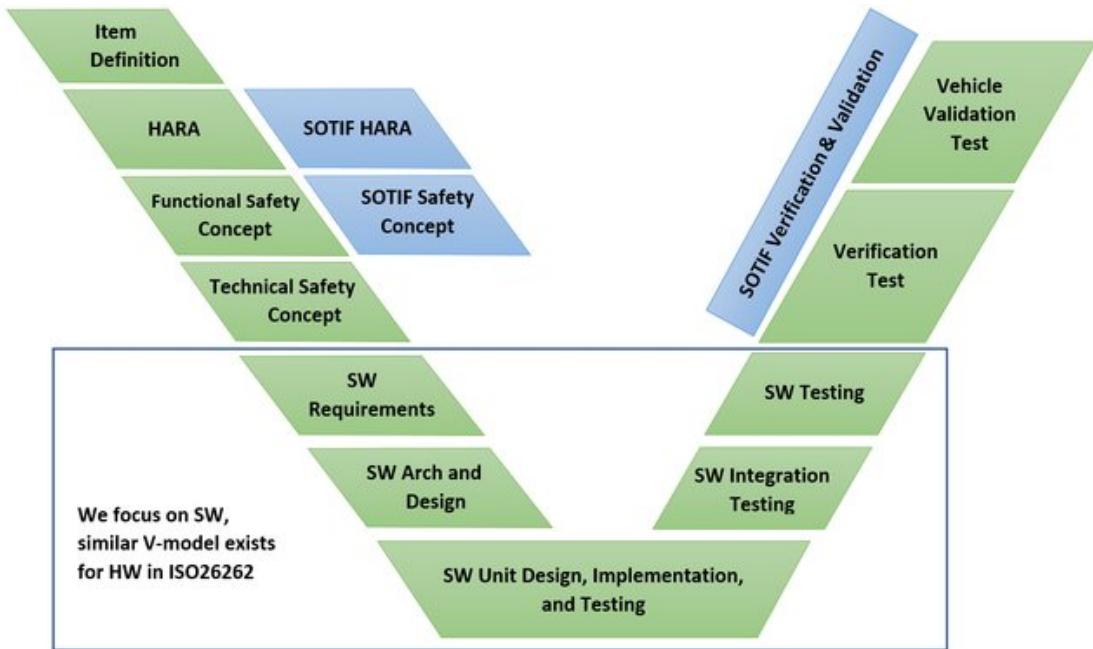


Abbildung 3.5: Vergleich zwischen ISO26262(Grün) und ISO/PAS 21448(Blau) des SW-Entwicklungsprozesses [34]

### 3.5 NHTSA

Die NHTSA<sup>13</sup>[40] ist eine Agentur der US-Bundesregierung, die sich mit der Verkehrssicherheit beschäftigt. Die NHTSA hat über Jahre mehrere Sicherheits-Frameworks veröffentlicht. Die NHTSA verlangt von jedem autonomen Fahrrentwickler, eine umfassende Sicherheitsstrategie zu entwickeln und zu beschreiben, die die 12 in seinem Leitfaden enthaltenen Konzepte abdeckt[41]. Der Framework lautet:

- Automated Driving Systems 2.0.: A Vision for Safety (2017, [42])

Ein weiteres Framework zur Erstellung von Testszenarien wurde publiziert. Das folgende Framework wird im Rahmen dieser Arbeit zusammengefasst:

- A Framework for Automated Driving System Testable Cases and Scenarios (2018, [43])

Das Framework ist an sich kein Standard sondern eine Empfehlung, die jeder Entwickler befolgen sollte. Der Testszyklus eines ADS<sup>14</sup> gemäß NHTSA[43] hat drei Zyklen (s. Abbildung 3.6):

<sup>13</sup>National Highway Transportation Safety Administration

<sup>14</sup>Automated Driving System

- Modellierung und Simulation (M&S)
- Tests auf geschlossener Strecke
- Tests auf offener Straße

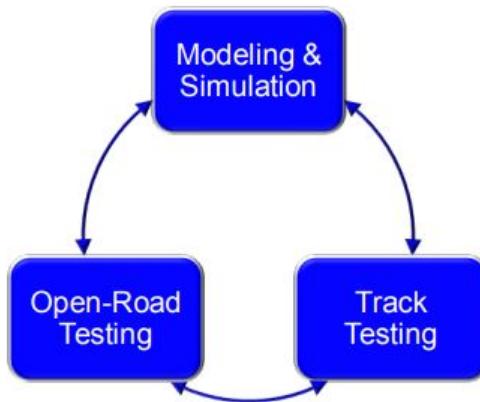


Abbildung 3.6: Konventioneller ADS-Testzyklus [43]

Modellierung und Simulation basieren auf einer virtuellen Umgebung mit virtuellen Agenten, um Wissen über ADS zu generieren. Die Fahrzeuge und deren Komponenten müssen physikalisch und/oder mathematisch modelliert werden bis, das virtuelle Fahrzeug ein reales Fahrzeugverhalten imitieren kann[43]. Je höher der Treuegrad<sup>15</sup> dieser Modelle, desto genauer können die tatsächlichen Eigenschaften des Fahrzeugs bzw. der Umgebung wiedergegeben und umso aussagekräftigere Daten für die Analyse können erfasst werden. Mehrere Unterfelder innerhalb des Bereichs M&S wurden identifiziert[43]:

- Software-in-the-loop (SIL) simulation: Software bezogen Simulation
- Hardware-in-the-loop (HIL) simulation: Hardware bezogen Simulation
- Vehicle-in-the-loop (VIL) simulation: Fahrzeug bezogen Simulation

Das Framework definiert ein Testszenario als eine vierdimensionale Testmatrix (s. Abbildung 3.7)[43]:

- Verhalten bei taktischen Manövern
- ODD-Elemente
- OEDR-Verhalten

---

<sup>15</sup>degree of fidelity

- Ausfallmodus-Verhalten

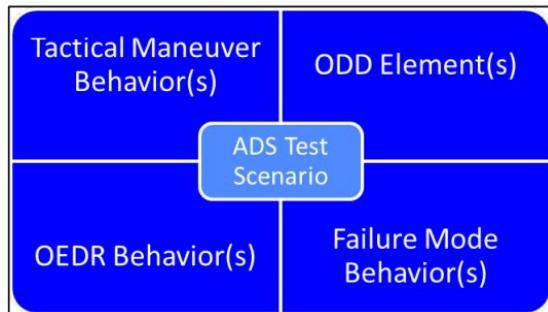


Abbildung 3.7: Testszenario Matrix [43]

Jede Dimension kann als eine Art Checkliste gesehen werden. Das erste Hauptelement ist das Verhalten bei taktischen Manövern. Es befindet sich innerhalb einer Hierarchie (s. Abbildung 7.2), die auf Grundlage der Dauer des taktischen Verhaltens auf mehreren Ebenen aufgeteilt ist[43]. Die taktischen und operationalen Manöver der L4<sup>16</sup> nach OEMs<sup>17</sup> sind in der Tabelle 7.4 zu finden[43]. Das zweite Hauptelement ist ODD. Die NHTSA hat ODD in Klassen unterteilt. Die Abbildung 7.3 zeigt eine Übersicht der ODD Klassen und dessen Unterklassen. Das dritte Hauptelement ist OEDR. Laut NHTSA[43] hat OEDR 19 Elemente, sie sind in der Tabelle 7.5 dargestellt.

Das vierte Hauptelement ist das Ausfall-Verhalten. Das Ausfall-Verhalten basiert auf FO und FS Mechanismen. FO bzw. Fail-Operational bedeutet operativ ausfallen, wenn ein Fehler auftritt, kann das Fahrzeug für eine begrenzte Zeit weiter funktionieren[43]. FS bzw. Fail-Safe bedeutet sicher ausfallen, wenn ein kritischer Fehler auftritt, kann das Fahrzeug nicht mehr wie vorgesehen funktionieren[43]. Diese Mechanismen ermöglichen es einem ADS einen MRC<sup>18</sup> zu erreichen, der das Fahrzeug und seine Insassen soweit wie nötig von der Gefahr wegbringt. Zwei Testmethoden sind im Framework[43] festgelegt:

- Black-Box-Test: Die Funktionalität des Systems wird getestet, während das interne Design und die Implementierung des Systems dem Tester weitgehend unbekannt sind[43].
- White-Box-Test: Die interne Struktur und die Funktionen eines Systems werden getestet, während das interne Design und die Implementierung des Systems dem Tester bekannt sind. [43] .

<sup>16</sup>Level 4

<sup>17</sup>Original Equipment Manufacturer, hier als Automobilhersteller bezeichnet

<sup>18</sup>minimal risk condition

In einigen Fällen können Black-Box-Tests für Sicherheitsüberprüfungen von ADS ausreichend sein, jedoch geben die mit dem Whitebox-Test verbundenen Fragen ein besseres Verständnis der Leistungsgrenzen eines Systems[43] aus.

Das taktische Manöververhalten bezieht sich auf die unmittelbaren Steuerungsaufgaben, die das ADS ausführt (z.B. Spurverfolgung, Spurwechsel, Abbiegen, etc..)[43]. ODD-Elemente definieren im Allgemeinen die Betriebsumgebung in der das ADS während des Tests navigieren soll (z.B Fahrbahntyp, Verkehrsbedingungen oder Umweltbedingungen)[43]. OEDR-Fähigkeiten betreffen direkt die Objekte und Ereignisse, auf die das ADS während des Tests trifft (z.B. Fahrzeuge, Fußgänger, Verkehrssignale, etc. )[43]. Das Ausfallmodus-Verhalten kann durch Injektion von Fehlern oder einer Simulation von Ausfällen getestet werden[43].

Testszenarien können aus der Kombination der vierdimensionalen Testmatrix erstellt werden. Beispiele von Testszenarienprotokollen wurden im NHTSA Framework beschrieben. Ein Beispiel vom Szenarioprotooll wurde aus dem Framework importiert und befindet sich im Anhang NHTSA 7.1. Dieses Beispiel wird hier analysiert. Das Szenario betrachtet den Fall des Spurwechsels und langsamen in dem Verkehrsflusses einordnen bzw. einfließen<sup>19</sup> (s. Abbildung7.4). Das Protokoll enthält Informationen über ODD, OEDR, Ego-Fahrzeug, andere Fahrzeuge, Personal, Ausrüstung und weitere Daten, die die Szenarioumgebung definieren[43]. Dieses Szenario enthält mehrere Kombinationen von Szenarienmanövern mit Variationen der SV<sup>20</sup> und POV<sup>21</sup> Geschwindigkeit und Position (s. Tabelle7.6).

Die Szenarien haben eine konventionelle Schreibweise. Beispielsweise Szenario (Baseline 15: PLC\_B\_15) bedeutet folgendes[43]:

- PLC: Abkürzung des Szenarios (Perform Lane Change)
- B: Baseline ODD für dieses Szenario
- 15: Die Geschwindigkeit der SV in mph<sup>22</sup>

Dieses Protokoll ist ein Muster für die Erstellung allgemeiner Testszenarien. Die NHTSA hat 28 Verhaltenskompetenzen empfohlen (s. Tabelle7.1), die als Grundlage der Testszenarienerstellung[43] dienen sollen. Waymo hat die Liste auf 47 Verhaltenskompetenzen (s. Tabelle7.2) erweitert[5]. NHTSA hat die Verkehrsunfälle in vier Kategorien zusammengefasst. 28 Unfallszenarien

---

<sup>19</sup>Lane change and slow merge

<sup>20</sup>Subject Vehicle: Das Ego-Fahrzeug

<sup>21</sup>Principal other vehicle: andere Fahrzeuge

<sup>22</sup>mile per hour

sind der Ursprung aller Verkehrsunfälle (s. Tabelle7.3)[43].

Die Methodik zur Erstellung und Auswertung von Testszenarien ist im NHTSA-Framework ausführlich definiert. Entwickler können die vorgeschlagenen Verhaltenskompetenzen und Muster-Protokolle aus dem Framework als Leitfaden für die Weiterentwicklung von Testszenarien verwenden. Wissenschaftler und Entwickler streben ebenfalls an, eine öffentliche Datenbank mit allen möglichen Testszenarien in ein einheitliches Format zu bringen. Dieser Ansatz kann dabei helfen alle Szenarien übersichtlich abrufen und testen zu können. PEGASUS[10] Framework hat diesen Ansatz beschrieben. Im nächsten Abschnitt wird die PEGASUS Methode erklärt.

## 3.6 PEGASUS

Das PEGASUS<sup>23</sup> Projekt gemäß[10] ist ein Verfahren um automatisierte Fahr-funktionen zu testen mit dem Ziel, eine schnelle Umsetzung des automatisierten Fahrens in die Praxis zu ermöglichen[10]. Die PEGASUS-Methode enthält 20 Etappen. Die Abbildung7.10 stellt die 20 Etappen dar. Im folgendem werden diese Schritte mit Bezug auf die Nummern (0-20) in der Abbildung7.10 sinngemäß[10] zusammengefasst:

Die Daten wurden aus verschiedenen Quellen (0) zusammengesammelt. Kenntnisse und Expertise der Normen und Standards (1) definieren die Anforderungsanalysen (3), die als Metrik (6) für die Evaluierung gilt. Aus (1) können systematisch Szenarien identifiziert werden. Verkehrsdaten (2) aus FOT<sup>24</sup>, NDS<sup>25</sup> und Autounfalldatenbanken werden bearbeitet (5) um daraus Szenarien zu extrahieren (s. Abbildung7.11).

Ein Szenario nach PEGASUS besteht aus 6 Schichten (s. Abbildung7.12). Die Daten werden ins PEGASUS-Format konvertiert (7). Metriken zur Identifizierung der Szenarien wurden erforscht (8). Logische Szenarien (9) enthalten keine konkrete Beschreibung eines Verkehrsszenarios und keine festgelegten Parameter für Distanz und Geschwindigkeit[10].

Mittels des „Test Automation Tool“ (s. Abbildung7.13) werden Testfälle mit festen Parametern erstellt. Dabei wurde eine stochastische Variation angewendet um das Parameterspektrum zu erkunden. Für die Beschreibung

---

<sup>23</sup>Project for the Establishment of Generally Accepted quality criteria, tools and methods as well as Scenarios and Situations on the release of highly-automated driving functions

<sup>24</sup>Field Operation Testing

<sup>25</sup>naturalistic driving study

von konkreten Szenarien sind zwei Formate notwendig: OpenDrive[44] beschreibt die statische Umgebung. OpenSCENARIO[45] die dynamischen Elemente.

Anhand von Transpiler und OpenDrive Generator (s. Abbildung7.13) werden konkrete Szenarien in standardisiertem Format generiert. Testszenarien werden via „Simulation Tools“ (s. Abbildung7.14) getestet. Simulationsergebnisse werden anhand von variierten Pass/Fail Kriterien ausgewertet. Das Testresultat steht letztendlich bereit[10].

Einige Schritte der gesamten Methode wurden im PEGASUS Framework nicht beschrieben bzw. waren nicht verfügbar und wurden in dieser Arbeit deshalb nicht erwähnt. Die PEGASUS-Methode basiert auf einer Datensammlung und ermöglicht dadurch eine automatische Erstellung und Auswertung von Testszenarien.

# **Kapitel 4**

# **Implementierung**

In diesem Kapitel wird das ADORe Framework mit seinen Bestandteilen und Tools beschrieben. Das Konzept zur Erstellung von Testszenarien wird im ADORe implementiert.

## **4.1 ADORe Framework**

The Automated Driving Open Research (ADORe) gemäß[8] ist eine modulare Softwarebibliothek und Toolkit für die Entscheidungsfindung, Planung, Steuerung und Simulation von automatisierten Fahrzeugen. Bis zur Open-Source-Veröffentlichung als Eclipse-Projekt wurde es ausschließlich im Deutschen Zentrum für Luft- und Raumfahrt (DLR) entwickelt.

Die Kernkomponenten von Eclipse ADORe sind gemäß[46] in folgende Bibliotheken definiert:

- libadore/env: Umgebungsmodelle, die für das automatisierte Fahren verwendet werden
- libadore/view: Schnittstellenbibliothek zur Abstraktion von Umgebungsmodellen, die erlaubt, darstellungsunabhängige und aufgabenspezifische Informationen an Planungs- und Steuerungsmodulen der Fahrzeugautomatisierung zu liefern
- libadore/fun: Bewegungsplan- und Steuerungsmodule für automatisierte Fahrzeuge
- libadore/sim: Notwendige Simulationsmodule für Fahrzeugbewegung, Wahrnehmung und Kommunikation zwischen den Modulen
- libadore/apps: Systemunabhängige Anwendungen für automatisiertes Fahren, zusammengestellt aus den oben genannten Modulen

- `adore_if_ros`: Schnittstelle zu ROS: ROS-spezifische Realisierung von libadore/apps. Enthält einen Satz von ROS-Startdatei-Simulationsbeispielen.
- `adore_if_ros_msg`: Sammlung von ROS-Nachrichten für automatisiertes Fahren und deren Konvertierung
- `sumo_if_ros`: Eine Brücke zwischen Eclipse SUMO und ROS, die die Kopplung von `adore_if_ros` mit SUMO zur Co-Simulation von automatisierten Fahrzeugen im Stadtverkehr ermöglicht.
- `Plotlabserver` und `plotlablib`: 3D-Datenvisualisierung mit Matlab und Python-matplotlib-Export

Die Kernfunktionalität von Eclipse ADORe ist in systemunabhängigem C++ entwickelt worden. Es wird mit einer Build-Chain für gcc, cmake und catkin (ROS kinetic-spezifisch) sowie passenden Bibliotheksabhängigkeiten und Anleitungen für Ubuntu (18.04) geliefert[46].

ADORe ist ein Open-Source Projekt und ist unter der EPL<sup>1</sup> 2.0 lizenziert[46]. Das Projekt wurde auf der Github Plattform gehostet. Zur Verwendung oder Mitentwicklung kann das Projekt unter dem Link[47] gefunden werden.

#### 4.1.1 ROS

Robot Oerating System(ROS) nach[48] ist eine Middleware für Robotikentwicklung, welche Bibliotheken und Werkzeuge bietet, die Softwareentwickler bei der Erstellung von Roboteranwendungen unterstützen. Es bietet Hardware-Abstraktion, Gerätetreiber, Bibliotheken, Visualisierungen, Message-Passing, Paketverwaltung und mehr. ROS ist unter einer Open-Source, BSD<sup>2</sup>-Lizenz lizenziert[48]. ROS wurde im ADORe Framework genutzt, um eine unabhängige Software Entwicklung zu ermöglichen. Das Konzept heißt ROS-wrapper[49] und wurde im ADORe verwendet um die Portabilität des Frameworks zu gewährleisten. Die Abbildung4.1 zeigt das Konzept des Frameworks:

---

<sup>1</sup>Eclipse Public License

<sup>2</sup>Berkeley Software Distribution

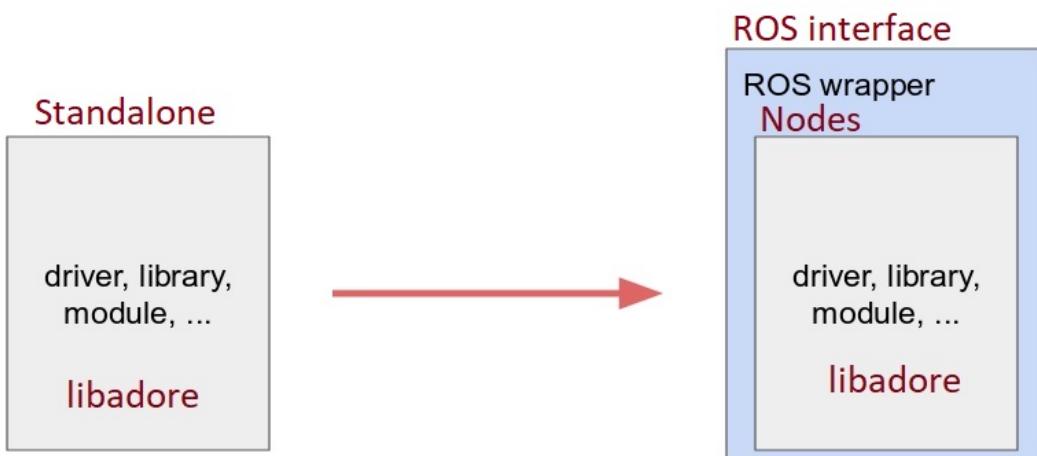


Abbildung 4.1: ROS wrapper [49]

Zudem hilft ROS dabei die Komponenten des automatisierten Fahrzeugs wie z.B. die Sensorik, Aktorik und den Hauptprozessor über ROS-definierte Nachrichten, miteinander zu verbinden. `adore_if_ros` ist die Schnittstelle zur ROS. ROS-Nodes<sup>3</sup> werden hiermit verwendet um in libadore definierte Anwendungen auszuführen und miteinander zu verbinden. `adore_if_ros_msg` ist das unabhängige Paket für Nachrichtenaustausch und Datenkonvertierung zwischen ADORe und ROS.

Ein ROS Paket enthält in der Regel Nodes, ein Master Node mit seinen Parameter Servern, Topics<sup>4</sup>, Nachrichten, Services und Bags. Ein definiertes Kommunikationsprotokoll ermöglicht die Kommunikation zwischen Nodes und Master, sowie direkt zwischen den Nodes. Nodes können Master Topics abonnieren oder Nachrichten in bestimmten Topics publizieren[50]. Die Abbildung 4.2 zeigt ein Beispiel über das Kommunikationskonzept.

Um eine detaillierte Beschreibung über ROS und Starthilfe im Umgang damit zu erhalten, ist die ROS Wiki-Seite[50] sehr hilfreich.

---

<sup>3</sup>Knoten

<sup>4</sup>Themen

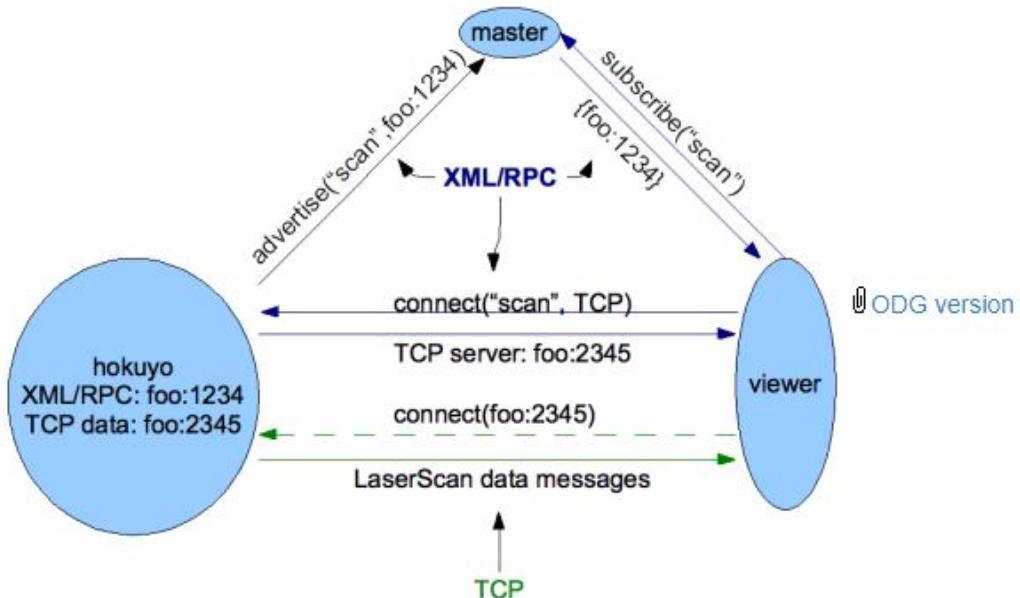


Abbildung 4.2: ROS Kommunikation [50]

#### 4.1.2 SUMO

Simulation of Urban Mobility(SUMO) nach[51] ist eine Open-Source Verkehrssimulationssoftware, die sich für die Erstellung und Bearbeitung von Netzwerken, Karten, Szenarien und Testfällen für zahlreiche Verkehrssituationen eignet[51]. Es wird hauptsächlich von Mitarbeitern des Instituts für Verkehrssysteme am Deutschen Zentrum für Luft- und Raumfahrt (DLR) entwickelt. SUMO ist unter der EPL<sup>5</sup> 2.0 lizenziert[51]. Mit SUMO können Netzwerke manuell gezeichnet oder Karten aus verschiedenen Quellen wie OpenStreetMap (OSM)[52] oder OpenDrive[44] importiert werden.

Die Abbildung 4.3 zeigt ein Beispiel des SUMO-Netzwerks.

---

<sup>5</sup>Eclipse Public License

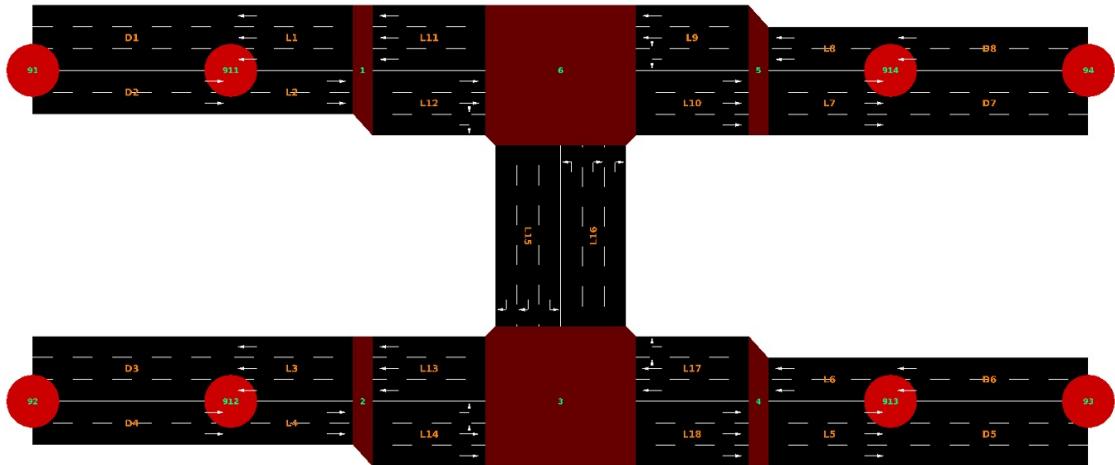


Abbildung 4.3: Beispiel Netzwerk Sumo [53]

Die statischen Elemente der Karte werden bearbeitet und angepasst, dann werden dynamische Elemente (Fahrzeuge, Fußgänger usw.) hinzugefügt. Ein gewünschtes Verkehrsszenario wird erstellt und kann letztendlich simuliert werden. Die erstellten Daten können zu einem bestimmten Format exportiert bzw. konvertiert werden.

Für diesen Arbeitszweck werden Daten im SUMO .net.xml Format gespeichert und mit dem SUMO-Werkzeug „netconvert“ ins OpenDrive Format (.xodr) konvertiert. Nur OpenDrive und Road2Simulation[54] Karten können zurzeit in ADORe angewendet werden, die Funktionalität wird in der Zukunft wohl noch erweitert werden. SUMO-Dynamische Elemente, die sogenannten „traffic demands“ werden in separaten Dateien im „rou.xml“ Format gespeichert. Um eine Simulation zu erstellen, muss eine „sumocfg“ Datei erstellt werden, die „.net.xml“ (Sumo-Karte) und „.rou.xml“ (Verkehrselemente) Dateien enthält. Die Abbildung 4.4 zeigt ein Beispiel einer Simulation.

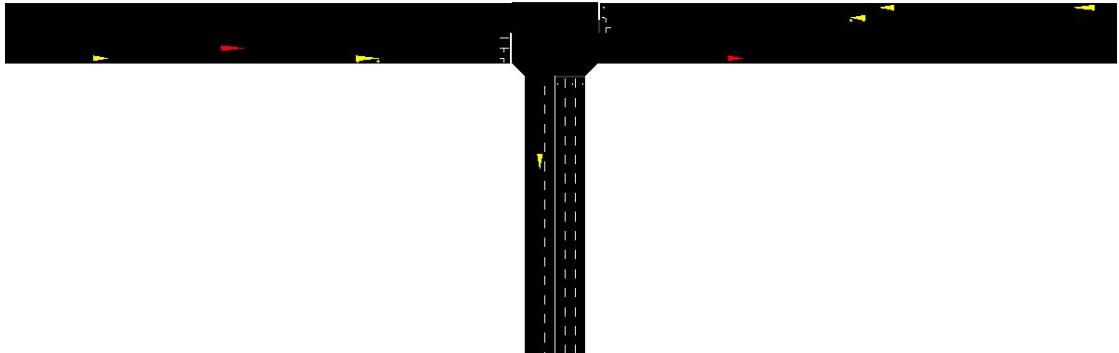


Abbildung 4.4: Beispiel simulation sumo [53]

Das SUMO Traffic Control Interface (TraCI) ermöglicht die Interaktion mit einer laufenden SUMO-Simulation, Werte von simulierten Objekten werden abgerufen und deren Verhalten kann beobachtet und bearbeitet werden[51]. Diese Schnittstelle wurde vom ADORe so implementiert, um Ego-Fahrzeuge mit SUMO-Fahrzeugen zu verbinden. Zuständig für diesen Zweck ist das Paket `sumo_if_ros`.

#### 4.1.3 Gitlab

Gitlab gemäß[55] ist eine Open-Source DevOps<sup>6</sup> Plattform mit integrierter Versionskontrolle (Git) zum Zweck der Fehlerverfolgung, Codeüberprüfung und CI<sup>7</sup>/CD<sup>8</sup>. ADORe wird intern über Gitlab verwaltet, um die Softwareentwicklung leicht, kontrollierbar und gesichert zu ermöglichen. Die ADORe Open-Source Version wird auf der Github Plattform gehostet. Die ADORe Github Version ist eine stabile Version. Die ADORe Gitlab Version ist ständig in Entwicklung.

Gitlab bietet insbesondere den CI/CD Service, der kontinuierliche Methodologie übermittelt. Diese Methodologie fördert die Automatisierung der Ausführung von Skripten und Tests, um die Wahrscheinlichkeit von Fehlern bei der Entwicklung von Anwendungen zu minimieren oder Fehler automatisch zu erkennen[56]. Für weitere Infos über Gitlab CI/CD ist die Gitlab Dokumentation abrufbar[56]. Der Einsatz von CI/CD wird in den nächsten Schritten beschrieben.

---

<sup>6</sup>Development and Operations

<sup>7</sup>Continuous Integration

<sup>8</sup>Continuous development

#### 4.1.4 Docker

Docker gemäß[57] ist eine Serviceplattform, die mithilfe der Virtualisierung auf Betriebssystemebene Software in Paketen bereitstellt, die als „Container“ bezeichnet werden. Container und VM<sup>9</sup> haben ähnliche Vorteile bei der Ressourcenisolierung und Ressourcenzuweisung. Sie funktionieren aber anders, da die Container das Betriebssystem und nicht die Hardware virtualisieren[57]. Container sind deswegen portabler und effizienter. In der Docker-Dokumentation[58] wird die Anwendung erklärt. Docker-Hub[59] ist eine Plattform, die Docker-Images online bereitstellt. Der Einsatz der Docker wird in den nächsten Schritten beschrieben.

## 4.2 ADORe Stand & Demos

Die Adore Github Version (siehe Link[47]) hat eigene Funktionalitäten des "System under test" (sut) bereitgestellt. Die Demonstration der Automatisierung bestimmter Fahrfunktionen bzw. bestimmter Features des Frameworks befindet sich in `adore_if_ros_demos` (unter dem Link[60] abrufbar).

Nach erfolgreicher Installation des ADORe Frameworks auf einem Rechner oder auf einer virtuellen Maschine mit mindestens Ubuntu 18.04 (siehe Installation mit folgendem Link[61]), kann die Simulation gestartet werden. Als Beispiel zeigt die "demo003\_lanefollowing.launch" ein Fahrspurfolgeverhalten. Mit folgendem „bash“ Befehl wird die demo003 gestartet:

```
1 cd ~/catkin_ws/src/adore/plotlabserver/;./stop.sh;./start.sh;roslaunch ~/catkin_ws/src/adore/adore_if_ros_demos/demo003_lanefollowing.launch
```

Listing 4.1: start demo003

Zuerst muss der Plotlabserver-Ordner aufgerufen werden. Dann folgt der Start des Plotten-Skripts und schließlich die Ausführung der Launch-Datei. Die Abbildung 4.5 zeigt einen Überblick dieses Verhaltens.

<sup>9</sup>virtuelle Maschinen, z.b. via Virtualbox

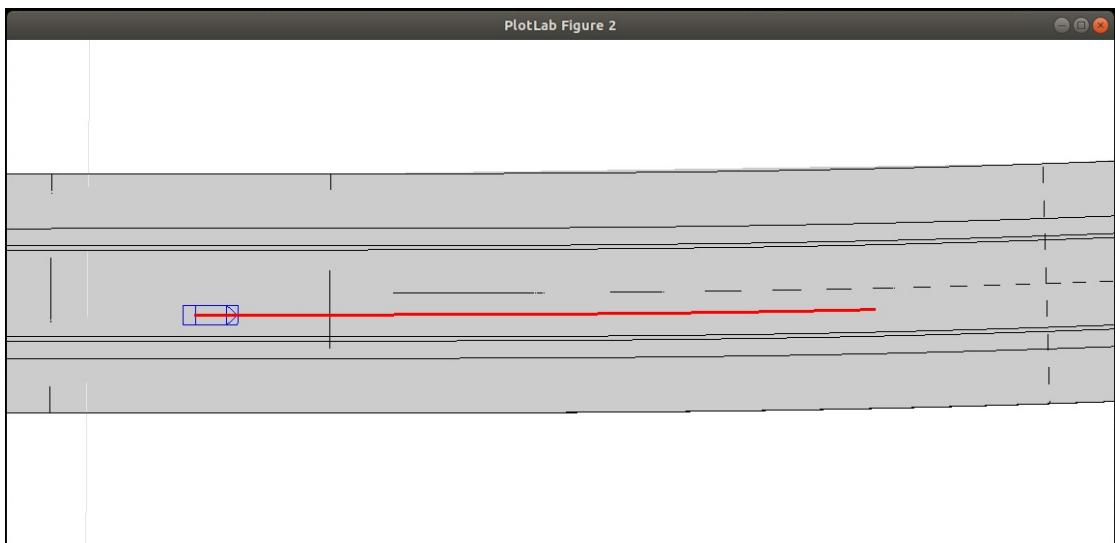


Abbildung 4.5: lane following [60]

In der Abbildung werden das Ego (blauer Kasten), die Referenztrajektorie (rot) und der Straßenabschnitt (grau) dargestellt.

### 4.3 Lösungsansatz

Frameworks zur Erstellung und Auswertung von Testszenarien werden im letzten Kapitel zusammengefasst. Das NHTSA-Framework kann als Konzept für die Erstellung grundlegender Szenarien verwendet werden. Die Wahl der NHTSA-Methodik liegt darin, dass dieser Ansatz mit den verfügbaren Ressourcen realisierbar und die Implementierung im ADORe-Framework möglich ist. Das PEGASUS-Framework kann als Konzept für die automatische Ausführung und Auswertung von Szenarien verwendet werden. Die NHTSA hat eine Vergleichstabelle (s. Tabelle 7.5-7.8) erstellt, um Testszenarien basierend auf der vierdimensionalen Matrix zu ermitteln. Die Tabelle(7.5-7.8) vergleicht NHTSA mit anderen Frameworks.

Die Tabelle(7.5-7.8) dient als Leitfaden für die Ableitung von Testszenarien. Der ADORe Szenarienkatalog wird zu dieser Tabelle7.7 hinzugefügt, um ADORe Framework mit anderen Frameworks vergleichen zu können. Zudem kann diese Tabelle7.7 als Referenz für die minimal akzeptierten Verhaltenskompetenz verwendet werden. Waymo hat die Tabelle (7.5-7.8) erweitert, um komplexere Testfälle zu generieren. Die Methodik der Szenarienerstellung wird entsprechend der NHTSA umstrukturiert.

Die Tabelle7.7 listet die Kategorien zu taktischen Manövern (Parken, Spurver-

folgung, Spurwechsel, Kreuzungsnavigation), sowie die erstellten Szenarien vom NHTSA, Waymo und ADORe auf. Pro taktisches Manöver wird mindestens ein Testfall erstellt. Beispielsweise, für das Spurfolgsmanöver wird entsprechend ein ADORe Szenario erstellt und als „test002\_lane\_following.launch“ definiert (s. Tabelle7.7). Analog dazu werden weitere Testszenarien erstellt. Zum Zeitpunkt der Verfassung dieser Arbeit konnten nicht alle Kategorien abgedeckt werden, da einige Funktionalitäten im ADORe Framework noch in Entwicklung waren bzw. sind.

Die Tabelle7.7 beinhaltet pro Kategorie bzw. Testszenario das Testverfahren und die Zielbeschreibung, den Status, die Evaluationsmetriken und die Parametervariation. Ziel und Ablauf je Szenario wird in diesen Kapitel später noch beschrieben. Der Teststatus wird wie folgt dargestellt:

- 0: nicht erstellt
- 1: erstellt
- 2: getestet
- 3: automatisiert
- 4: ausgewertet

Bewertungsmetriken werden in 3 Kategorien klassifiziert:

- 0: Simulation
- 1: abgeschlossenes Testgelände
- 2: öffentliche Straße

Metriken können je nach Kategorie, Testverfahren und Ziel allgemein oder spezifisch sein, d.h. die Bewertungsmetriken dieser drei Kategorien sind nicht unbedingt identisch. Das gilt auch für verschiedene Testszenarien der jeweiligen Kategorie[43]. Die Metriken sind wie folgt beschrieben:

- A-Auskopplung (gemeinsame Metrik für 1<sup>10</sup> und 2<sup>11</sup>)  
Eine Auskopplung tritt auf, wenn der Fahrer die zu bewertende ADS<sup>12</sup>-Funktion deaktiviert und die manuelle Steuerung des SV<sup>13</sup> übernimmt[43].
- B-Sicherheitsabstand (gemeinsame Metrik für 0<sup>14</sup>, 1 und 2)  
Der Sicherheitsabstand ist der minimale Abstand zwischen dem SV und

---

<sup>10</sup>geschlossene Strecke

<sup>11</sup>öffentliche Straße

<sup>12</sup>automated driving system

<sup>13</sup>Subject vehicle

<sup>14</sup>Simulation

der POVs<sup>15</sup>. Der Sicherheitsabstand wurde aus dem SV und den POVs, der Geschwindigkeit und der Position ermittelt. Er sollte kontinuierlich geprüft werden[43].

- C-Signalstatus (gemeinsame Metrik für 0, 1 und 2)

Der Signalstatus ist der Aktivierungszustand des SV-Blinkers, der in regelmäßigen Frequenzen gemessen wird um zu bestimmen, wann das Signal an oder aus ist[43].

- D-Beschleunigungs-/Entschleunigungsrate (gemeinsame Metrik für 0, 1 und 2)

Die Beschleunigungs-/Verzögerungsrate ist die Änderungsrate der Geschwindigkeit des Fahrzeugs. Die Beschleunigungs-/Verzögerungsrate, je nach Testzweck, erfolgt wenn die Bedingung eines Tests erfüllt ist. Die Änderungsrate soll gleichmäßig sein. Im Fall einer abrupten Verzögerung (z.b. wenn sich der SV dem POV nähert) muss diese Änderung erkannt werden[43].

- E-Kreis basierte Kollisionsprüfung (gemeinsame Metrik für 0, 1 und 2)

Die kreisbasierte Kollisionsprüfung ist eine der am meisten angewandte Methode zur Kollisionsprüfung. Ego und POV werden mit ein oder mehreren Kreisen (je nach Auflösung) dargestellt. Der Abstand zwischen beide Kreiszentren soll größer als die Summe der Radien des Ego plus des POVs sein. Entlang der Trajektorie sollte keine Kollision erkannt werden[62].

- F-Ziel erreicht (gemeinsame Metrik für 0, 1 und 2)

Die angegebene Zielpunktkoordinate sollte erreicht werden, d.h. die letzte Position von Ego sollte die Zielposition sein, gegebenfalls mit kleiner Abweichung. Die Prüfung kann auch erkennen, ob das Ego fehlgestartet ist (Ziel nie erreicht), oder nicht am Ziel anhält (Weiterfahrt nach dem Ziel)[43].

- G-Geschwindigkeitsgrenze (gemeinsame Metrik für 0, 1 und 2)

Die Geschwindigkeitsbegrenzung auf der Strasse muss eingehalten werden. Die Erkennung von Geschwindigkeitsvariationen entlang der Trajektorie und die Kalibrierung der Geschwindigkeit in Bezug auf ODD soll ebenso möglich sein[43].

Parametervariation:

Die Parameter können je nach Testzweck variiert werden [43]:

Ego-Geschwindigkeit, POV-Geschwindigkeit, Anzahl der Egos, Anzahl der POVs, zulässige Höchstgeschwindigkeit, Geschwindigkeit der Ampelschaltung, hinzufügen von verschiedenen Verkehrsteilnehmern (Fußgänger, Fahrräder, etc.).

---

<sup>15</sup>Principal other vehicle

Änderung des Verkehrsteilnehmerverhaltens (wütende Autofahrer, wilde Motorradfahrer, unachtsame Fußgänger, etc.). Die Parametervariation kann manuell oder automatisch erfolgen.

Verschiedene Variationstechniken wurden identifiziert:

- Fuzzing oder Fuzz-Testing (siehe Waymo Beispiel[63])
- Empirische Parameter-Variationsanalyse[64]
- Modellbasiertes Testen[65]

Die Parametervariation wird in dieser Masterarbeit nicht betrachtet und wird nur als Vorschlag einer möglichen Erweiterung diese Arbeit angenommen werden.

## 4.4 Erstellung von Testszenarien

In diesem Abschnitt werden Testszenarien aus der Tabelle 7.7 erstellt.

### 4.4.1 Test Vorlage

Testszenario-Launchdateien haben die gleiche xml-Struktur und unterscheiden sich nur in wenigen Nodes und Parametern. Eine Vorlage wurde erstellt und dient als Quelle für die Erstellung von Testszenarien. Die Vorlage ist im Code 7.1 zu sehen und wird hier beschrieben:

- Zeile 17: Der Tag `<group ns="vehicle0">` sorgt dafür, dass alle Nodes in der Launch Datei unter dem gleichen Namespace „vehicle0“ gestartet werden. Dazu verhindert der Namespace Konflikte, wenn mehrere Nodes durch mehrere Egos bzw. Launch Dateien gleichzeitig gestartet werden. Nachrichten lassen sich auch damit gut erkennen.
- Zeile 18 bis 21: Ein „Rosbag recorder“ Node wird dargestellt, der für die Aufzeichnung von Nachrichten verantwortlich ist. Die Nachrichten werden am Ende des Testszenarios in einer „.Bag“ Datei gespeichert. Diese „.Bag“ Datei wird für die Auswertung in späteren Schritten verwendet.
- Zeile 23: „Timer“ Node, der für die Zeitsynchronisierung der Module zuständig ist.
- Zeile 24: „sumotraffic2ros“ Node, die Schnittstelle zwischen SUMO und ROS

---

<sup>16</sup>Namespace

- Zeile 26: System under Test(sut) Parameter werden geladen. Hier werden mehrere Nodes aus der „sut.launch“ Datei abgerufen, die für die jeweils zugewiesene Funktion genutzt werden.
- Zeile 28 bis 30: Eine Straßenkarte „complex.xodr“ wird geladen und geplottet. Diese Karte ist anhand von SUMO erstellt worden. Die Abbildung4.6 zeigt die SUMO-Karte:

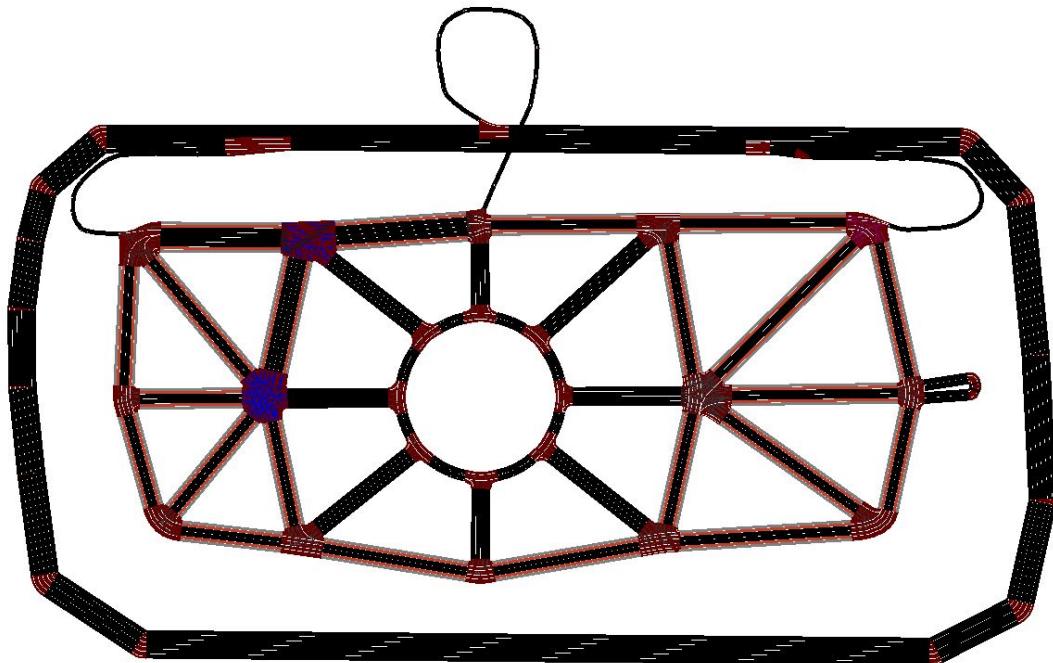


Abbildung 4.6: SUMO Karte complex

Mit netconvert kann die Sumo Karte in eine OpenDrive Karte konvertiert werden:

```
1 netconvert -s complex.net.xml --opendrive-output complex.xodr
2
```

Die konvertierte Karte wird in der Abbildung4.7 gezeigt:

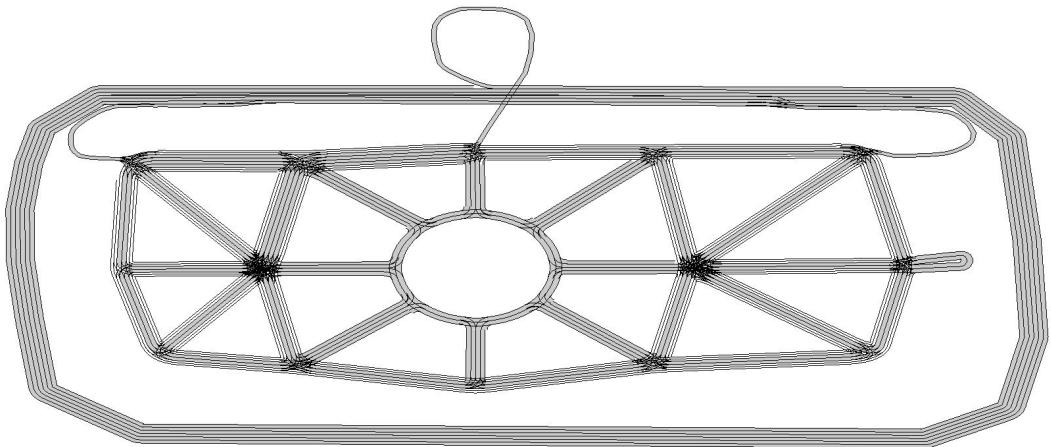


Abbildung 4.7: OpenDrive Karte complex

- Zeile 32: Das Ego wird einer Simulation ID zugewiesen.
- Zeile 34: Das Ego wird einer Startposition zugewiesen. Die x- und y-Werte können aus der SUMO-Karte im Netedit ausgelesen werden. Die Abbildung4.8 zeigt, wie die Mauszeigerposition ausgelesen wird. Unten Rechts sind die x- und y- Werte dargestellt. Hierbei muss darauf geachtet werden, dass die Ego-Position genau im Zentrum des Straßenabschnitts liegt, bzw. in der Mitte einer Spur.

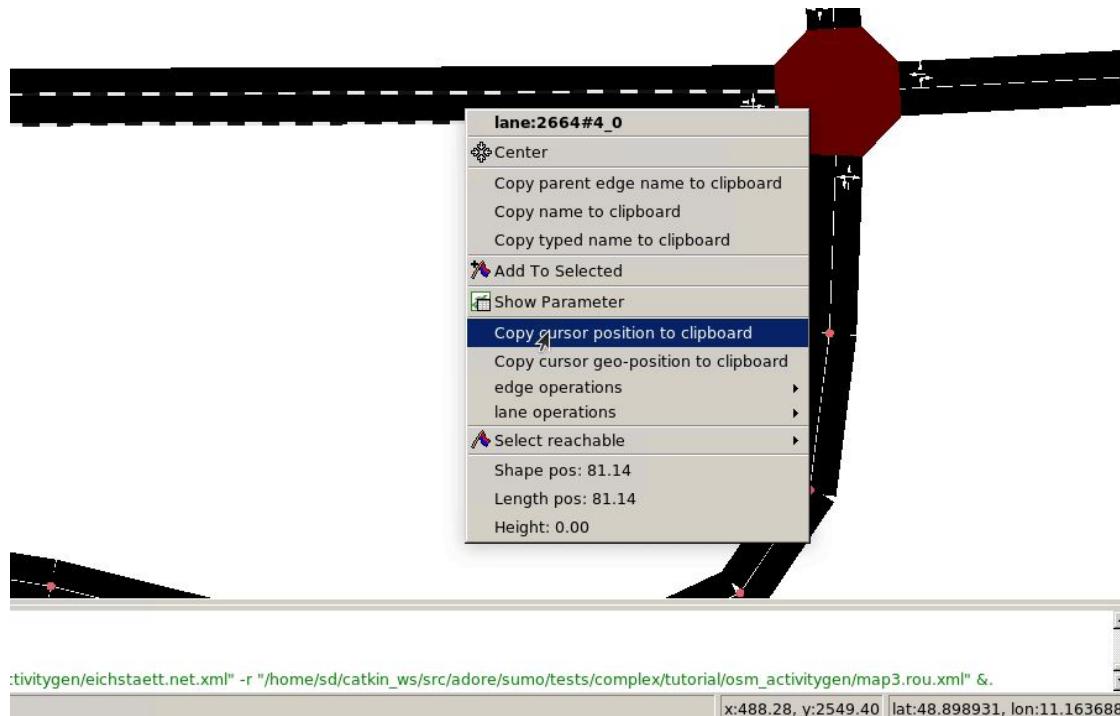


Abbildung 4.8: Auslesen der Startposition des Ego-Fahrzeugs

Die Orientierung des Egos wurde anhand von Quaternion[66] in ROS dargestellt.

- Zeile 36: Begrenzt die Ego-Geschwindigkeit auf den gegebenen Wert. Der Wert wird in m/s angegeben.
- Zeile 38: Eine Zielposition für die Navigation wird gesetzt. Die x- und y-Werte werden genauso wie in der Zeile 34 beschrieben, ausgelesen.
- Zeile 41 bis 45: Es werden die relevanten SUMO-Argumente wie z.B. sumocfg-Datei (siehe Abschnitt SUMO), Umgebungsvariablen und traciport geladen. Mit dem Starten des Plotlabserver und der Testvorlage "test000\_template.launch" mit folgendem Befehl wird das Ego von A nach B fahren:

```
1 cd ~/catkin_ws/src/adore/plotlabserver/;./stop.sh;./start.sh;
roslaunch ~/catkin_ws/src/adore/test/catalog_from_ntsa/
test000_template.launch
```

Listing 4.2: start test000

Die Abbildung 7.15 zeigt eine Übersicht. Das Ego ist als gelber-blauer Kasten dargestellt, die Referenztrajektorie(Richtung) als rote Linie und die Straßen grau.

#### 4.4.2 Parken

Der Parktest sollte folgende Kriterien erfüllen[43]:

- Parkplatz navigieren und suchen nach einem freien Parkplatz
- Geeignete Parkmanöver durchführen

Die Parkfunktion in ADORe ist noch nicht entwickelt worden (zum Zeitpunkt dieser Masterarbeit). Jedoch wurde das Testszenario (Test001\_paking.launch) mit den notwendigen Nodes erstellt, um den Test für das zuständige Entwickler-Team für die weitere Nutzung bereitzustellen, sobald die Parkfunktion integrierbar ist.

Das „Test001\_paking.launch“ enthält ein Parkszenario der SUMO Fahrzeuge. Das Ziel ist die Integration vom Ego in dieses Szenario, sodass das Ego rechtzeitig einen Parkplatz erkennt und das Parkmanöver richtig durchführen kann. Die Abbildung 7.16 zeigt eine Übersicht der SUMO-Parksimulation. Fahrzeuge sind als graue Dreiecke dargestellt, Busse sind Grün, besetzte Parkplätze rot und freie Parkplätze grün markiert.

#### 4.4.3 Fahrspurfolgen

Der Test für das Fahrspurfolgen (lane following) sollte folgende Kriterien erfüllen[43]:

- Sicherheitsabstand zu anderen Autos(Auto sicher folgen)
- Geschwindigkeitseinhaltung
- Spurzentrierung

In diesem Szenario folgt das Ego dem führenden Sumofahrzeug, behält eine Geschwindigkeit unter 5 m/s bei und fährt während des Tests in der Mitte der Spur. Das Sumo-Fahrzeug hat eine Höchstgeschwindigkeit von 5 m/s. Die Abbildung 7.17 zeigt den Ablauf. Zusätzlich sind hier Sumofahrzeuge als weiße Kästen dargestellt. Um die Geschwindigkeit von Ego- und Sumofahrzeugen im Blick zu behalten wird der „traffic topic“ durch diesen Befehl aufgerufen:

```
1 rostopic echo vehicleX/traffic
```

Listing 4.3: rostopic traffic

Die Variable X(s. Zeile 17 der Testvorlage) beschreibt die ID des Ego-Fahrzeugs in der Launchdatei. In diesem Fall X=2. Die JSON-Daten 7.2 zeigen bereits veröffentlichte Verkehrsdaten. Hier entspricht die Ego-TrackingID 1000, die Sumofahrzeug-TrackingID liegt über 1000, beispielsweise entspricht 1001 dem ersten Sumo-Fahrzeug, das in der Simulation erscheint, 1002 entspricht

dem nächsten Sumo Fahrzeug usw. Die JSON-Daten7.2 enthalten zusätzliche Informationen in bezug auf Ego- und Sumofahrzeuge, wie z.b die Position(pose: pose: position) und die Geschwindigkeit(Twist: Twist: Linear) und werden wie folgt (s. JSON-Daten7.2) gelesen:

- In Zeile 13 lässt sich die Position des SUMO-Fahrzeugs ermitteln. Sie beträgt ca 40,7 m auf der x-Achse, 163 m auf der y-Achse.
- In Zeile 26 ist die Geschwindigkeit des SUMO-Fahrzeugs dargestellt. Sie beträgt ca 5 m/s, was genau dem maximal eingestellten Wert im SUMO entspricht.
- In Zeile 54 ist die Tracking-ID angegeben, sie beträgt 1001 und bezieht sich auf das erste erscheinende SUMO-Fahrzeug in der Simulation.
- In Zeile 69 lässt sich die Position des Ego-Fahrzeugs ermitteln. Sie beträgt ca. 65,6 m auf der x-Achse, 163,8 m auf der y-Achse.
- In Zeile 82 befindet sich die Geschwindigkeit des Ego-Fahrzeugs. Sie beträgt ca. 5m/s, was genau der Anforderung zur Einhaltung der Geschwindigkeit entspricht
- In Zeile 110 ist die TrackingID gegeben, sie beträgt 1000 und weist auf das Ego-Fahrzeug in der Simulation hin.

Um den Sicherheitsabstand zu prüfen, muss folgende Rechnung vollzogen werden[67]:

```

1 xEgo=65 #ego pos
2 xPov=40 #sumo pos
3 s=5 #speed ego
4 Delta = abs(xEgo-xPov) #distance: 25m
5 # since we are driving inside urban, the speed limit is 50km/h or max ~15m/s,
   the ego speed speed is under 50km/h so the following formula applies:
6 SafeDistance = int((s*3.6) / 3.33) #minimum safe distance: 5m
7 # the distance is bigger than minimum safe distance which is safe

```

Listing 4.4: test002\_safe\_distance\_formula

Der Sicherheitsabstand wurde, wie das Ergebnis dieser Berechnung beweist, eingehalten[67].

#### 4.4.4 Spurwechsel

Der Spurwechseltest (lane change) muss folgende Kriterien erfüllen[43]:

- Spurwechsel/Überholen
- Ins Verkehr einfließen/einordnen

Zwei Spurwechselszenarien werden entsprechend erstellt.

## **test003\_lane\_change.launch**

In diesem Szenario müssen folgende Kriterien erfüllt werden[43]:

- Spurwechsel / Überholen

Das Ego führt ein Spurwechselmanöver von links nach rechts aus. Die Abbildung 7.18 zeigt diesen Vorgang. Ego-Fahrzeug ist als blauer Kasten dargestellt, Sumo-Fahrzeuge sind rot. In der „test003\_lane\_change.launch“ Datei wurde ein anderes „system under test“ (sut1. siehe 4.5) für den Spurwechsel inkludiert.

```
1  <!-- Load sut Parameter -->
2  <include file="$(dirname)/sut1.launch"/>
```

Listing 4.5: rostopic traffic

## **test003\_lane\_change\_merge.launch**

In diesem Szenario müssen folgende Kriterien erfüllt werden[43]:

- mit niedriger Geschwindigkeit einordnen

Das Ego führt ein Spurwechselmanöver mit niedriger Geschwindigkeit von rechts nach links aus. Die Abbildung 7.19 zeigt diesen Vorgang. Ego-Fahrzeug ist als blauer Kasten dargestellt, Sumo-Fahrzeuge sind rot. Die rote Linie zeigt wie das Ego die Lücke erkennt und einfährt.

### **4.4.5 Kreuzungsnavigation**

Der Kreuzungsnavigationstest muss folgende Kriterien erfüllen[43]:

- Einfahrt/Ausfahrt auf Autobahn (Ein-/Aus-Rampen)
- Kreisverkehr
- Kreuzung (links, rechts, geradeaus)
- Zebrastreifen
- U-Turn (Umdrehung)

6 Kreuzungsnavigationszenarien werden entsprechend erstellt.

## **test004\_navigate\_intersection\_onramp**

In diesem Szenario muss folgende Kriterium erfüllt sein[43]:

- Einfahrt navigieren

Die Abbildung 7.20 zeigt (von oben nach unten), wie das Ego die Einfahrt auf die Autobahn nimmt. Das Ego fährt entlang der Einfahrt um hohe Geschwindigkeit zu erreichen und fließt vorsichtig in die Autobahn ein.

### **test004\_navigate\_intersection\_offramp**

In diesem Szenario muss folgendes Kriterium erfüllt sein[43]:

- Ausfahrt navigieren

Die Abbildung7.21 zeigt (von oben nach unten), wie das Ego die Autobahn-ausfahrt nimmt, um in die Stadt zu fahren. Das Ego verlässt die mittlere Spur mit Berücksichtigung der Verkehrsteilnehmern um auf der rechten Fahrspur zu gelangen.

### **test004\_navigate\_intersection\_Roundabouts**

In diesem Szenario muss folgendes Kriterium erfüllt sein[43]:

- Im Kreisverkehr navigieren

Die Abbildung7.22 zeigt (von oben nach unten), wie das Ego im Kreisverkehr navigiert und die nächste Ausfahrt annimmt. Das Ego ist blau, die SU-MO Fahrzeuge sind weiß dargestellt. Die Richtung des Egos ist in roter Linie dargestellt.

### **test004\_navigate\_intersection\_lrs**

In diesem Szenario muss folgendes Kriterium erfüllt sein:

- in einer Kreuzung navigieren (links, rechts und geradeaus)

Die Abbildung7.23 zeigt (von oben nach unten), wie das Ego in einer Kreuzung navigiert um zum jeweiligen Ziel zu gelangen. Durch die erste Kreuzung nach rechts, durch die zweite nach links und durch die dritte fährt das Ego geradeaus zu seinem Ziel.

### **test004\_navigate\_intersection\_crosswalk**

In diesem Szenario muss folgendes Kriterium erfüllt sein:

- Über Zebrastreifen fahren

Die Abbildung7.24 zeigt wie das Ego bei einer Kreuzung Fußgänger beim Überqueren erkennt und rechtzeitig anhält. Die Fußgänger sind als schwarze Punkte dargestellt.

### **test004\_navigate\_intersection\_uturn**

In diesem Szenario muss folgendes Kriterium erfüllt sein:

- U-Turn (Umdrehung) fahren

Die Abbildung 7.25 zeigt (von oben nach unten), wie das Ego einen U-Turn durchführt. Einige Funktionen im Framework (wie z.B. das Spurwechsel) sind noch in der Entwicklungsphase, daher stellen diese Demonstrationen keine Beweise oder Garantie der Wiederholbarkeit von Testabläufe bzw. Testergebnisse dar.

## **4.5 Automatisierung Testszenarien**

In diesem Abschnitt wird die Automatisierung von den oben genannten Testszenarien durch die Gitlab-CI/CD durchgeführt. Die Gitlab-CI/CD (siehe 4.1.3) hilft bei der Automatisierung von Prozessen. Die Verwendung der CI mit den Testszenarien kann folgendes mitbringen:

- a- automatische Ausführung von Testszenarien
- b- automatische Auswertung von Testszenarien
- c- automatische Generierung von Testszenarien

Punkt a sorgt für fehlerfreie Ausführung von Testszenarien während der Entwicklung vom ADORe. Durch die kontinuierliche Entwicklung des Frameworks können eigene Funktionen fehlschlagen oder auf eine Inkompatibilität des Quellcodes verweisen. Damit können diese Fehler frühzeitig erkannt werden.

Punkt b hilft bei der Auswertung dieser Testszenarien. Bewertungsmetriken (s. Tabelle 7.7) werden eingesetzt, um diese Tests auszuwerten. Pass/Fail Kriterien entscheiden, ob die Testszenarien gelungen sind. Somit können Testergebnisse auf Redundanz und Reproduzierbarkeit geprüft werden. Die Auswertungsmethodik wird im folgenden beschrieben.

Punkt c kann als Ausblick für diese Arbeit angesehen werden. Beliebige Parameter können in die Tabelle eingefügt werden (s. Tabelle 7.7) um neue Testszenarien in Abhängigkeit der jeweils gewünschten Daten (z.B. der Geschwindigkeit) automatisch zu generieren. Die Komplexität steigt und ein großes Parameterspektrum kann dadurch exploriert werden. Im nächsten Abschnitt wird der Punkt a beschrieben.

#### **4.5.1 Gitlab CI/CD Pipelines**

Pipelines gemäß Gitlab-CI/CD[68] sind die Top-Level-Komponenten der kontinuierlichen Integration, Lieferung (delivery) und Bereitstellung (deployment). Pipelines sind eine Sammlung von Stages, die Jobs enthalten. Jobs[68] sind Aufgaben, die ausgeführt werden, z.B. Jobs für kompilieren, testen, etc.

Stages organisieren die Reihenfolge der Ausführung von Jobs[68], z.B. bei Stage Install werden Jobs für die Installation ausgeführt, bei Stage Test werden Jobs zum testen aufgerufen. Jobs werden von Gitlab Runner[69] ausgeführt. Runner[69] ist eine Jobmanager-Applikation, die auf dem Betriebssystem installiert wird.

Für detaillierte Informationen über die Runner Installation, Registrierung und Ausführung siehe Gitlab-Runner[69]. Jobs können sequentiell oder parallel laufen. Wenn Jobs auf verschiedene Stages verteilt sind, laufen sie nacheinander. Im Gegensatz dazu laufen sie gleichzeitig wenn die Jobs alle zur gleichen Stage gehören[69].

Wenn alle Jobs in einer Stage erfolgreich sind, fährt die Pipeline mit der nächsten Stage fort. Andernfalls wird die Pipeline unterbrochen. Ausnahmen existieren und werden im folgenden beschrieben. In der Regel laufen Pipelines völlig automatisch bis diese abgeschlossen oder unterbrochen werden. Im üblichen CI/CD Prozess besteht eine Pipeline aus vier Stages[68]:

- Build-Stage mit dem Job „Compile“.
- Test-Stage mit dem Job „test“.
- Staging-stage mit dem Job „Deploy-to-Stage“.
- Produktion-Stage mit dem Job „Deploy-to-Prod“.

Pipelines werden durch Trigger gestartet. Der übliche Trigger ist der „git push“ Befehl, der die letzte Änderung der lokalen Git-Repository<sup>17</sup> auf dem Remote-Repository (Server) hochlädt (pusht)[68]. Somit wird die Pipeline getriggert und führt die Schritte in der YAML-Datei durch. Die YAML-Datei wird im nächsten Abschnitt erläutert.

#### **4.5.2 GitLab-YAML**

Yaml ist eine Datei mit dem Namen .gitlab-ci.yml, sie befindet sich im Rootverzeichnis des Git-Repositorys und enthält die CI/CD-Konfiguration (s.[70])

---

<sup>17</sup>Git-Repository: Arbeitsordner, indem git initialisiert wird

für mehr Details). Eine neue Git-Branche<sup>18</sup> wurde im ADORe erstellt, um diese Masterarbeit unabhängig von der Master<sup>19</sup>/develop<sup>20</sup>-Branche zu erstellen.

Die experimentellen Änderungen werden somit von dem Framework-Kern isoliert, bleiben jedoch auf dem neuesten Stand. Die derzeitige aktuelle Branche dieser Masterarbeit heißt „developCi“ und die Ci-Pipeline wird nur auf dieser Branche laufen. Die Yaml Datei im „developCI“ enthält vier Stages. Die erste Stage „Builld“ (s. Code7.3) wird im folgendem beschrieben.

Der CI-Prozess wird auf einem Docker Container laufen. Das Docker Image „gitlabciadore/adoreci:v0.1“[71] wurde aus Ubuntu 20.04 mit den ADORe-notwendigen Abhängigkeiten bereitgestellt und auf dem Docker-Hub gehostet. Um eine detaillierte Beschreibung der Docker-Befehle zu erhalten, bietet die Docker-Doku Website[72] relevante Informationen darüber. Der Code7.3 wird wie folgt beschrieben:

- Zeile 1-12: zeigt die Beschreibung (Kommentare) der gesamten Yaml Datei (werden im Weiteren detailliert beschrieben)
- Zeile 14: variables: globale Variable, der Ordner wird als globale Variable gegeben.
- Zeile 17: Stages: Die Reihenfolge von Stages (werden einzeln beschrieben)
- Zeile 23: build\_adore(Job) der Stage(build)
- Zeile 25: tags: ci\_test (um den richtigen Runner zuzuweisen)
- Zeile 26: rules: if: '\$CI\_COMMIT\_BRANCH == "developCi" (um die richtige Branche zuzuordnen)
- Zeile 28: Skript: die Abfolge von Skripten/Befehle, die während des Jobs ausgeführt werden.
- Zeile 29: „sudo“ (root Privileg), „chmod 777“ erlaubt den Zugriff auf dem Ornder gitlab-runner, „|| true“ falls der Zugriff schon erlaubt.
- Zeile 30-32: docker stop: stoppt laufende Container, docker rm: löscht gestoppte Container, docker prune: um sicher zu löschen, „|| true“ um die Zeile zu überspringen, falls keine Container gefunden werden.

---

<sup>18</sup>git-Branche: Eine Branche repräsentiert eine unabhängige Entwicklungslinie in Git, die bei der Entwicklung neuer Features oder Fehlerfindung hilft, in Unabhängigkeit von anderen Entwicklern

<sup>19</sup>Master-Branche: neueste stabile Entwicklungsversion

<sup>20</sup>develop-Branche: neueste Entwicklungsversion (meistens nicht stabil)

- Zeile 33-36: löscht den Ordner(as glob. Variable Oben deklariert) von letzter Session und erstellt ein neuen, Kopiert vom Standard Runner Build-Dir ins SharedVol und exportiert die Variable der adore-libadore-Branche.
- Zeile 38-45: docker run[73]: startet den Container mit dem Namen „test\_cont“ aus dem Image(gitlabciadore/adoreci:v0.1), exportiert die genannten Variablen (-e), erstellt die genannte Volumen<sup>21</sup>[74] (-v). Die Option „-idt und –entrypoint=/bin/bash“ sorgt dafür, dass der Container nicht sofort schließt und während des gesamten Prozesses an bleibt.
- Zeile 47: docker exec: führt einen Befehl im laufendem Container aus, hierbei wird das „runner\_script“ ausgeführt. Im runner\_script werden adore\_if\_ros und sumo\_if\_ros Pakete(im Container) installiert und die entsprechenden Umgebungsvariablen exportiert.
- 53: artifacts: sind Dateien, die auf dem CI-Server hochgeladen werden. Diese Dateien werden für nächste Stages verfügbar. In dem Fall werden alle Dateien im build Ordner mit der Erweiterung txt und log übermittelt.

Die zweite Stage im CI ist der unit\_test. Diese enthält einen ctest Job und wurde mit dem Befehl “docker exec ctest” durchgeführt(s. Zeile 8 im folgendem Code).

```

1 ctest:
2   stage: unit_test
3   tags: [ci_test]
4   rules:
5     - if: '$CI_COMMIT_BRANCH == "developCi"'
6
7   script:
8     - docker exec -t test_cont bash -c "cd /home/adore_workspace_dir/build/
9       adore_if_ros && ctest && touch /home/outputdir/build/variables"
10      - sudo cp -r $SharedVol/adore/build/* $CI_PROJECT_DIR/build/
11      - sudo chmod 777 -R /home/sd/gitlab-runner/ || true
12
13 artifacts:
14   when: always
15   paths:
16     - build/*.txt
     - build/*.log
     - build/variables

```

Listing 4.6: Stage unit-test

Hier wird zusätzlich ein Ordner „variables“ erstellt und als artifacts übergeben. Dieser Ordner wurde in den nächsten Stages für die Auswertung von Testszenarien gebraucht.

---

<sup>21</sup>Volume: Schnittstelle zwischen Container und Host-Maschine

Die dritte Stage „ci\_scenarios“ beinhaltet Testszenarien, die gleichzeitig ausgeführt werden. Wie bereits oben erklärt, Jobs in der gleichen Stage werden parallel ausgeführt. Jobs bzw. Testszenarien in dieser Stage haben die gleiche Struktur, deswegen wird nur einen Job hiermit erläutert:

```

1 test0:
2   stage: ci_scenarios
3   tags: [ci_test]
4   rules:
5     - if: '$CI_COMMIT_BRANCH == "developCi"'
6       allow_failure: true
7
8   script:
9     - docker exec -t test_cont bash -c 'chmod +x /home/adore_workspace_dir/
10    src/adore/test/ci/testScenario_template.bash'
11    - docker exec -t test_cont "/home/adore_workspace_dir/src/adore/test/ci/
12      testScenario_template.bash"
13    - sudo cp -r $SharedVol/adore/build/* $CI_PROJECT_DIR/build/
14    - sudo chmod 777 -R /home/sd/gitlab-runner/ || true
15
16   artifacts:
17     when: always
18     paths:
19       - build/*.txt
20       - build/*.log
21       - build/*.bag
22       - build/test_ci/*.csv
23       - build/variables

```

Listing 4.7: Stage ci\_scenarios

Der Job test0 ist für das Template-Szenario quasi als Vorlage gemacht. Die Zeile 9 macht das Skript ausführbar. Die Zeile 10 führt dann den testScenario\_template.bash aus. In diesem Skript4.8 werden Umgebungsvariablen exportiert, dann wird die Launchdatei gestartet und als letztes wird das Ergebnis mittels Python Skript ausgewertet.

```

1 export ROS_DISTRO=noetic
2 export ROS_MASTER_URI=http://localhost:11350
3 export SUMO_HOME=/home/adore_workspace_dir/src/adore/sumo
4 #source /opt/ros/noetic/setup.bash
5 source /home/adore_workspace_dir/install/setup.bash
6
7 cd /home/adore_workspace_dir/src/adore/test/ci
8 roscore -p 11350 &
9 sleep 1
10 roslaunch -p 11350 --wait test000_template.launch > /home/outputdir/build/
    test_ci.log

```

Listing 4.8: Testszenario Skript

Da mehrere Launchdateien(Testszenarien) gleichzeitig laufen sollten, sollten mehrere ROS-Master-Ports pro Launchdatei zugeordnet werden(s. Zeile

2). Das liegt daran, dass ähnliche Nodes gleichzeitig von mehreren Launchdateien zugegriffen werden. Der Namespace(s. Erstellung Testszenarien) reicht hier nicht um den Konflikt zu beheben. Daher sind Launchdateien mit separaten Master-Ports ausgestattet. Die Zahl 11350 ist der PORT für die Kommunikation mit dem Master. Pro Testszenario sind unterschiedliche PORTs genannt.

Der Roscore(s. Zeile 8) wird mit dem zugewiesenen PORT gestartet. Die Launchdatei(s. Zeile 10) wird ebenfalls mit dem zutreffenden PORT gestartet. Somit wird garantiert, dass kein Konflikt mit dem parallelen starten geben wird. Zudem sind Anpassungen in der Launchdatei notwendig, um diese im CI-Prozess zu integrieren. Die Rosbag-record Node4.9 wurde aktiviert, um relevante Nachrichten für die spätere Auswertung aufzuzeichnen.

```

1 <launch>
2   <group ns="vehicle0">
3     <arg name="record_rosbag" default="true" />
4     <!-- this starts the rosbag recording - name of the bag should be same as
      launchfile -->
5     <node pkg="rosbag" type="record" name="sim_rosbag_recorder" args="record
       -O /home/outputdir/build/test_ci.bag -a"
6       if="$(arg record_rosbag)"/>
```

Listing 4.9: Rosbag-record Node

Ein Terminator-Node4.10 wurde hinzugefügt, um den Testablauf zeitlich zu kontrollieren. Nach 60 Sekunden wird der Terminator Node ausgeschaltet. Die Option(Zeile 3) required="true" lässt die gesamte Launchdatei nach 60s herunterfahren.

```

1   <!-- this node is important to have the scenario stop at time /PARAMS/SIM/
    timeout -->
2   <param name="/PARAMS/SIM/timeout" value="60.0" type="double"/>
3   <node pkg="adore_if_ros" name="terminator" type="adore_ci_terminator_node
     .py" output="screen" required="true"/>
```

Listing 4.10: Terminator Node

Das Plotten wird deaktiviert und der Plotter Node(s.4.11) wird auskommentiert da im CI-Prozess kein graphisches Interface(Headless Mode) der Simulation gebraucht wird. Zusätzlich unterstützt der Docker Container keine graphische Applikation.

```

1   <!-- do not start anything that tries to plot -->
2   <!--node name="plotter" pkg="adore_if_ros" type="adore_fancybird_node"-->
```

Listing 4.11: Headless Mode

Im letzten Teil des Skripts4.12 geht es um die Auswertung des Testszenarios. Der „Modelchecker.py“ (Zeile 2) hat folgende Argumente:

- **-input:** die zugehörige Bag-Datei4.9 wird als input für das Skript genommen.
- **-output:** eine Textdatei mit dem gleichem Namen wie Bag-Datei wurde als output genommen, in dem das Resultat "test passed" oder "test failed" basierend auf dem Ergebnis geschrieben wird.
- **-topic:** kann eins bzw. mehrere Topics annehmen, je nach Szenario. Die bisher analysierten Topics sind dargestellt. Der Inhalt je Topic wird im nächsten Kapitel beschrieben.

```

1 chmod +x ModelChecker.py
2 python3 ModelChecker.py --input /home/outputdir/build/test_ci.bag --output /
   home/outputdir/build/test_ci.txt --topic /vehicle0/ENV/propositions /
   vehicle0/VEH/ax /vehicle0/odom /vehicle0/traffic
3 EVAL=$(head -n 1 /home/outputdir/build/test_ci.txt)
4 if [ "$EVAL" = "test passed" ]; then echo "test0" >> /home/outputdir/build/
   variables && echo $EVAL; else EVAL=false && echo $EVAL && exit 1; fi

```

Listing 4.12: Testszenario Skript

Die Variable EVAL liest die erste Zeile der Textdatei und gibt aus, ob der Test erfolgreich war oder nicht. Falls der Test gelungen ist, wird der Name dieses Tests (Job) in variables angefügt. Somit werden 11 Tests(Jobs) implementiert, die gleichzeitig im CI ausgeführt und ausgewertet werden.

Die vierte (letzte) Stage4.13 zeigt, welche Tests erfolgreich waren.

```

1
2 evaluate:
3   stage: evaluation_deploy
4   tags: [ci_test]
5   rules:
6     - if: $CI_COMMIT_BRANCH == "developCi"
7     script:
8       - cat $SharedVol/adore/build/variables | while read line || [[ -n $line
9         ]]; do echo "successful test's name:$line"; done
10      - sudo cp -r $SharedVol/adore/build/* $CI_PROJECT_DIR/build/
11      - sudo chmod 777 -R /home/sd/gitlab-runner/ || true
12      - docker stop $(docker ps -aq) || true ##stop running container: || true
13      =1 if no container already found
14      - docker rm $(docker ps -aq) || true ## remove them
15      - docker container prune ## Removes all stopped containers.
16
17 artifacts:
18   when: always
19   paths:
20     - build/*.txt
21     - build/*.log
22     - build/*.bag
23     - build/test_ci/*/*.csv

```

**Listing 4.13: Stage evaluation\_deploy**

Die Zeile 8 iteriert durch die variables und gibt die Namen der bestehenden Tests aus. Diese Stage kann auch zusätzliche Aufgaben enthalten, beispielsweise die Parametervariation von erfolgreichen Tests und die automatische Generierung neuer Tests, oder die Bereitstellung zu der Produktionsphase, z.b. von Gitlab(internal) auf Github(extern) zu übertragen. Dies ist jedoch nicht der Kern dieser Arbeit und wird als Erweiterung dieser Arbeit betrachtet. Im nächsten Kapitel wird die Methodik der Auswertung sowohl theoretisch als auch praktisch erläutert.

# Kapitel 5

## Auswertung

In diesem Kapitel wird die Methodik der Auswertung von Testszenarien erläutert. Als erstes wird der theoretische Teil zusammengefasst, dann die praktische Umsetzung erklärt.

### 5.1 Model Checking

Model-Checking (Modellprüfung) sinngemäß[75, 76] ist ein Prozess zur automatischen Prüfung eines Modells mit einer Formel. Wenn ein Modell ( $M$ ) eine Formel ( $\phi$ ) erfüllt, stoppt der Algorithmus und liefert ein Korrektheitszertifikat als Ausgabe. Die mathematische Formulierung lautet[76]:

$$M \models \phi \quad [77] \tag{5.1}$$

(engl.  $M$  satisfies  $\phi$ )

Model-Checking wird durch eine Kripke-Struktur modelliert. Die Beschreibung von der Kripke-Struktur folgt im nächsten Abschnitt.

#### 5.1.1 Kripke-Struktur

Eine Kripke-Struktur sinngemäß[78] ist ein endliches Transitionssystem, welches durch einen gerichteten Graphen<sup>1</sup> dargestellt wird. Die Knoten stellen die erreichbaren Zustände<sup>2</sup> des Systems dar und die Kanten stellen die Zustandsübergänge<sup>3</sup> dar. Die Kripke-Struktur nach[78] ist ein Tuple<sup>4</sup>(**S,I,R,L**):

- **S:** endliche Menge(SET) von Zuständen(states)

---

<sup>1</sup>directed Graph: einer Graph mit Knoten und Kanten repräsentiert[78]

<sup>2</sup>states

<sup>3</sup>state transitions

<sup>4</sup>ist eine endliche geordnete Liste (Folge) von Elementen in der Mathematik

- **I**: ein Satz von Initialenzuständen,  $I \subset^5 S$ .
- **R**: eine ÜbergangsRelation-Satz:  $\forall s^6 \in^7 S \exists^8 s' \in S$  sodass  $(s,s') \in R$ .
- **L** :  $S \rightarrow 2^{(AP)^9}$ : Labeling-Funktion, ordnet jedem Zustand eine Reihe von Atomarsätzen zu.

Die folgende Abbildung 5.1 zeigt ein Beispiel, um die Kripke-Struktur zu verdeutlichen.

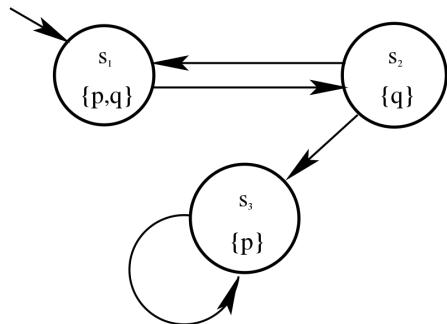


Abbildung 5.1: Beispiel Kripke-Struktur [79]

Die Abbildung 5.1 zeigt eine Kripke-Struktur  $K = (\mathbf{S}, \mathbf{I}, \mathbf{R}, \mathbf{L})$  wobei:

- $\mathbf{S} = \{s_1, s_2, s_3\}$
- $\mathbf{I} = \{s_1\}$
- $\mathbf{R} = \{(s_1, s_2), (s_2, s_1), (s_2, s_3), (s_3, s_3)\}$
- $\mathbf{L} = \{(s_1, p, q), (s_2, q), (s_3, p)\}$

Formaler ausgedrückt, für eine Kripke-Struktur  $K = (\mathbf{S}, \mathbf{I}, \mathbf{R}, \mathbf{L})$  und eine Formel  $\phi$  existiert  $S' \subseteq S$  sodass für alle  $s_i \in S'$  folgende Gleichung gilt[75]:

$$K, s_i \models \phi \text{ (engl. } K, s_i \text{ satisfies } \phi)[75] \quad (5.2)$$

### 5.1.2 Temporale Logik

Temporale Logik gemäß[80] ist eine Erweiterung der üblichen Aussagenlogik in Bezug auf die Zeit. Die Temporale Logik gibt zeitliche Aussagen über ein Modell. Temporale Logiken werden in der Regel als Kripke-Strukturen interpretiert. Für die Auswertung von Testszenarien mittels Model-Checking wird die temporale Logik „Computation Tree Logic\*“ verwendet[80].

---

<sup>5</sup>Subset: Teilmenge

<sup>6</sup>states

<sup>7</sup>in: gehört zu

<sup>8</sup>existiert

<sup>9</sup>atomic proposition

### 5.1.3 Computation Tree Logic\*

Die Computation Tree Logic\* (CTL\*)[81] ist eine temporale Logik, die die Aussagenlogik<sup>10</sup> erweitert und die Eigenschaften von Berechnungsbaum-Logik<sup>11</sup> über Kripke-Strukturen beschreibt[81]. Die CTL\* ist eine Verallgemeinerung der beiden temporalen Logiken LTL<sup>12</sup> und CTL<sup>13</sup>. In LTL hat die Zukunft nur einen Pfad. In CTL hingegen hat die Zukunft mehrere Pfade (Branching Time). Die CTL\* Logik wird durch eine Kripke-Struktur(s.Abbildung5.2) modelliert[76] und in der weiteren praktischen Umsetzung verwendet.

### 5.1.4 Syntax

Die CTL\* kombiniert Standardlogiken, temporale und Pfad-Operatoren in ihr Alphabet[81]. In der Tabelle7.8 wird die Syntax erklärt. Die folgende Abbildung5.2 zeigt Beispielmodelle.

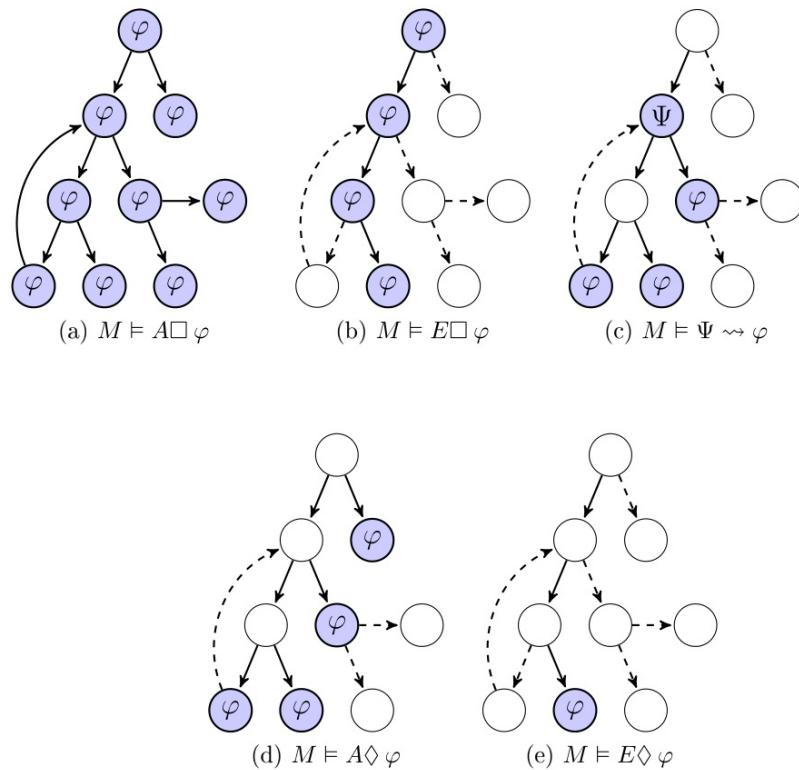


Abbildung 5.2: Beispiele CTL Formeln [82]

<sup>10</sup>PL: Propositional Logics

<sup>11</sup>Computation Tree Logic

<sup>12</sup>Linear temporal logic

<sup>13</sup>Computation Tree Logic

Für folgende Beispiele, A bedeutet  $\forall$  und E bedeutet  $\exists$  (s. Tabelle 7.8).

- Beispiel (a):  $M \models A\Box\varphi$ : für alle Pfade,  $\varphi$  gilt auf dem kompletten nachfolgenden Pfad(in jedem Zustand). Das Modell M erfüllt(satisfies) die Formel  $A\Box\varphi$ .
- Beispiel (b):  $M \models E\Box\varphi$ : Es existiert mindestens ein Pfad, sodass  $\varphi$  entlang des ganzen Pfades gilt. Das Modell M erfüllt(satisfies) die Formel  $E\Box\varphi$ .
- Beispiel (c):  $M \models \Psi U\varphi$ :  $\Psi$  gilt solange, bis  $\varphi$  im nächsten Zustand gilt. Das Modell M erfüllt(satisfies) die Formel  $\Psi U\varphi$ .
- Beispiel (d):  $M \models A\Diamond\varphi$ :  $\varphi$  gilt irgendwann auf dem nachfolgenden Pfad. Das Modell M erfüllt(satisfies) die Formel  $A\Diamond\varphi$ .
- Beispiel (e):  $M \models E\Diamond\varphi$ : Es existiert ein Pfad, sodass  $\varphi$  irgendwann auf dem nachfolgenden Pfad gilt. Das Modell M erfüllt(satisfies) die Formel  $E\Diamond\varphi$ .

## 5.2 Umsetzung

Nach der Erklärung des theoretischen Teils der Auswertung von Testszenarien durch die Modellprüfung (s. Model Checking 5.1) wird die praktische Umsetzung im ADORe erklärt.

### 5.2.1 ModelChecker.py

Wie im Kapitel Implementierung bereits erwähnt, wurde ein Python Skript „ModelChecker.py“ für die Auswertung erstellt (s. 4.12). Das Skript nimmt Bag-Dateien als Input, Textdatein als Output und ausgewählte Topics je nach Szenario. Um die Bag-Datei lesen zu können, wird das Modul „Bagpy“ installiert und importiert. Die Bag Datei enthält alle Topics im Szenario, ausgewählte Topics werden filtriert. Dataframes (Datenrahmen) werden aus den gefilterten Daten erstellt. Je nach Topic werden Schlüsselwörter extrahiert und zusammen in einer Tabelle abgelegt.

Aus den Datenrahmen werden Kripke-Strukturen generiert. Dafür ist das Modul „pyModelChecking[83]“ notwendig, um Model-Checking zu integrieren. Je Kripke Struktur wird eine passende Formel geprüft, ob die Kripke Struktur diese Formel erfüllt. Falls die Kripke-Struktur die Formel erfüllt, gilt das Resultat als „wahr“(true). Ausgewählte Topics werden gleichmäßig geprüft. Wenn das Resultat in allen Topics Wahr ist, gilt der Test als bestanden, andernfalls nicht bestanden. Eine Übersicht des „ModelChecker.py“ Skripts ist hiermit dargestellt:

```

1 from MonitorClass import *
2
3 a = Analyser()
4 a.ParseArgument()
5 a.AnalyseTopics()

```

Listing 5.1: ModelChecker.py

Um die Lesbarkeit und Portabilität des Skripts zu gewährleisten, wurde ein „MonitorClass“ Modul erstellt. Alles, was im Modul ist, wurde zuerst importiert.(Zeile 1). Eine Instanz(a) der Klasse Analyser wurde erzeugt. Die Methoden ParseArgument und AnalyseTopics wurden dann aufgerufen(Zeile3-5). Der interne Mechanismus des „MonitorClass“ wird im Weiteren beschrieben.

### 5.2.2 MonitorClass.py

Der MonitorClass.py enthält zwei Klassen: Monitor und Analyser. „Class Monitor“ enthält ganze Methoden (Funktionen) für die Datenverarbeitung. „Class Analyser“ enthält nur zwei Methoden: ParseArgument und AnalyseTopics. Konstruktor-Methode wird nicht gezählt.

### 5.2.3 ParseArgument

Die Methode ParseArgument ist für die Skriptargumente zuständig, d.h. die Kommandozeile wird damit definiert. Eine Übersicht dieser Methoden ist im Code7.4 zu sehen. Der Block(Zeile 15-27) beschreibt das „Parsing“ von Argumenten. Das Modul „argparse“ wurde verwendet.

### 5.2.4 AnalyseTopics

Der Kernmechanismus läuft unter der AnalyseTopics Methode. Zunächst wird die Liste der Topics iteriert und einzelne Topics analysiert.

- Topic /vehicleX/ENV/propositions (s. Code7.5): enthält Nachrichten für „IN\_COLLISION“ (Kollisionsprüfung) und „NEAR GOAL“(Ziel Entfernung). IN\_COLLISION ist wahr (true) wenn eine Kollision auftreten kann (s. Kreis basierte Kollisionsprüfung im Kapitel 4). NEAR GOAL ist wahr, wenn die Entfernung zum Ziel nah zur Null ist. Ein Maximum von 50 Tests wurde angenommen. Das heißt für Test0, i ist 0 und somit wird der String '/vehicle0/ENV/propositions'. Wenn der Name des Topics entspricht dem Tests, dann beginnt die Schleife (Zeile5).

Zuerst wurde die Monitor Klasse instanziert (Zeile8). Die aus der Kommandozeile angegebenen Daten wurden hier verwendet. Als nächstes

wurde einer Datenrahmen aus der Bag Datei anhand des Topicnamens erzeugt (Zeile9). IN\_COLLISION und NEAR GOAL wurden in separaten Datenrahmen (s. Abbildung5.3) sortiert (Zeile 11,12).

The diagram illustrates the extraction of a DataFrame **df** into two separate DataFrames, **df1** and **df2**. The original DataFrame **df** contains columns: Time, order, term, value, timeout, time, and has\_timeout. A red box highlights the 'label' column, which contains values like 'NEAR\_GOAL' and 'IN\_COLLISION'. Another red box highlights the 'term' column. These highlighted columns are mapped to the corresponding columns in **df1** and **df2**.

	Time	order	term	value	timeout	time	has_timeout
0	1.610486e+09	0	NEAR_GOAL	False	0.0	1.995	False
1	1.610486e+09	0	NEAR_GOAL	False	0.0	2.020	False
2	1.610486e+09	0	NEAR_GOAL	False	0.0	2.045	False
3	1.610486e+09	0	NEAR_GOAL	False	0.0	2.065	False
4	1.610486e+09	0	NEAR_GOAL	False	0.0	2.095	False
...	...	...	...	...	...	...	...
3420	1.610486e+09	0	IN_COLLISION	False	0.0	56.170	False
3421	1.610486e+09	0	NEAR_GOAL	True	0.0	56.170	False
3422	1.610486e+09	0	NEAR_GOAL	True	0.0	56.170	False
3423	1.610486e+09	0	IN_COLLISION	False	0.0	56.170	False
3424	1.610486e+09	0	NEAR_GOAL	True	0.0	56.170	False

	label	term	value	timeout	time	has_timeout
86e+09	0	IN_COLLISION	False	0.0	3.665	False
86e+09	0	IN_COLLISION	False	0.0	3.695	False
86e+09	0	IN_COLLISION	False	0.0	3.745	False
86e+09	0	IN_COLLISION	False	0.0	3.795	False
86e+09	0	IN_COLLISION	False	0.0	3.805	False
...	...	...	...	...	...	...
86e+09	0	IN_COLLISION	False	0.0	56.170	False
86e+09	0	IN_COLLISION	False	0.0	56.170	False
86e+09	0	IN_COLLISION	False	0.0	56.170	False
86e+09	0	IN_COLLISION	False	0.0	56.170	False
86e+09	0	IN_COLLISION	False	0.0	56.170	False

	label	term	value	timeout	time	has_timeout
0	1.610486e+09	0	NEAR_GOAL	False	0.0	False
1	1.610486e+09	0	NEAR_GOAL	False	0.0	False
2	1.610486e+09	0	NEAR_GOAL	False	0.0	False
3	1.610486e+09	0	NEAR_GOAL	False	0.0	False
4	1.610486e+09	0	NEAR_GOAL	False	0.0	False
...	...	...	...	...	...	...
2193	1.610486e+09	0	NEAR_GOAL	True	0.0	False
2194	1.610486e+09	0	NEAR_GOAL	True	0.0	False
2195	1.610486e+09	0	NEAR_GOAL	True	0.0	False
2196	1.610486e+09	0	NEAR_GOAL	True	0.0	False
2197	1.610486e+09	0	NEAR_GOAL	True	0.0	False

Abbildung 5.3: Extraktion df in df1 und df2

Zwei Kripke Strukturen wurden für die beiden Datenrahmen generiert (Zeile 13,14). Für die Erstellung einer Kripke-Struktur  $K = (\mathbf{S}, \mathbf{I}, \mathbf{R}, \mathbf{L})$  aus diesem Datenrahmen sollten folgende Elemente betrachtet werden:

- **S** = Zustände(States) entsprechen den Zeilen im Datenrahmen.
- **I** = Initialzustand entspricht der ersten Zeile.
- **R** = Übergangsrelation(transitions relation) entspricht dem Übergang zwischen den Zeilen.
- **L** = Label-Funktion entspricht der Verbindung der Zeilen mit den zugehörigen atomaren Aussagen.

In df1 (s. Abbildung5.3), wenn IN\_COLLISION wahr ist, wird ein  $\{p\}$  gesetzt, sonst ein  $\{\text{not } p\}$ . Die Abbildung7.26 zeigt die erstellte Kripke-Struktur aus dem Datenrahmen df1 als Ausgabe des „Jupyter Notebook<sup>14</sup>“. In df2 (s. Abbildung5.3), wenn NEAR GOAL wahr ist, wird ein  $\{q\}$  gesetzt, sonst ein  $\{\text{not } q\}$ . Die Abbildung7.27 zeigt die erstellte Kripke-Struktur aus dem Datenrahmen df2 als Ausgabe des „Jupyter Notebook“.

Die Größe der Kripke-Struktur ( $\text{size}=2197$ , s. Abbildung7.27) entspricht der Größe der Tabelle (df2:2197, s. Abbildung5.3). S0 (Initial State)

<sup>14</sup>ermöglicht Python Programme in einzelnen Blöcke zu testen

wird in „set of States“ direkt integriert. Falls der Datenrahmen leer war oder keine relevante Daten enthielt, wurde an der Stelle gestoppt und zum nächsten Topic gesprungen (Zeile 15-20). Um einen Modell gegen eine Formel zu prüfen, sollte eine Formel die Spezifikation des Systems darstellen. Die folgende Abbildung 5.4 zeigt, die Spezifikation unseres Systems.

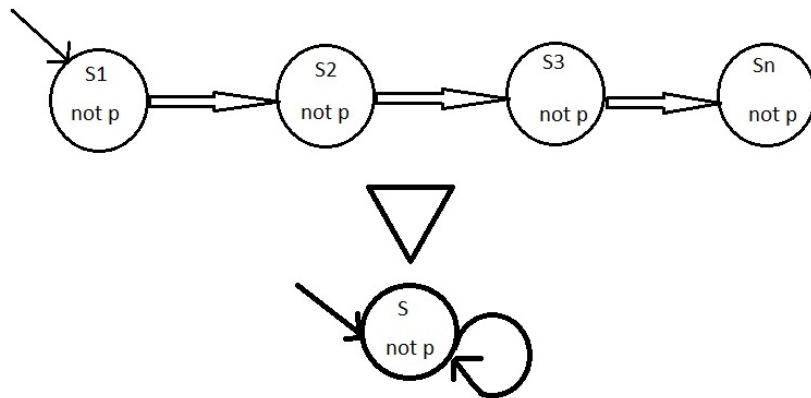


Abbildung 5.4: Spezifikation des Systemmodells

Entlang des Szenarios sollte keine Kollision auftreten. Daher sollte IN\_COLLISION entlang des gesamten Pfades (Datenrahmen df1) immer falsch sein{not p}. In Python (Zeile 22) lautet die Formel:

$$\phi = A(G(\text{Not}(p))) \quad (5.3)$$

mit  $A \equiv \forall$  (für alle),  $G \equiv \square$  (global) und  $\text{Not} \equiv \neg$  (Negation) in CTL\* Syntax. Mit Aufruf der Methode „CheckModel“ (Zeile 24) wurde die Kripke-Struktur mit der Formel 5.3 geprüft. Die Methode prüft, ob bei jedem Zustand die Formel erfüllt werden könnte. Wenn die Formel erfüllt wurde, ergibt sich ein „SET<sup>15</sup>“ mit allen Zuständen. Sonst wird ein leer SET() zurückgegeben. Somit kann die Methode „len()“ hiermit verwendet werden:

```
1 if len(m) == len(K.labelling_function())
```

Listing 5.2: CheckModel Methode

In Jupyter 5.3 wurde die Ausführung hiermit gezeigt:

```
1 phi = M.GetFormula(IN_COLLISION, p)
2 print('The No Collision Formula is: ', phi)
```

---

<sup>15</sup>in python SET ist eine Sammlung, die sowohl ungeordnet als auch unindiziert ist

```

3 M1=M.CheckModel(K1, phi)
4 print(len(M1))
5 print(M1)

```

Listing 5.3: Model Checking IN COLLISION

Das Resultat (s. Abbildung5.5) zeigt dass die Kripke-Struktur die Formel5.3 erfüllt und hat als Rückgabe eine SET mit 1227 Elementen (gleiche Größe wie df1):

```

The No Collision Formula is: A(G(not p))
The Kripke Structure satisfies the defined formula, saving test result
1227
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
9, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
14, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192,

```

Abbildung 5.5: Resultat Model Checking K1

Analog dazu wurde das NEAR GOAL Kriterium geprüft.

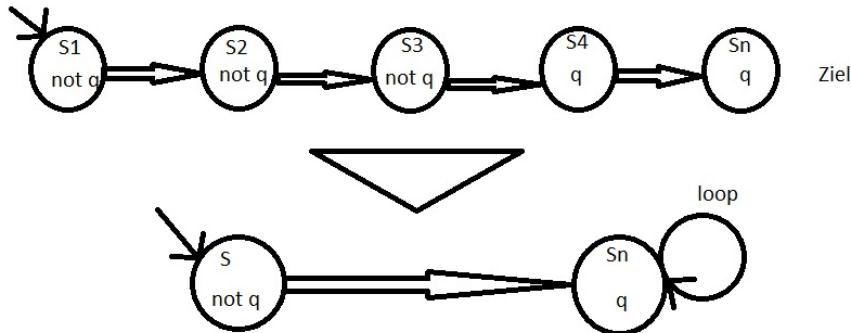


Abbildung 5.6: Near Goal Model

Die Abbildung5.6 zeigt den Graphen für Near Goal. Es wird erwartet, dass das Ego am Ziel ankommt. Das heißt, anfangs bleibt NEAR GOAL falsch  $\{\text{not } q\}$ , bis sich das Ego dem Ziel annähert(einige Metern entfernt), dann wird NEAR GOAL auf Wahr gesetzt bis das Ego das Ziel erreicht hat. Die Formel für dieses Modell lautet:

$$\psi = A(E(G(F(q)))) \quad (5.4)$$

Die Formel5.4 besagt, dass für alle Pfade mindestens ein Pfad existiert, sodass q unendlich oft (infinitely often) wahr ist. In anderen Wörtern bedeutet es, dass, sobald q wahr ist, dann bleibt es immer wahr. Zwei Gegebenfälle können daraus entstehen. Einerseits kann q nie wahr werden, wenn das Ego nie am Ziel ankommt oder nie gestartet ist. Andererseits kann q irgendwann wahr werden und dann wieder falsch, wenn das Ego am Ziel vorbeigefahren ist. Das Resultat des unten dargestellten Blocks ist in die Abbildung5.7 zu sehen:

```

1 psi = M.GetFormula(NEAR_GOAL, q)
2 print('The Goal Achieving Formula is: ', psi)
3 M2=M.CheckModel(K2, psi)
4 print(len(M2))
5 print(M2)

```

Listing 5.4: Model Checking NEAR GOAL

Die Kripke-Struktur erfüllt die Formel5.4, als Rückgabe ist keine leere SET(), sondern eine SET mit 2198 Elementen (gleiche Größe wie die df2):

```

The Goal Achieving Formula is: A(E(G(F(q))))
The Kripke Structure satisfies the defined formula, saving test result
2198
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
9, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
14, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,

```

Abbildung 5.7: Resultat Model Checking K2

- Topic /vehicleX/VEH/ax: enthält Nachrichten für die Ego-Beschleunigung/das Verzögerungsverhalten (s. folgender Code).

```

1
2 elif topic == '/vehicle'+str(i)+'/VEH/ax':
3     print('Analysing second topic:', topic)
4     N = Monitor(self.inputfile, self.outputfile, topic)
5     df = N.ReadBagFileByTopic(self.inputfile, topic)
6     ax_df = (N.DataFrameExtractByTerm(df, 'data', None)).rename(
    columns={"data": "Ego_ax"})

```

Listing 5.5: class Analyser: Topic 2

Daten aus diesem Topic werden extrahiert und in einem Datenrahmen sortiert. Die gewonnenen Daten werden für nächsten Topics bereitgestellt.

- Topic /vehicleX/odom (s. Code7.6): enthält Nachrichten für die Ego-Odometrie. Daten für die Position und die Geschwindigkeit können damit gewonnen werden. Daten zur Position (x,y) und Geschwindigkeit

$(x,y)$  werden in einem Datenrahmen zusammengefügt (Zeile 5-16). Die maximale Geschwindigkeit wurde geprüft und 15 m/s bzw. 54 km/h wird als Maximum gesetzt. Wenn das Ego über dieses Limit gefahren ist, wird  $\{\text{not } v\}$  gesetzt, anderenfalls ein  $\{v\}$ . Die Kripke-Struktur wird dann erstellt (Zeile 17). Das Modell (s. Abbildung 5.8) wird mit folgender Formel geprüft:

$$taw = A(G(v)) \quad (5.5)$$

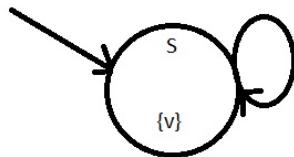


Abbildung 5.8: Model Checking Max Velocity

Das Resultat kann mittels „Jupyter“ überprüft werden:

```

1 Analysing third topic: /vehicle0/odom
2 [INFO] Data folder ../Bagfiles/test_ci_1_old already exists. Not
       creating.
3 Checking Speed limit..
4 Kripke Structure created.
5 The Speed Limit Formula is: A(G(v))
6 The Kripke Structure satisfies the defined formula, saving test result

```

Listing 5.6: result max velocity

- Topic /vehicleX/traffic (s. Code 7.7): enthält Nachrichten für den gesamten Verkehr (Ego und SUMO Fahrzeuge). Die Position und die Geschwindigkeit der SUMO Fahrzeuge wurden sortiert. Zusammen mit den vorherigen Ego-Daten können Kriterien wie Sicherheitsabstand in Quer- und Längsrichtung geprüft werden. Dazu könnten Beschleunigung und Verzögerungsraten des Egos in Bezug auf den Status der Blinker, Bremse und andere Faktoren ermittelt werden. Da die Daten ein Json-Format haben, werden diese zuerst konvertiert (Zeile 8-12) und mit vorherigen Topics-Daten in einem Datenrahmen (Zeile 14) zusammengestellt.

Der Längssicherheitsabstand (Zeile 16-25) wurde zuerst geprüft. Falls der Abstand zwischen Ego und POVs kleiner als der Sicherheitsabstand

ist, wird ein  $\{x\}$  gesetzt, ansonsten ein  $\{\text{not } x\}$ . Beim Modell (s. Abbildung 5.9) wurde geprüft, ob der Sicherheitsabstand entlang der Fahrt eingehalten wird. Die Formel lautet:

$$\omega = A(G(x)) \quad (5.6)$$

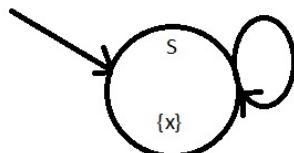


Abbildung 5.9: Sicherheitsabstand (Längs) Modell

Ähnlich dazu, wird der Querabstand geprüft. Das Modell (s. Abbildung 5.10) wurde somit geprüft. Falls der seitliche Abstand zwischen Ego und POVs kleiner als der Sicherheitsabstand ist, wird ein  $\{y\}$  gesetzt, ansonsten ein  $\{\text{not } y\}$ . Die Formel lautet:

$$\gamma = A(G(y)) \quad (5.7)$$

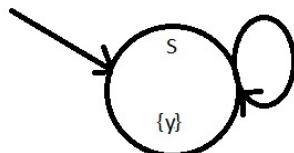


Abbildung 5.10: Sicherheitsabstand (seitlich) Modell

Als letztes wird die Verzögerungsrate des Egos geprüft (s. Abb 5.11). Das bedeutet, wenn das Bremslicht des vorderen Autos an ist, sollte das Ego bremsen. Wenn die Differenz zwischen aktueller und alter

Beschleunigung negativ ist, wird  $\{a\}$  gesetzt, ansonsten  $\{\text{not } a\}$ . Die Formel lautet:

$$\lambda = A(G(a)) \quad (5.8)$$

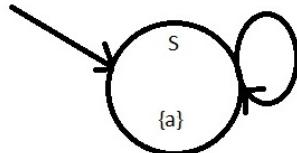


Abbildung 5.11: Verzögerungsrate Modell

Eine Bag-Datei wird als Beispiel im Code5.7 gezeigt:

```
1 python3 ModelChecker.py -i ../Bagfiles/test_ci_1_old.bag -o outfile.txt -t
   /vehicle0/ENV/propositions /vehicle0/VEH/ax /vehicle0/odom /vehicle0/
   traffic
```

Listing 5.7: Beispiel1: Auswertung einer Bag-Datei

Das Resultat wird im Code7.8 gezeigt. Das oben gezeigte Beispiel hat die Prüfung bestanden. Es gab keine Kollision (Zeile 10) und das Ego ist am Ziel angekommen (Zeile 12). Das Ego hat die Grenzgeschwindigkeit eingehalten (Zeile 20). Verkehrsdaten (Zeile 25) fehlten und wurden daher übersprungen. Das Gesamtergebnis ist positiv.

Ein weiteres Beispiel wird unten im Code5.8 gezeigt:

```
1 python3 ModelChecker.py -i ../Bagfiles/test_ci.bag -o outfile.txt -t /
   /vehicle0/ENV/propositions /vehicle0/VEH/ax /vehicle0/odom /vehicle0/
   traffic
```

Listing 5.8: Beispiel2: Auswertung einer Bag-Datei

Das Resultat befindet sich im Code7.9. Das oben gezeigte Beispiel hat die Prüfung nicht bestanden. Es gab mögliche Kollision (Zeile 10) und das Ego ist nicht am Ziel angekommen (Zeile 12). Das Ego hat nicht die Grenzgeschwindigkeit eingehalten (Zeile 20). Sicherheitsabstände(Längs und seitlich) wurden nicht respektiert (Zeile 29, 33). Daten für Bremslichtstatus wurden nicht gefunden. Das Gesamtergebnis ist negativ.

Der komplette Quellcode für das Evaluierungsskript befindet sich im Code7.10. Im folgenden Kapitel wird das Resultat der automatischen Ausführung und Auswertung der erstellten Testszenarien dargestellt.

# **Kapitel 6**

# **Ergebnisse**

In diesem Kapitel wird das Resultat der CI-Pipeline gezeigt. Die Pipeline wurde anhand der Konfiguration der yaml-Datei erstellt. Zuerst wird ein Docker-Container erzeugt. Im Inneren des Containers wird ADORe installiert. Dann werden Testszenarien automatisch ausgeführt und dann ausgewertet. Das Ergebnis wird im Gitlab-CI Server hochgeladen und der Docker-Container wird am Ende gelöscht.

## **6.1 Zusammenfassung**

Mit dem Befehl „git push“ wird die Pipeline erstellt. Die folgende Abbildung 6.1 zeigt die Pipeline direkt nach dem „Push“ Befehl:

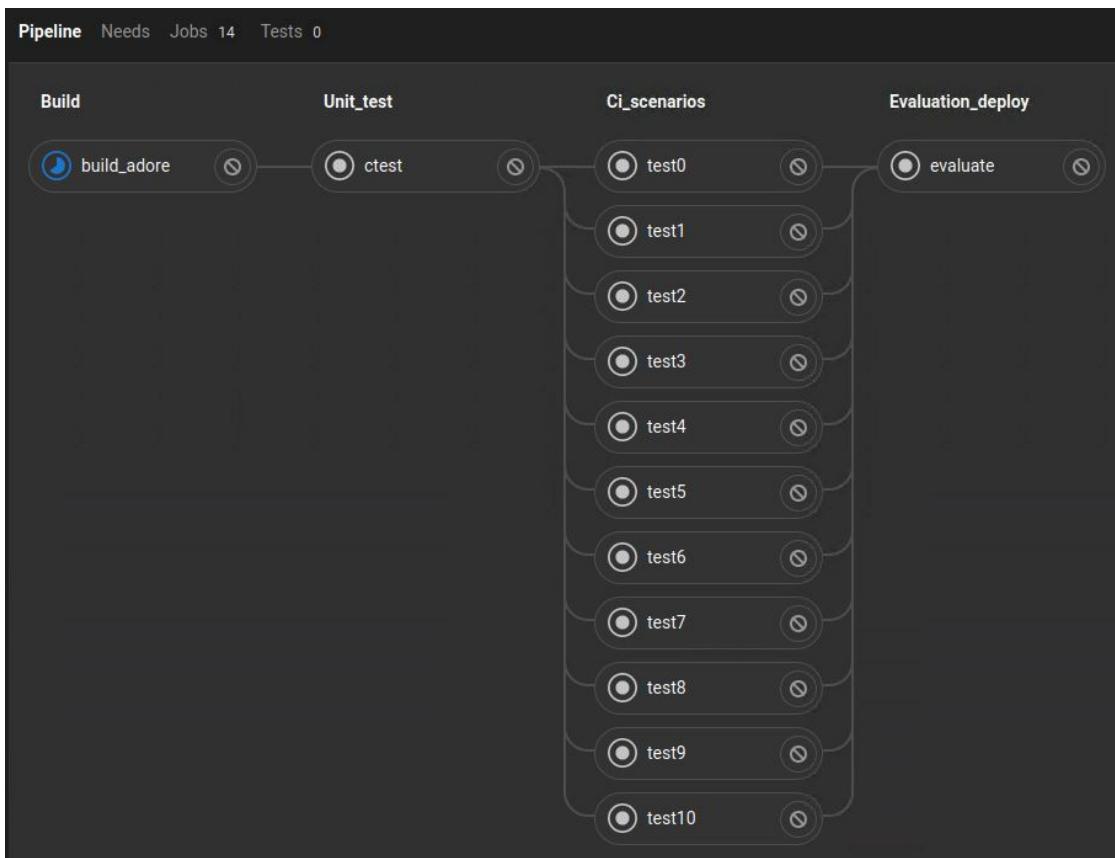


Abbildung 6.1: CI Pipeline zum Beginn des Prozesses

Zuerst wird der „`build_adore`“ Job unter der Build Stage laufen (blau markiert). Die grau-markierten Jobs sind noch nicht ausgeführt. Am Ende der Ausführung ist die Pipeline beendet. Die folgende Abbildung 6.2 zeigt die fertige Pipeline:

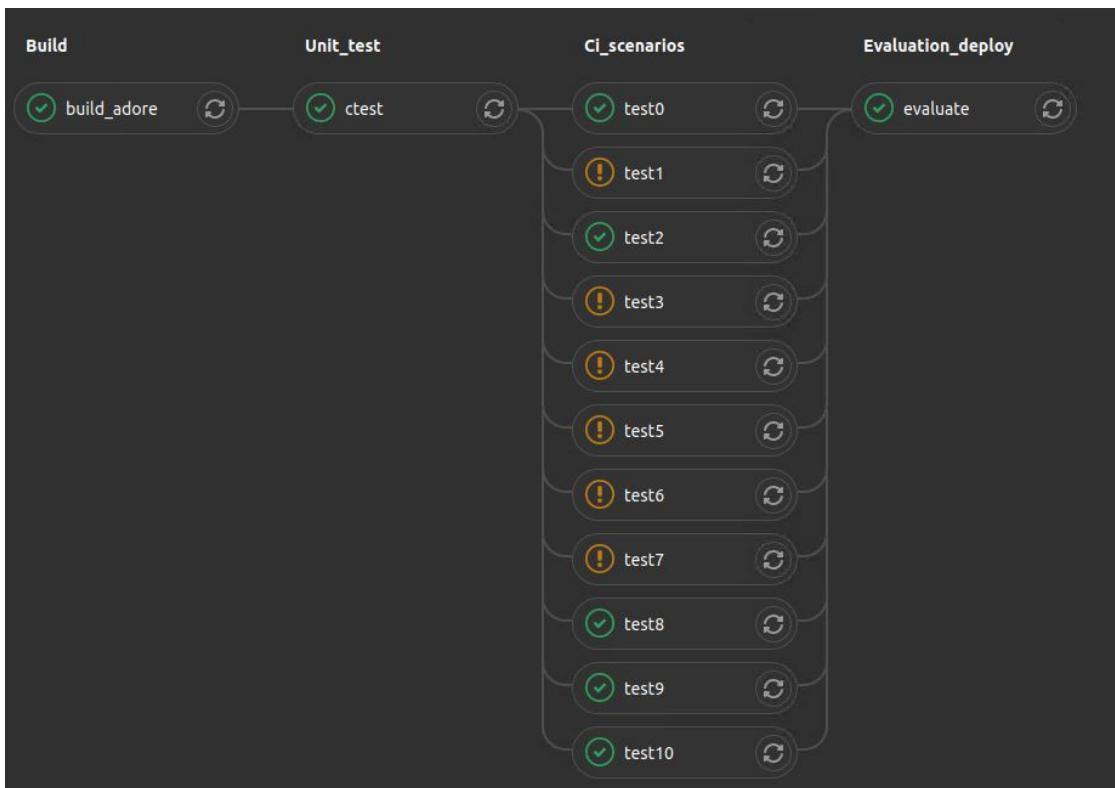


Abbildung 6.2: CI Pipeline zum Ende des Prozesses

Anhand der Abbildung ist erkennbar, dass erfolgreiche Jobs mit grünen Häkchen markiert sind. Jobs, die mit gelben Ausrufezeichen markiert sind, sind fehlgeschlagene Jobs mit der Konfiguration, dass die Pipeline weitergeführt werden sollte. Das heißt, Jobs in der Ci-Scenarios Stage dürfen scheitern und die CI-Pipeline wird bis zur letzten Stage durchgeführt.

Diese besondere Konfiguration hilft beim Debuggen für fehlgeschlagene Jobs. Bestandene Tests ( $\{0,2,8,9,10\}$ ) werden in einer Datei gespeichert. In letzter Stage wurden die erfolgreichen Tests (s. Abbildung 6.3) aus der Datei gelesen:

```
$ cat $SharedVol/adore/build/variables | while read line || [[ -n $line ]]; do echo "successful test's name:$line"; done
successful test's name:test0
successful test's name:test2
successful test's name:test10
successful test's name:test9
successful test's name:test8
```

Abbildung 6.3: Ergebnis der Pipeline

## 6.2 Debuggen

Die fehlgeschlagenen Tests sollten einzeln im weiteren Schritt debuggt werden. Anhand der Ergebnisse der automatisierten Auswertung können Informationen zur Fehlerursache unter dem jeweiligen Topic (s. Kapitel Auswertung) verwendet werden. Der „test3“ im CI wird als Beispiel für das Debuggen gezeigt:

```
1 The No Collision Formula is: A(G(not p))
2 The Kripke Structure satisfies the defined formula, saving test result
3 The Goal Achieving Formula is: A(E(G(F(q))))
4 The Kripke Structure doesn't satisfy the defined formula, saving test result
5 Analysing second topic: /vehicle3/VEH/ax
6 [INFO] Data folder /home/outputdir/build/test_ci_3 already exists. Not
      creating.
7 Analysing third topic: /vehicle3/odom
8 [INFO] Data folder /home/outputdir/build/test_ci_3 already exists. Not
      creating.
9 Checking Speed limit..
10 Kripke Structure created.
11 The Speed Limit Formula is: A(G(v))
12 The Kripke Structure satisfies the defined formula, saving test result
13 checking overall test Result..
14 test failed, Result saved to: /home/outputdir/build/test_ci_3.txt
```

Listing 6.1: Debuggen „test3“

Im „test3“ gab es keine Kollision (Zeile 2). Das Ego ist nicht am Ziel angekommen (Zeile 4) und es ist unter der maximalen Geschwindigkeit gefahren (Zeile 12).

Der „test3“ im CI entspricht dem „test003\_lane\_change.launch“ mit dem graphischen Interface. Zum Debuggen werden der Plotter und die Launch-Datei gestartet und die Nachrichten werden in Echtzeit mit folgendem Befehl gezeigt:

```
1 cd ~/catkin_ws/src/adore/plotlabserver/;./stop.sh;./start.sh;roslaunch ~/catkin_ws/src/adore/test/catalog_from_nhtsa/test003_lane_change.launch
2 rostopic echo -p /vehicle3/ENV/propositions
```

Listing 6.2: Debuggen

Dabei sollen die graphische Simulation und die Nachrichten geprüft werden. Alternativ dazu kann die Bag Datei vom „test3“ (als Artefakt auf dem CI Server gespeichert) wiedergegeben werden und die Nachrichten mit folgendem Befehl mitgezeigt werden. Roscore sollte hier zuerst gestartet werden:

```
1 roscore&
2 rostopic echo -b test_ci_3.bag /vehicle3/ENV/propositions
```

Listing 6.3: Debuggen „test3“ zweite Variante

Nach dem Debuggen hat sich ergeben, dass das Ego nicht am Ziel stoppt und weiterfährt. Dieses Verhalten wurde in anderen fehlgeschlagenen Tests festgestellt. Als Schlussfolgerung lässt sich ableiten, dass das Spurwechsel-Verhalten (Node) einen Konflikt mit der Ziellocation (Node) zeigt. Das Problem wurde an das zuständige Team gemeldet. Da das Spurwechsel-Verhalten noch in Entwicklung ist, wurde das Problem an der Stelle nicht weiter erforscht.

Ein weiteres merkwürdiges Verhalten wurde bei anderen CI-Pipelines erkannt. Einige Jobs haben den voreingestellten Timeout von 60 Minuten überschritten (s. Abbildung6.4).

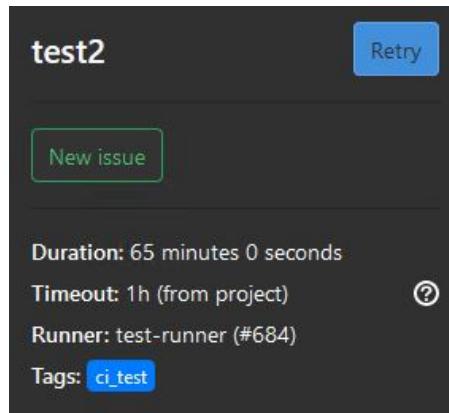


Abbildung 6.4: Überschreitung Timeout Jobs in CI-Pipeline

Nach dem Debuggen hat sich ausgestellt, dass die Launch-Datei nicht richtig beendet wird. Obwohl ein Timeout von 60 Sekunden via Terminator-Node in der Launch-Datei (s. Code4.10) eingestellt war, hat die Launch-Datei diese Zeit überschritten. Um diese Fehler zu beheben, wird ein zusätzlicher Timeout außerhalb der Launch-Datei eingesetzt. Im Testszenario Skript (s. Code4.8) wird die Zeile 10 wie folgt geändert:

```
1 timeout 120 roslaunch -p 11350 --wait test000_template.launch > /home/  
    outputdir/build/test_ci.log
```

Listing 6.4: Global Timeout

Ein globales Timeout von 120s wird bei der Ausführung von der Launch-Datei eingesetzt. Die Wahl vom Maximum 120s liegt an der unterschiedlichen internen Timeouts bei jedem Szenario.

Ein letztes seltsames Verhalten wurde bei der parallelen Ausführung von Jobs in CI-Pipelines erkannt. Jobs in gleicher Stage sollten gleichzeitig vom Runner ausgeführt werden. Es hat sich ergeben, dass nicht alle Jobs tatsächlich gleichzeitig ausgeführt sind. Die Abbildung6.5 zeigt die CI-Pipeline während der „ci\_scenarios“ Stage:

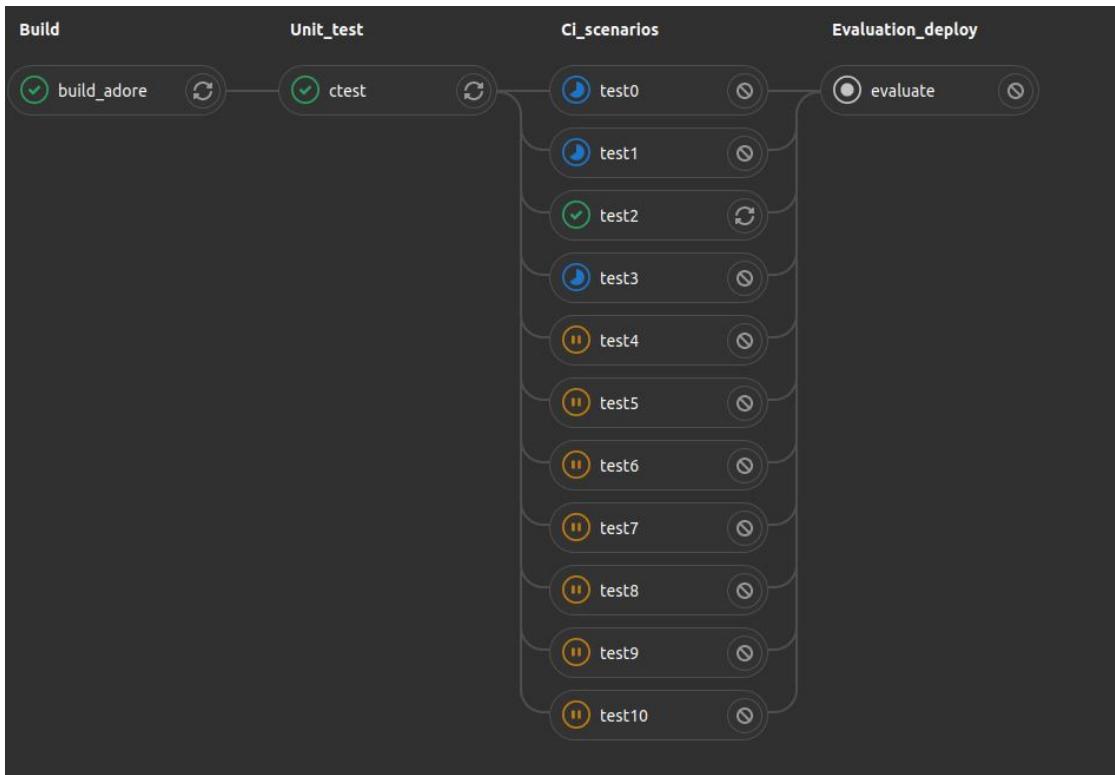


Abbildung 6.5: Parallelle Jobs in CI-Pipeline

Die Abbildung 6.5 zeigt, dass nur vier von elf Jobs tatsächlich parallel ausgeführt worden sind (Blau und Grün markiert). Die sieben restlichen Jobs (gelb markiert) sind pausiert. Es wird vermutetet, dass dieses Problem an der CPU<sup>1</sup>-Leistung hängt. Die CI-Pipeline läuft auf dem Docker-Container, der auf einer VM<sup>2</sup> mit nur vier freigegebenen Kernprozessoren ausgestattet wurde. Das könnte der Grund sein, dass nur maximal vier Jobs gleichzeitig ausführbar sind. Um dieses Problem zu untersuchen, sollte die Pipeline auf einem Server mit höherer CPU-Leistung ausgeführt werden. Der Aufwand zur Bereitstellung eines Servers (speziell für diese Anwendung) ist im Rahmen dieser Arbeit nicht vorgesehen.

Zur Problemumgehung mit verfügbaren Ressourcen müssen die Testszenarien in Reihe ausgeführt werden. Diese Variante ist zeitlich ungünstig. Falls mehrere Testszenarien (Jobs), beispielsweise 50 Tests nacheinander ausgeführt werden, wird die Zeit summiert und die CI-Pipeline wird sehr lange dauern. Wenn ein Test (Job) für die automatische Ausführung und Auswertung fünf Minuten im Durchschnitt braucht, wird die Zeit für 50 Tests vier

---

<sup>1</sup>Prozessor

<sup>2</sup>virtuelle Maschine

Stunden dauern. solch eine Lösung ist nicht akzeptabel da im Prinzip mehrere hunderte Tests durchgeführt werden sollen.

# **Kapitel 7**

## **Fazit**

Zusammengefasst wurde in dieser Masterarbeit zuerst die Motivation an das Thema "Simulative Absicherung vernetzter automatisierter Fahrzeuge" gezeigt und der eigene investierte Beitrag deklariert (s. Kapitel 1). Im Kapitel 2 wurde der Stand der Technik der automatisierten Fahrzeuge erforscht. Dabei wurde das Zeitdiagramm (s. Abb2.1) der Entwicklung von automatisierten Fahrzeugen, die Taxonomie (L0 bis L5) nach SAE (s. Abb2.2) und die AFs-spezifische Begriffe definiert. Das automatisierte Fahrzeug als Paket (Sensorik, Hardware, Software und Kommunikationsprotokolle V2x) wurde im zweiten Kapitel untersucht.

Im dritten Kapitel wurden registrierte Unfälle als Beweise der Kritikalität der Sicherheit von AFs erwähnt. Automotive Sicherheit Frameworks (ISO26262, ISO/PAS 21448.1, NHTSA und PEGASUS) wurden zusammengefasst. NHTSA wurde als Konzept für die Erstellung grundlegender Testszenarien angenommen. PEGASUS wurde als inspirierendes Konzept für die automatische Ausführung und Auswertung von Testszenarien verwendet.

Im vierten Kapitel wurde das ADORe Framework und die zugehörigen Tools (Ros, Sumo, Gitlab und Docker) beschrieben. Ein Demo-Beispiel aus den Testszenarien der ADORe Github Version wurde gezeigt. Eine neue Methodik zur Erstellung von Testszenarien (s. Tabelle7.7) wurde implementiert. Mehrere Szenarien wurden manuell erstellt und getestet. Mittels Gitlab-Ci wurden Tests automatisch ausgeführt.

Im fünften Kapitel wurde die Auswertung von Szenarien anhand der Modellprüfung<sup>1</sup> Technik erklärt. Die Umsetzung wurde mittels Python Skript durchgeführt. Das Ergebnis der automatischen Ausführung und Auswertung im CI-Pipeline wurde im sechsten Kapitel erkundet. Letztendlich wurden Debug-Methoden untersucht.

---

<sup>1</sup>Model Checking

Im Rahmen dieser Arbeit wurden folgende Aufgaben erfüllt:

- Recherche zum Stand der Technik der automatisierten Fahrzeuge
- Recherche von Normen und Standards zur Absicherung automatisierter Fahrzeuge
- Anwendung der Kombination aus der NHTSA und PEGASUS Frameworks als Lösungskonzept und Implementierung dieses Konzepts im ADORe Framework.
- Erstellung ausgewählter Testszenarien mit graphischem Interface
- Automatische Ausführung der Testszenarien im Gitlab-CI (Headless Mode)
- Anwendung der Model-Checking Technik, um Tests auszuwerten
- Automatische Auswertung der Testszenarien
- Analyse der Ergebnisse und Debuggen Methoden

Das Gesamtergebnis aller Aufgaben erweist sich als sehr zufriedenstellend. Die Aufgabenstellung (s. Aufgabenblatt) ist mit dem Erwerb der notwendigen Software-Kenntnisse erfüllt worden. Das Ergebnis des letzten Kapitels (6) ist von der Entwicklung der internen Funktionen des ADORe-Frameworks und den verfügbaren Ressourcen abhängig. Eine Optimierung bzw. Erweiterung dieser Arbeit lässt sich im nächsten Schritt erklären.

## 7.1 Ausblick

Im Rahmen dieser Arbeit wurden folgende Punkte nicht betrachtet und können als Erweiterung dieser Arbeit angenommen werden:

- a- Extraktion von Daten für die Szenarienerstellung aus bestimmten Quellen
- b- Erstellung einer Datenbank von Szenarien in einem standardisierten Format
- c- Automatische Generierung von Testszenarien
- d- Zugabe von Fehlern/Agenten, um die Grenze des Systems und entsprechende Funktionen zu testen
- e- Verwendung eines open-source-Simulators, z.b. Carla Simulator für bessere Auflösung und Nutzbarkeit

Punkt a wurde bereits im PEGASUS Projekt[10] erwähnt. Daten können aus verschiedenen Quellen erfasst werden. Verkehrsdaten werden aus Quellen wie FOT<sup>2</sup>, NDS<sup>3</sup> und Autounfalldatenbanken generiert. Diese sind meistens von Anbietern online verfügbar. Verschiedene Methoden sind heutzutage verfügbar, um Daten aus dem Internet zu sammeln. Als Beispiel, Python bietet die sogenannten „Web Scraping“ Tools, um in Webseiten Daten zu suchen und zu manipulieren. Die Schwierigkeit hier liegt in der Übersetzung von Daten in ein konformes Format, die für die Szenarienerstellung nutzbar ist. PEGASUS hat eine Methode beschrieben, um Daten zu übersetzen. Eine ausführliche Beschreibung befindet sich in der Dokumentation[10]. Die Abbildung7.11 zeigt das Pegasus-Prinzip.

Punkt b wurde ebenfalls in PEGASUS[10] erwähnt. Eine konkrete Implementierung ist nicht vorhanden. Das Ziel ist die Sammlung aller Szenarien in einer Datenbank und dies online verfügbar zu machen. Das kann dazu beitragen, dass Entwickler Szenarien aus dieser Datenbank für die Entwicklung nutzen können und deren Resultat bzw. neue Szenarien zu dieser Datenbank hinzufügen. Durch eine Zusammenarbeit wird die Entwicklung autonomer Fahrzeuge weltweit beschleunigt.

„CommonRoad“[84, 85] ist einer sehr gutes Beispiel für eine online-Datenbank mit bereits verfügbaren 2307 Testszenarien.

Punkte c ist auch aus dem PEGASUS[10] inspiriert worden. Mit mehreren zur Verfügung stehenden Daten können mehrere Testszenarien automatisch generiert werden. Die Abbildung7.13 zeigt das Pegasus-Prinzip der stochastischen Variation. Ein „Test Automation Tool“ (s. Abbildung7.13) kann dabei helfen ein großes Parameterspektrum zu explorieren und dementsprechend automatisch Testfälle zu erstellen. Im ADORe kann solches Prinzip auch integriert werden. Beispielsweise können Tabellen automatisch mit Parametern ausgefüllt werden. Ein Programm liest die Werte aus der Tabelle und generiert dementsprechend Launch-Dateien mit neuen Parametern. Die Launch-Dateien werden automatisch ausgeführt und ausgewertet. Erfolgreiche Szenarien werden weiter parametrisiert, fehlgeschlagene Szenarien werden untersucht.

Punkt d wurde im NHTSA-[43] und Waymo-Sicherheit-[63, 5] Frameworks beschrieben. Die NHTSA hat das vierte Dimension eines Testszenarios als „Fehlermodi“(s. auch Tab7.8) definiert. Wodurch können Fehler in Tests injiziert werden um die Reaktion des Systems zu untersuchen. Es könnten beispielsweise Signalfehler wie Sensor-rauschen in Szenarien simuliert werden[43]. Zusätzlich verwendet Waymo das sogenannte Fuzzing[5](s. Abb7.9),

---

<sup>2</sup>Field Operation Testing

<sup>3</sup>naturalistic driving study

kombiniert mit der Stärke der neuronalen Netze, um Stufen von Komplexität in Szenarien zu addieren. Zudem nutzt Waymo DeepMind[63] und Deep Reinforcement Learning[63], um Agenten und Policy<sup>4</sup> zu erstellen[63]. Waymo kann das Verhalten eines Fahrers beliebig ändern, z.b. kann das Verhalten eines wütenden oder unachtsamen Fahrers simuliert werden[63]. Beim Testen achten die Waymo-Entwickler besonders darauf, wie sich deren Algorithmus verhält und versuchen es zu verbessern. Wenn der Test bzw. der Algorithmus erfolgreich war, wird diese neue Fähigkeit erbracht und gespeichert und für das gesamte Netzwerk aller Waymo-Fahrzeuge verfügbar[63].

Der letzte Punkt e liegt bereits auf der Agenda von ADORe. Carla Simulator ist open-source-Simulator für die Entwicklung und Validierung autonomer Fahrsysteme[86]. Carla basiert auf „Unreal Engine<sup>5</sup>“ und hat deswegen eine sehr hohe Auflösung wie in den Videospielen. Carla unterstützt ROS als Schnittstelle, OpenDrive für Kartenerstellung, ScenarioRunner für Szenariendefinition, Sensoren-Virtualisierung, APIs<sup>6</sup> und mehr. Folgende Abbildung 7.1 stellt eine Aufnahme von Carla dar.

Der Spezialisierungskurs „Introduction to Self-Driving Cars“[12] bietet eine ausführliche Einleitung im Bereich des autonomen Fahren, wobei Carla für die Simulation verwendet wird.



Abbildung 7.1: Carla Simulator [87]

<sup>4</sup>Beim Reinforcement Learning ist eine Policy ein Verhalten[63]

<sup>5</sup>game engine

<sup>6</sup>application programming interface

# Literatur

- [1] *Fahrerassistenzsysteme*. URL: <https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/assistenzsysteme/fahrerassistenzsysteme> (besucht am 06.03.2021).
- [2] Anthony Freeman u.a. „The legacy of the SIR-C/X-SAR radar system: 25years on“. In: *Remote Sensing of Environment* 231 (2019), S. 111255. ISSN: 0034-4257. DOI: <https://doi.org/10.1016/j.rse.2019.111255>. URL: <https://www.sciencedirect.com/science/article/pii/S0034425719302743> (besucht am 17.09.2020).
- [3] Kalra Nidhi und Susan M. Paddock. „Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?“ In: (2016). ISSN: RR-1478-RC. URL: [https://www.rand.org/pubs/research\\_reports/RR1478.html](https://www.rand.org/pubs/research_reports/RR1478.html) (besucht am 17.09.2020).
- [4] Zhiyuan Huang u.a. „Accelerated Evaluation of Automated Vehicles Using Piecewise Mixture Models“. In: *IEEE Transactions on Intelligent Transportation Systems* PP (Jan. 2017). DOI: 10.1109/TITS.2017.2766172. URL: [https://www.researchgate.net/publication/313157554\\_Accelerated\\_Evaluation\\_of\\_Automated\\_Vehicles\\_Using\\_Piecewise\\_Mixture\\_Models](https://www.researchgate.net/publication/313157554_Accelerated_Evaluation_of_Automated_Vehicles_Using_Piecewise_Mixture_Models) (besucht am 17.09.2020).
- [5] waymo. „Waymo Safety Report“. In: (2020). URL: <https://storage.googleapis.com/sdc-prod/v1/safety-report/2020-09-22-safety-report.pdf> (besucht am 27.09.2020).
- [6] *Tesla safety report*. URL: [https://www.tesla.com/en\\_EU/VehicleSafetyReport](https://www.tesla.com/en_EU/VehicleSafetyReport) (besucht am 16.02.2021).
- [7] Brussels: European Commission. „FESTA Handbook Version 2 Deliverable T6.4 of the Field opErational teSt supporT Action“. In: (2008). URL: [https://repository.lboro.ac.uk/articles/report/FESTA\\_Handbook\\_version\\_2/9354539/1](https://repository.lboro.ac.uk/articles/report/FESTA_Handbook_version_2/9354539/1) (besucht am 17.09.2020).
- [8] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR). „Eclipse Automated Driving Open Research (ADORE)“. In: (2017). URL: <https://projects.eclipse.org/proposals/eclipse-automated-driving-open-research-adore> (besucht am 18.09.2020).

- [9] DLR. *Deutsches Zentrum für Luft- und Raumfahrt*. URL: [https://www.dlr.de/DE/Home/home\\_node.html](https://www.dlr.de/DE/Home/home_node.html) (besucht am 08.03.2021).
- [10] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR). „PEGASUS (Project for the Establishment of Generally Accepted quality criteria, tools and methods as well as Scenarios and Situations“. In: (2015). URL: <https://www.pegasusprojekt.de/files/tmp1/Pegasus-Abschlussveranstaltung/PEGASUS-Gesamtmethoden.pdf> (besucht am 18.09.2020).
- [11] Adnan Qayyum u. a. „Securing Connected Autonomous Vehicles: Challenges Posed by Adversarial Machine Learning and The Way Forward“. In: (Mai 2019). URL: <https://arxiv.org/pdf/1905.12762.pdf> (besucht am 20.09.2020).
- [12] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars by University of Toronto*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/home/welcome> (besucht am 20.09.2020).
- [13] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars by University of Toronto*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/supplement/AnLqz/glossary-of-terms> (besucht am 20.09.2020).
- [14] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week1, Lesson 1: Taxonomy of Driving*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/lecture/BdNR6/lesson-1-taxonomy-of-driving> (besucht am 20.09.2020).
- [15] Laurene Claussmann. „Motion Planning for Autonomous Highway Driving: A Unified Architecture for Decision-Maker and Trajectory Generator“. Diss. Okt. 2019. URL: [https://www.researchgate.net/publication/340479318\\_Motion\\_Planning\\_for\\_Autonomous\\_Highway\\_Driving\\_A\\_Unified\\_Architecture\\_for\\_Decision-Maker\\_and\\_Trajectory\\_Generator](https://www.researchgate.net/publication/340479318_Motion_Planning_for_Autonomous_Highway_Driving_A_Unified_Architecture_for_Decision-Maker_and_Trajectory_Generator) (besucht am 29.09.2020).
- [16] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week2, Lesson 1: Sensors and Computing Hardware*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/lecture/LrLty/lesson-1-sensors-and-computing-hardware> (besucht am 22.09.2020).
- [17] K.J. Bussemaker. *Sensing requirements for an automated vehicle for highway and rural environments*. Techn. Ber. Technische Universiteit Delft, 2014. URL: <https://repository.tudelft.nl/islandora/object/uuid:2ae44ea2-e5e9-455c-8481-8284f8494e4e?collection=education> (besucht am 22.09.2020).

- [18] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week1, Lesson 3: Driving Decisions and Actions*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/lecture/vPSnD/lesson-3-driving-decisions-and-actions> (besucht am 21.09.2020).
- [19] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week2, Lesson 3: Software Architecture*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/lecture/RQCqM/lesson-3-software-architecture> (besucht am 22.09.2020).
- [20] Dimosthenis Grillias. „Vehicle-to-Everything (V2X) communications Security and challenges of V2X communications“. In: *International Journal of Vehicle Safety* (Mai 2020). URL: [https://www.researchgate.net/publication/341609834\\_Vehicle-to-Everything\\_V2X\\_communications\\_Security\\_and\\_challenges\\_of\\_V2X\\_communications](https://www.researchgate.net/publication/341609834_Vehicle-to-Everything_V2X_communications_Security_and_challenges_of_V2X_communications) (besucht am 22.09.2020).
- [21] CISSP Lawrence Miller. „Connected Car For Dummies“. In: (2018). URL: [https://media.wiley.com/assets/7369/72/Connected\\_Car\\_FD\\_Qorvo\\_Special\\_Edition\\_9781119510901.pdf](https://media.wiley.com/assets/7369/72/Connected_Car_FD_Qorvo_Special_Edition_9781119510901.pdf) (besucht am 23.09.2020).
- [22] Hugues-Antoine Lacour Tom Rebbeck Janette Stewart u.a. „Final Report for 5GAA Socio-Economic Benefits of Cellular V2X 5GAA“. In: (2017). URL: [https://5gaa.org/wp-content/uploads/2017/12/Final-report-for-5GAA-on-cellular-V2X-socio-economic-benefits-051217\\_FINAL.pdf](https://5gaa.org/wp-content/uploads/2017/12/Final-report-for-5GAA-on-cellular-V2X-socio-economic-benefits-051217_FINAL.pdf) (besucht am 23.09.2020).
- [23] Jian Wang u.a. „A Survey of Vehicle to Everything (V2X) Testing“. In: *Sensors* 19 (Jan. 2019), S. 334. DOI: 10.3390/s19020334. URL: <https://www.mdpi.com/1424-8220/19/2/334/htm> (besucht am 23.09.2020).
- [24] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week 3 Lesson 1: Safety Assurance for Self-Driving Vehicles*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/lecture/gtsMT/lesson-1-safety-assurance-for-self-driving-vehicles> (besucht am 24.09.2020).
- [25] *Department of motor vehicles: Collision reports*. URL: <https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/> (besucht am 10.03.2021).
- [26] Neal E. Boudette. „Autopilot Cited in Death of Chinese Tesla Driver“. In: (2016). URL: <https://www.nytimes.com/2016/09/15/business/fatal-tesla-crash-in-china-involved-autopilot-government-tv-says.html> (besucht am 25.09.2020).
- [27] *National Transportation Safety Board*. URL: <https://www.ntsb.gov/Pages/default.aspx> (besucht am 10.03.2021).

- [28] NTSB: National Transportation Safety Board. „Collision Between a Car Operating With Automated Vehicle Control Systems and a Tractor-Semitrailer Truck Near Williston, Florida May 7, 2016“. In: (2016). URL: <https://www.ntsb.gov/investigations/AccidentReports/Pages/AccidentReports.aspx> (besucht am 25.09.2020).
- [29] Associated Press. „Google self-driving car caught on video colliding with bus“. In: (2016). URL: <https://www.theguardian.com/technology/2016/mar/09/google-self-driving-car-crash-video-accident-bus> (besucht am 25.09.2020).
- [30] Samuel Gibbs. „GM sued by motorcyclist in first lawsuit to involve autonomous vehicle“. In: (2018). URL: <https://www.theguardian.com/technology/2018/jan/24/general-motors-sued-motorcyclist-first-lawsuit-involve-autonomous-vehicle> (besucht am 25.09.2020).
- [31] NTSB. „PRELIMINARY REPORT HIGHWAY HWY18MH010“. In: (2018). URL: <https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf> (besucht am 25.09.2020).
- [32] *Functional Safety*. URL: <https://www.kpit.com/insights/functional-safety-fusa-in-automotive-industry> (besucht am 10.03.2021).
- [33] ISO/TC 22/SC 32. *ISO 26262-1:2018 Road vehicles – Functional safety*. Techn. Ber. ISO. URL: <https://www.iso.org/standard/68383.html> (besucht am 28.09.2020).
- [34] Sina Mohseni u. a. *Practical Solutions for Machine Learning Safety in Autonomous Vehicles*. Techn. Ber. Dez. 2019. URL: [https://www.researchgate.net/publication/338115926\\_Practical\\_Solutions\\_for\\_Machine\\_Learning\\_Safety\\_in\\_Autonomous\\_Vehicles](https://www.researchgate.net/publication/338115926_Practical_Solutions_for_Machine_Learning_Safety_in_Autonomous_Vehicles) (besucht am 28.09.2020).
- [35] *FAILURE MODE AND EFFECTS ANALYSIS (FMEA)*. URL: <https://asq.org/quality-resources/fmea> (besucht am 16.02.2021).
- [36] Bogdanovská Gabriela und Marcela Pavlickova. „Hazard and operability study“. In: Mai 2016, S. 82–85. DOI: 10.1109/CarpPathianCC.2016.7501071. (Besucht am 10.02.2021).
- [37] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week 3 Lesson 3: Safety Frameworks for Self-Driving*, Video-Transkript. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/learn/intro-self-driving-cars/lecture/hCttH/lesson-3-safety-frameworks-for-self-driving> (besucht am 28.09.2020).
- [38] Martin Hillenbrand. *Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen*. Techn. Ber. KIT. URL: <https://publikationen.bibliothek.kit.edu/1000025616/2325028> (besucht am 28.09.2020).

- [39] ISO/PAS 21448:2019. URL: <https://www.iso.org/standard/70939.html> (besucht am 10.02.2021).
- [40] National Highway Traffic Safety Administration. URL: <https://www.nhtsa.gov/> (besucht am 10.02.2021).
- [41] Steven Waslander Jonathan Kelly. *Introduction to Self-Driving Cars, Week 3 Lesson 2: Industry Methods for Safety Assurance and Testing*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/lecture/intro-self-driving-cars/lesson-2-industry-methods-for-safety-assurance-and-testing-Mn5cl> (besucht am 28.09.2020).
- [42] NHTSA. „AUTOMATED DRIVING SYSTEMS 2.0: A VISION FOR SAFETY“. In: (2016-2017). URL: [https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13069a-ads2.0\\_090617\\_v9a\\_tag.pdf](https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13069a-ads2.0_090617_v9a_tag.pdf) (besucht am 25.09.2020).
- [43] NHTSA. „A Framework for Automated Driving System Testable Cases and Scenarios“. In: (2018). URL: [https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13882-automateddrivingsystems\\_092618\\_v1a-tag.pdf](https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13882-automateddrivingsystems_092618_v1a-tag.pdf) (besucht am 07.10.2020).
- [44] ASAM. „ASAM OpenDRIVE®“. In: (). URL: <https://www.asam.net/standards/detail/opendrive/> (besucht am 09.02.2021).
- [45] OpenScenario. URL: <https://www.asam.net/standards/detail/openscenario/> (besucht am 16.02.2021).
- [46] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR). „Eclipse Automated Driving Open Research (ADORE)“. In: (2017). URL: <https://projects.eclipse.org/projects/technology.adore> (besucht am 08.02.2021).
- [47] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR). „Eclipse Automated Driving Open Research (ADORE)“. In: (2017). URL: <https://github.com/eclipse/adore> (besucht am 08.02.2021).
- [48] Robot Operating System. URL: <http://wiki.ros.org/Documentation> (besucht am 10.02.2021).
- [49] Create a ROS Driver Package. URL: <https://roboticsbackend.com/create-a-ros-driver-package-introduction-what-is-a-ros-wrapper-1-4/> (besucht am 10.02.2021).
- [50] ROS Wiki technical overview. URL: <http://wiki.ros.org/ROS/Technical200verview> (besucht am 10.02.2021).
- [51] Simulation of Urban MObility. URL: <https://sumo.dlr.de/docs/index.html> (besucht am 10.02.2021).
- [52] „OpenStreetMap“. In: (). URL: <https://www.openstreetmap.org> (besucht am 09.02.2021).

- [53] *Simulation of Urban MObility: Tutorials*. URL: [https://sumo.dlr.de/docs/Tutorials/quick\\_start.html](https://sumo.dlr.de/docs/Tutorials/quick_start.html) (besucht am 10.02.2021).
- [54] *Road2Simulation*. URL: [https://www.dlr.de/ts/en/desktopdefault.aspx/tabcid-11648/20367\\_read-46771/](https://www.dlr.de/ts/en/desktopdefault.aspx/tabcid-11648/20367_read-46771/) (besucht am 10.02.2021).
- [55] *Gitlab: The complete DevOps platform*. URL: <https://about.gitlab.com/> (besucht am 10.02.2021).
- [56] *Gitlab CI/CD*. URL: <https://docs.gitlab.com/ee/ci/README.html> (besucht am 10.02.2021).
- [57] *docker official website*. URL: <https://www.docker.com/> (besucht am 10.02.2021).
- [58] Docker. *Docker docs*. URL: <https://docs.docker.com/> (besucht am 19.02.2021).
- [59] Docker. *Docker Hub*. URL: <https://hub.docker.com/> (besucht am 19.02.2021).
- [60] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR). „Eclipse Automated Driving Open Research (ADORE)“. In: (2017). URL: [https://github.com/eclipse/adore/tree/master/adore\\_if\\_ros\\_demos](https://github.com/eclipse/adore/tree/master/adore_if_ros_demos) (besucht am 11.02.2021).
- [61] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR). „ADORE Installation Documentation“. In: (2017). URL: <https://github.com/eclipse/adore/blob/master/documentation/installation.md> (besucht am 12.02.2021).
- [62] Steven Waslander Jonathan Kelly. *Motion Planning for Self-Driving Cars Lesson 2: Collision Checking*. Techn. Ber. University of Toronto. URL: <https://www.coursera.org/lecture/motion-planning-self-driving-cars/lesson-2-collision-checking-mxs6N> (besucht am 28.09.2020).
- [63] Jeremy Cohen. „How Google’s Self-Driving Cars Work“. In: (2020). URL: <https://heartbeat.fritz.ai/how-googles-self-driving-cars-work-c77e4126f6e7> (besucht am 10.10.2020).
- [64] S. Klapp. „Empirical Parameter Variation Analysis for Electronic Circuits“. In: *IEEE Transactions on Reliability* R-13.1 (1964), S. 34–40. DOI: 10.1109/TR.1964.5218242. (Besucht am 19.02.2021).
- [65] Anton Lawrence. *How to Improve Your Workflow Using Model-Based Testing*. URL: <https://www.freecodecamp.org/news/improve-your-workflow-using-model-based-testing/> (besucht am 19.02.2021).
- [66] *Ros quaternions*. URL: <http://wiki.ros.org/tf2/Tutorials/Quaternions> (besucht am 16.02.2021).
- [67] *Bremsweg und Sicherheitsabstand*. URL: [https://www.vbg.de/zeitarbeit-fb/daten/apl/arbhilf/unterw/176\\_bus.htm#:~:text=Also](https://www.vbg.de/zeitarbeit-fb/daten/apl/arbhilf/unterw/176_bus.htm#:~:text=Also) (besucht am 10.02.2021).

- [68] Gitlab. *CI/CD pipelines*. URL: <https://docs.gitlab.com/ee/ci/pipelines/> (besucht am 16.02.2021).
- [69] Gitlab. *gitlab runner*. URL: <https://docs.gitlab.com/runner/> (besucht am 16.02.2021).
- [70] Gitlab. *gitlab ci yaml*. URL: [https://docs.gitlab.com/ee/ci/yaml/gitlab\\_ci\\_yaml.html](https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html) (besucht am 19.02.2021).
- [71] DockerHub: *AdoreCi Image*. URL: <https://hub.docker.com/r/gitlabciaadore/adoreci> (besucht am 16.02.2021).
- [72] *docker get started*. URL: <https://docs.docker.com/get-started/> (besucht am 19.02.2021).
- [73] *Docker Run Docs*. URL: <https://docs.docker.com/engine/reference/run/> (besucht am 17.02.2021).
- [74] *Docker Volume Docs*. URL: <https://docs.docker.com/storage/volumes/> (besucht am 18.02.2021).
- [75] *pyModelChecking: Model checking*. URL: [https://pymodelchecking.readthedocs.io/en/latest/model\\_checking.html](https://pymodelchecking.readthedocs.io/en/latest/model_checking.html) (besucht am 21.02.2021).
- [76] *Model Checking*. URL: [https://dewiki.de/Lexikon/Model\\_Checking](https://dewiki.de/Lexikon/Model_Checking) (besucht am 16.02.2021).
- [77] *Model Checking*. URL: [https://de.wikipedia.org/wiki/Model\\_Checking](https://de.wikipedia.org/wiki/Model_Checking) (besucht am 16.02.2021).
- [78] *Reactive Systems: KripkeStructures*. URL: <https://pymodelchecking.readthedocs.io/en/latest/models.html#kripke-structures> (besucht am 21.02.2021).
- [79] *Kripke structure (model checking)*. URL: [https://en.wikipedia.org/wiki/Kripke\\_structure\\_\(model\\_checking\)](https://en.wikipedia.org/wiki/Kripke_structure_(model_checking)) (besucht am 21.02.2021).
- [80] F. Kröger. „Temporale Logik und Zustandssysteme“. In: (2004). URL: <https://www.pst.ifi.lmu.de/Lehre/WS0405/tl/skriptum/skriptum.pdf> (besucht am 16.02.2021).
- [81] *pyModelChecking: Logics*. URL: <https://pymodelchecking.readthedocs.io/en/latest/logics.html> (besucht am 21.02.2021).
- [82] *Computational Tree Logic*. URL: [https://de.wikipedia.org/wiki/Computation\\_Tree\\_Logic](https://de.wikipedia.org/wiki/Computation_Tree_Logic) (besucht am 19.02.2021).
- [83] *pyModelChecking*. URL: <https://pypi.org/project/pyModelChecking/> (besucht am 21.02.2021).
- [84] Matthias Althoff, Markus Koschi und Stefanie Manzinger. „Common-Road: Composable benchmarks for motion planning on roads“. In: *Proc. of the IEEE Intelligent Vehicles Symposium*. 2017, S. 719–726. ISBN: 9781509048045. DOI: 10.1109/ivs.2017.7995802. URL: <https://mediatum.ub.tum.de/doc/1379638/1379638.pdf> (besucht am 01.10.2020).

- [85] *CommonRoad TUM*. URL: <https://commonroad.in.tum.de/> (besucht am 18.02.2021).
- [86] *Carla Simulator*. URL: <https://carla.org/> (besucht am 12.03.2021).
- [87] *Setting up CARLA Simulator for the Self-Driving Cars Specialization*. URL: <https://medium.datadriveninvestor.com/setting-up-carla-simulator-for-the-self-driving-cars-specialization-d38d4f6a0486> (besucht am 12.03.2021).
- [88] Diego Galar und Uday Kumar. „Chapter 1 - Sensors and Data Acquisition“. In: *eMaintenance*. Hrsg. von Diego Galar und Uday Kumar. Academic Press, 2017, S. 1–72. ISBN: 978-0-12-811153-6. DOI: <https://doi.org/10.1016/B978-0-12-811153-6.00001-4>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128111536000014> (besucht am 07.01.2021).

# Glossar

**ADORE** Eclipse Automated Driving Open Research. 2, 3

**ADS** Automated Driving System. 21

**AF** Automatisierte Fahrzeuge, Autonome Fahrzeuge, selbstfahrende Autos.  
1–3, 6, 9–11, 17, 19, 72

**ASIC** application-specific integrated circuit. 12

**DL** Deep Learning. 12

**DLR** deutsches Zentrum für Luft und Raumfahrt. 3

**Ego** Ego-Fahrzeug: Ziel Fahrzeug, das autonom gesteuert wird. 8, 9, 11, 13

**FO** Fail-operational. 23

**FOT** Field Operational Test. 2

**FPGA** Field Programmable Gate Array. 12

**FS** Fail-Safe. 23

**FuSa** Functional Safety. 20

**GNSS** Global Navigation Satellite System. 11

**GPS** Global Positioning System. 11

**GPU** graphics processing unit. 12

**HARA** Hazard Analysis and Risk Assessment. 20

**HAZOP** Hazard and operability study. 20

**IMU** Inertial Measurement Unit. 11

**ISO** International Organization for Standardization. 20

- Lidar** Light Detection and Ranging. 10
- merge** in den Verkehr einfließen bzw. einordnen. 24
- ML** Machine Learning. 12
- MRC** minimal risk condition. 23
- NHTSA** National Highway Transportation Safety Administration. 21
- ODD** Operational Design Domain. 8, 9, 23
- OEDR** Object and Event Detection and Response. 9, 23
- PAS** Publicly Available Specification. 20
- PEGASUS** Project for the Establishment of Generally Accepted quality criteria, tools and methods as well as Scenarios and Situations. 3
- Radar** Radio Detection And Ranging. 10
- SAE** Society of Automotive Engineers. 7
- Sensorfusion** Daten werden von mehreren Sensoren zusammengeführt, um die Unsicherheiten zu reduzieren, die bei der Ausführung von Aufgaben eines autonomen Systems auftreten können[88]. 11
- Sonar** Sound Navigation And Ranging. 10
- SOTIF** Safety of Intended Functionality. 20
- V2X** Vehicle to everything. 14, 16

# **Anhang**

# NHTSA

Set of Behavioral Competencies Recommended by NHTSA	
<b>1</b>	Detect and Respond to Speed Limit Changes and Speed Advisories
<b>2</b>	Perform High-Speed Merge (e.g., Freeway)
<b>3</b>	Perform Low-Speed Merge
<b>4</b>	Move Out of the Travel Lane and Park (e.g., to the Shoulder for Minimal Risk)
<b>5</b>	Detect and Respond to Encroaching Oncoming Vehicles
<b>6</b>	Detect Passing and No Passing Zones and Perform Passing Maneuvers
<b>7</b>	Perform Car Following (Including Stop and Go)
<b>8</b>	Detect and Respond to Stopped Vehicles
<b>9</b>	Detect and Respond to Lane Changes
<b>10</b>	Detect and Respond to Static Obstacles in the Path of the Vehicle
<b>11</b>	Detect Traffic Signals and Stop/Yield Signs
<b>12</b>	Respond to Traffic Signals and Stop/Yield Signs
<b>13</b>	Navigate Intersections and Perform Turns
<b>14</b>	Navigate Roundabouts
<b>15</b>	Navigate a Parking Lot and Locate Spaces
<b>16</b>	Detect and Respond to Access Restrictions (One-Way, No Turn, Ramps, etc.)
<b>17</b>	Detect and Respond to Work Zones and People Directing Traffic in Unplanned or Planned Events
<b>18</b>	Make Appropriate Right-of-Way Decisions
<b>19</b>	Follow Local and State Driving Laws
<b>20</b>	Follow Police/First Responder Controlling Traffic (Overriding or Acting as Traffic Control Device)
<b>21</b>	Follow Construction Zone Workers Controlling Traffic Patterns (Slow/Stop Sign Holders)
<b>22</b>	Respond to Citizens Directing Traffic After a Crash
<b>23</b>	Detect and Respond to Temporary Traffic Control Devices
<b>24</b>	Detect and Respond to Emergency Vehicles
<b>25</b>	Yield for Law Enforcement, EMT, Fire, and Other Emergency Vehicles at Intersections, Junctions, and Other Traffic Controlled Situations
<b>26</b>	Yield to Pedestrians and Bicyclists at Intersections and Crosswalks
<b>27</b>	Provide Safe Distance From Vehicles, Pedestrians, Bicyclists on Side of the Road
<b>28</b>	Detect/Respond to Detours and/or Other Temporary Changes in Traffic Patterns

Tabelle 7.1: NHTSA empfohlene Verhaltenskompetenzen [5]

<b>Examples of Additional Behavioral Competencies Tested by Waymo</b>	
<b>29</b>	Moving to a Minimum Risk Condition When Exiting the Travel Lane is Not Possible
<b>30</b>	Perform Lane Changes
<b>31</b>	Detect and Respond to Lead Vehicle
<b>32</b>	Detect and Respond to a Merging Vehicle
<b>33</b>	Detect and Respond to Pedestrians in Road [Not Walking Through Intersection or Crosswalk]
<b>34</b>	Provide Safe Distance from Bicyclists Traveling on Road (With or Without Bike Lane)
<b>35</b>	Detect and Respond to Animals
<b>36</b>	Detect and Respond to Motorcyclists
<b>37</b>	Detect and Respond to School Buses
<b>38</b>	Navigate Around Unexpected Road Closures (e.g. Lane, Intersection, etc.)
<b>39</b>	Navigate Railroad Crossings
<b>40</b>	Make Appropriate Reversing Maneuvers
<b>41</b>	Detect and Respond to Vehicle Control Loss (e.g. reduced road friction)
<b>42</b>	Detect and Respond to Conditions Involving Vehicle, System, or Component-Level Failures or Faults (e.g. power failure, sensing failure, sensing obstruction, computing failure, fault handling or response)
<b>43</b>	Detect and Respond to Unanticipated Weather or Lighting Conditions Outside of Vehicle's Capability (e.g. rainstorm)
<b>44</b>	Detect and Respond to Unanticipated Lighting Conditions (e.g. power outages)
<b>45</b>	Detect and Respond to Non-Collision Safety Situations (e.g. vehicle doors ajar)
<b>46</b>	Detect and Respond to Faded or Missing Roadway Markings or Signage
<b>47</b>	Detect and Respond to Vehicles Parking in the Roadway

Tabelle 7.2: Waymo weiterentwickelte Verhaltenskompetenzen [5]

Crash Avoidance Category	Example Test Scenario
<b>Rear-end</b> Demonstrate ability to avoid or mitigate crashes with lead vehicles.	Fully self-driving vehicle approaches stopped lead vehicle
	Fully self-driving vehicle approaches disabled vehicle
	Fully self-driving vehicle approaches lead vehicle traveling at lower constant speed
	Fully self-driving vehicle approaches lead vehicle traveling at slower speed and initiating strong braking
	Fully self-driving vehicle approaches lead vehicle accelerating
	Fully self-driving vehicle following a lead vehicle making a maneuver (e.g. cutting into lane or pulling out of driveway)
	Fully self-driving vehicle approaches lead vehicle decelerating
	Fully self-driving vehicle approaches other vehicle(s) reversing
	Fully self-driving vehicle approaches other vehicle(s) parking
	Fully self-driving vehicle approaches protected intersection, Vehicle A approaches from right
<b>Intersection</b> Demonstrate ability to detect vehicle entering path at perpendicular angle and apply brakes.	Fully self-driving vehicle approaches protected intersection, Vehicle A approaches from left
	Fully self-driving vehicle prepares to turn across unprotected intersection, oncoming Vehicle A approaches
	Crossing path collisions - other vehicle running red light
	Crossing path collisions - other vehicle running stop sign

Crash Avoidance Category (continued)	Example Test Scenario (continued)
<b>Road Departure</b> Demonstrate ability to steer clear of roadway edge and stay within lane.	Fully self-driving vehicle travels down straight road (with or without prior vehicle maneuver)
	Fully self-driving vehicle travels down curved road (with or without prior vehicle maneuver)
	Fully self-driving vehicle travels down straight road with visible lane marking
	Fully self-driving vehicle travels down straight road with faded or missing lane marking
	Fully self-driving vehicle travels down curved road with visible lane marking
	Fully self-driving vehicle travels down curved road with faded or missing lane marking
	Fully self-driving vehicle travels down wet road with lane marking
	Fully self-driving vehicle approaches other vehicle(s) reversing
	Fully self-driving vehicle travels down wet road with faded or missing lane marking
	Lane changes - other vehicles turning same direction
<b>Lane Change</b> Demonstrate ability to avoid or mitigate crash when other vehicles make lane changes or merge.	Lane changes - other vehicles parking same direction
	Lane changes - other vehicles changing lanes same direction
	Lane changes - other vehicles drifting same direction
	Lane merges

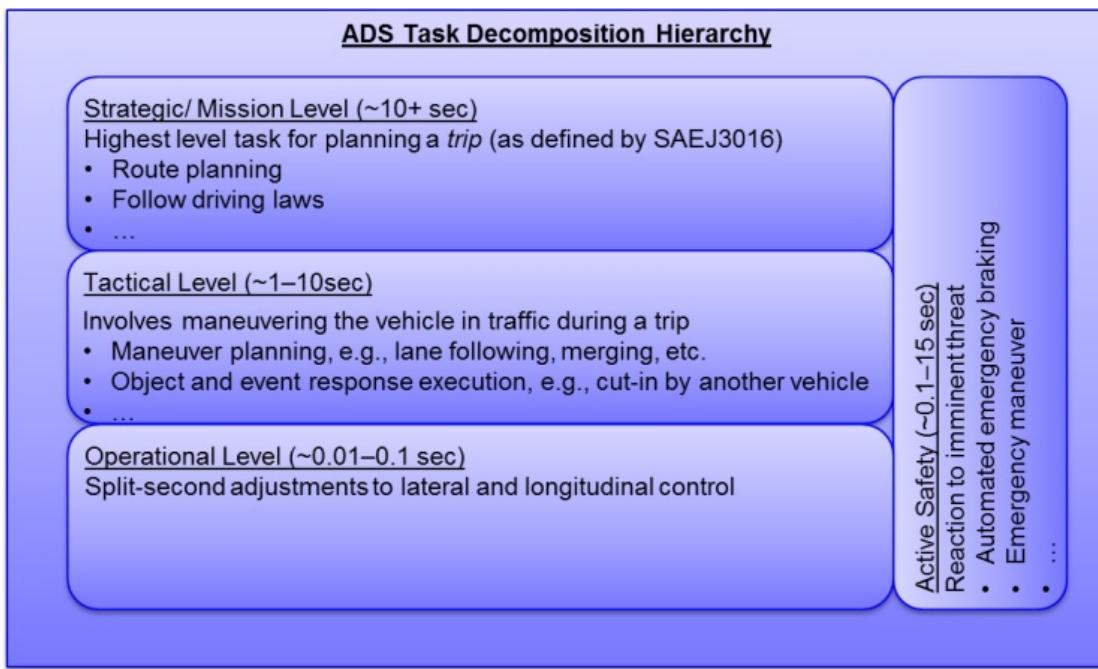


Abbildung 7.2: Aufgabe Verteilung innerhalb einer Hierarchie basierend auf der Dauer des Verhaltens [43]

ADS Features and Tactical and Operational Maneuvers  (X = demonstrated, ? = speculated)	Commercially Available? (Y/N)	Level of Automation (SAE 1-5)															
		Parking	Maintain Speed	Car Following	Lane Centering	Lane Switching/Overtaking	Enhancing Conspicuity	Merge	Navigate On/Off Ramps	Follow Driving Laws	Navigate Roundabouts	Navigate Intersection	Navigate Crosswalk	Navigate Work Zone	N-Point Turn	U-Turn	Route Planning
Waymo Automated TNC	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Tesla Self-Drive	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Volkswagen I.D. Pilot	N	4?	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Volvo IntelliSafe Auto Pilot	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Nissan Autonomous Drive (2020)	N	4?	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
GM Cruise Automation	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Uber Automated TNC	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Honda Automated Drive (2020)	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Ford Automated TNC (2022)	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Baidu Automated TNC	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Toyota Chauffeur	N	4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabelle 7.4: L4 Hochautomatisierte Fahrzeugmanöver [43]

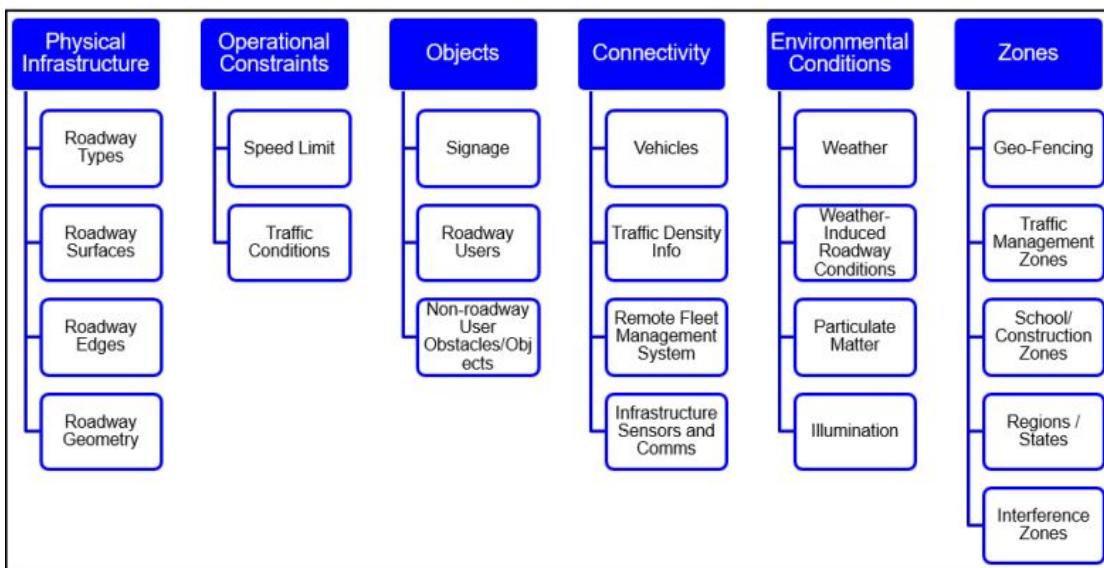


Abbildung 7.3: ODD Klassifikation [43]

Detect & Respond to Speed Limit Changes	Detect & Respond to Relevant School Buses
Detect & Respond to Encroaching, Oncoming Vehicles	Detect & Respond to Relevant Emergency Vehicles
Perform Vehicle Following	Detect & Respond to Relevant Pedestrians
Detect & Respond to Relevant Stopped Vehicles	Detect & Respond to Relevant Pedalcyclists
Detect & Respond to Relevant Lane Changes/Cut-ins	Detect & Respond to Relevant Animals
Detect & Respond to Relevant Static Obstacles in Lane	Detect & Respond to Relevant Vehicle Cut-out/Reveal
Detect & Navigate Work Zones	Detect & Respond to Relevant Vehicle Roadway Entry
Detect & Respond to Relevant Safety Officials	Detect & Respond to Relevant Adjacent Vehicles
Detect & Respond to Relevant Access Restrictions	Detect & Respond to ODD Boundary Transition
Detect & Respond to Relevant Dynamic Traffic Signs	

Tabelle 7.5: Liste 19 NHTSA OEDR [43]

## 1. PERFORM LANE CHANGE/LOW-SPEED MERGE (see [43] for other scenarios)

### ODD Characteristics

- Multi-lane divided highway (or similar)
- Asphalt or concrete

- Straight, flat
- Clear lane markers
- Clear sky, dry, daylight

### **OEDR Characteristics**

- Optional object vehicles

### **Failure Behaviors**

- None

### **Test Protocol**

### **Vehicle Platforms**

Subject Vehicle(SV)– The vehicle equipped with the ADS feature being tested. Principal Other Vehicles(POV)– The primary object vehicles for which the detection and response of the subject vehicle are being tested.

### **Vehicle Roles**

The SV is a light-duty vehicle equipped with an ADS feature that is being evaluated. The POVs are other fully functional (operational brake lights, etc.) light-duty vehicles (e.g., sedan, SUVs, pickup trucks, etc.) or vehicle surrogates. If a vehicle surrogate is used, it would ideally be frangible and should possess similar mobility and detection characteristics as a regular light-duty vehicle.

- Ability to be towed or remotely controlled to follow the test course
- Ability to achieve test speeds
- Similar visual appearance
- Similar radar and/or lidar reflectivity

Maneuver	SV Speed kph (mph)	POV <sup>15</sup> Speed kph (mph)	Location of POV_1	Location of POV_2	Location of POV_3
Baseline 15 PLC_B_15	24 (15)	N/A	N/A	N/A	N/A
Baseline 25 PLC_B_25	40 (25)	N/A	N/A	N/A	N/A
Baseline 35 PLC_B_35	56 (35)	N/A	N/A	N/A	N/A
Simple Positive 15 PLC_SP_15	24 (15)	24 (15)	Rear bumper 6 m (20 ft) in front of SV front bumper	N/A	N/A
Simple Positive 25 PLC_SP_25	40 (25)	40 (25)	Rear bumper 6 m (20 ft) in front of SV front bumper	N/A	N/A
Simple Positive 35 PLC_SP_35	56 (35)	56 (35)	Rear bumper 6 m (20 ft) in front of SV front bumper	N/A	N/A
Complex Positive 15 PLC_CP_15	24 (15)	24 (15)	Rear bumper 8 m (25 ft) in front of SV front bumper	Front bumper 25 ft (8 m) behind SV rear bumper	N/A
Complex Positive 25 PLC_CP_25	40 (25)	40 (25)	Rear bumper 8 m (25 ft) in front of SV front bumper	Front bumper 25 ft (8 m) behind SV rear bumper	N/A
Complex Positive 35 PLC_CP_35	56 (35)	56 (35)	Rear bumper 8 m (25 ft) in front of SV front bumper	Front bumper 25 ft (8 m) behind SV rear bumper	N/A
Simple Negative 15 PLC_SN_15	24 (15)	24 (15)	Rear bumper ≤ 5 m (15 ft) in front of SV front bumper	Front bumper even with SV front bumper	Front bumper ≤ 15 ft (5 m) behind SV rear bumper
Simple Negative 25 PLC_SN_25	40 (25)	40 (25)	Rear bumper ≤ 6 m (20 ft) in front of SV front bumper	Front bumper even with SV front bumper	Front bumper ≤ 20 ft (6 m) behind SV rear bumper
Simple Negative 35 PLC_SN_35	56 (35)	56 (35)	Rear bumper ≤ 8 m (25 ft) in front of SV front bumper	Front bumper even with SV front bumper	Front bumper ≤ 25 ft (8 m) behind SV rear bumper

Tabelle 7.6: Perform Lane Change Test Scenarios [43]

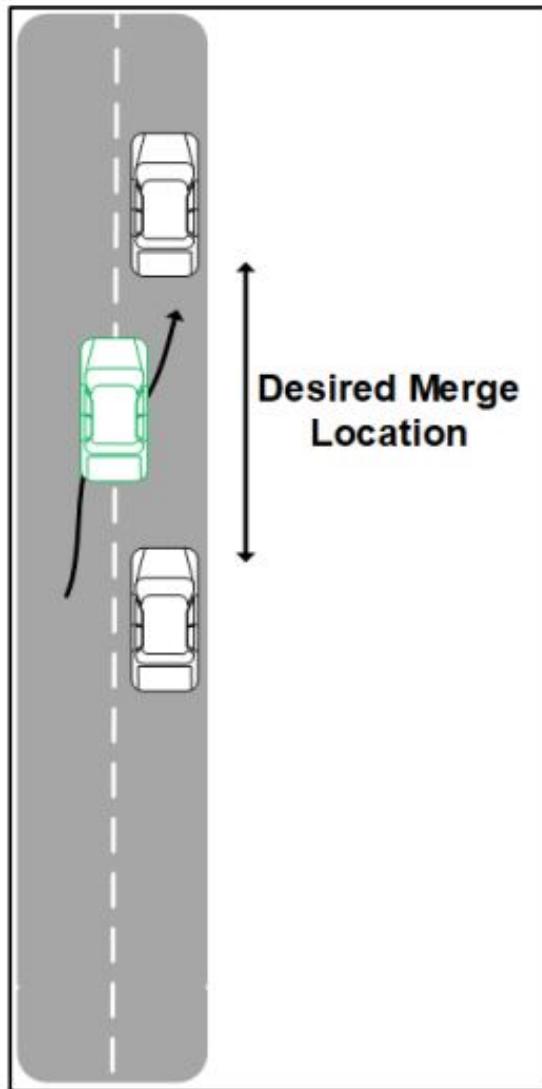


Abbildung 7.4: Einordnen (Merge) Testszenario [43]

## General Procedures

### Ambient Conditions

- The ambient temperature shall be between 0 °C (32 °F) and 38 °C (100 °F).
- The maximum wind speed shall be no greater than 10 m/s (22 mph).
- Tests should not be performed during periods of inclement weather. This includes, but is not limited to, rain, snow, hail, fog, smoke, or ash.
- Unless specified otherwise, the tests shall be conducted during daylight hours with good atmospheric visibility (defined as an absence of fog and the

ability to see clearly for more than 5,000 m). The test shall not be conducted with the vehicle oriented into the sun during very low sun angle conditions (the sun is oriented 15 degrees or less from horizontal), where low sun angles degrade forward visibility for the test vehicle operators.

- Unless stated otherwise, all tests shall be conducted such that there are no overhead signs, bridges, or other significant structures over, or near, the testing site. Each trial shall be conducted with no vehicles, obstructions, or stationary objects within one lane width of either side the vehicle path.

## **Personnel**

A test execution team would include an SV safety driver, an experimenter, and one or more POV operators, and potentially external observers. The team would typically coordinate using person-to-person radios for communication. The SV safety driver would be skilled in the operation of the ADS feature under test. This skill and knowledge would include familiarity with the ADS feature user interface, activation and deactivation procedures, and potential failure modes. The safety driver must be capable of disengaging the ADS feature under test and bringing the vehicle to a minimal risk state, if the experiment approaches or reaches an unsafe state.

The experimenter observes and directs execution of each test trial and would typically be in the SV as the test is executed. The experimenter would also be knowledgeable of the operation of the ADS feature under test to determine if it is functioning properly. The experimenter records test conditions and test trial notes, and judges apparent test trial validity. The experimenter might also operate the data acquisition system and other test equipment.

The POV operator would hold a valid driver's license and be comfortable operating the POVs. The POV operator would be responsible for positioning the POVs for each trial. If the POV is a vehicle surrogate, the POV operator would be knowledgeable of its construction and mobility and be able to position and control the surrogate for the prescribed trials. The other observers may be responsible for operating external data collection equipment (e.g., video recording of test execution, etc.). Test Data and Equipment Relevant data listed below should be collected to support the metrics identified for each test scenario/trial. Options for equipment to collect the individual data elements are also provided.

- Vehicle Positions (SV and POVs): GPS/inertial navigation system (< X cm RMS, 95% confidence interval)
- Vehicle Speeds (SV and POVs): GPS/INS, estimated from position information
- Ranges (closest points between SV and POV): lidar, radar, estimated from position information
- Turn signal status
- Ambient Conditions:

- o Temperature: thermometer ( $^{\circ}\text{C}$ ,  $^{\circ}\text{F}$ )
- o Wind Speed: anemometer (mph, kph)
- o Precipitation: range gauge (in/h, cm/h)
- o Time: clock
- o Sun position: manual observation
- Test Documentation: camera
- Experimenter Notes

### **Test Facility**

For performing lane change competency tests, the test facility is a straight, flat, and level roadway that includes one driving lane, whose surface is constructed of asphalt or concrete, and whose driving lane is at least 12 ft wide and delineated by lane markings visible to the vehicle operators. The only exceptions to this may be for tests where the roadway is curved instead of straight. The length of the roadway will be sufficient to allow the ADS feature under test to establish and maintain a specified lane and speed, and to allow the SV to stop or exit the course, if applicable. The length of the test course is at least greater than the maximum SV perception range, or 105 m, whichever is greater.

## **Scenario Test: PLC-Comp-15, Straight Road, Complex, 15 mph**

### **Scenario Description**

A vehicle equipped with an ADS feature is driving along a straight urban street with multiple lanes. It is approaching a necessary turn and needs to change lanes to position itself in the appropriate lane to make the turn.

### **Test Subject and Purpose**

The subject of this test is an ADS feature whose specified ODD includes operation on improved urban roads with other traffic vehicles. The test determines the ability of the ADS feature to change lanes in the presence of other traffic vehicles.

### **Initial Conditions**

The SV will initially be static in the prescribed positions and orientations. The POVs will initially be static in the prescribed positions ahead of the SV in an adjacent lane. The leading edge of POV-2 will be approximately 3 m behind the trailing edge of POV1.

### **Test Velocities**

The steady state velocities of the SV and POV are specified for each trial or set of trials.

### **Metrics**

### **Disengagements**

A disengagement is defined as the SV safety driver deactivating the ADS feature being evaluated and taking manual control of the SV. The location

and manner of the disengagement should be included in the experimenter's notes.

### **Separation Distances**

The separation distances are the distances between the SV and each of the POVs. The minimum separation distances (closest approach) should be identified, as well as the separation distances being observed as a continuum.

### **Signal Status**

Signal status is the activation state of the SV turn signal, to be measured at a periodic rate to determine when the signal is activated and deactivated.

### **Execution of Procedure**

1. The POVs are positioned in the center of the right lane of the test road at their specified locations.
2. The SV is positioned in the center of a left lane of the test road immediately adjacent to POV\_2.
3. The SV is given a target destination in the right lane at the end of the test course.
4. The SV's navigation system is activated to begin traversing the course.
5. As the SV begins moving, the POVs simultaneously begin accelerating to the specified steady state velocity while maintaining the approximate separation distance.
6. Each trial ends when the SV successfully changes lanes to merge between POV\_1 and POV\_2 and stops at the target destination, or the SV driver must intervene.
7. After the end of the trial, the SV driver disengages the ADS feature (if it is not already disengaged).

### **Trial Validity**

An individual trial is valid if during the trial:

1. The velocity of the POVs did not exceed  $\pm X$  kph from the specified steady state velocities.
2. The separation distance between the POVs did not exceed  $\pm X$  m from the specified separation distance.
3. The POVs did not deviate from the specified lane. NOTE: Other trial validity requirements might include GPS coverage requirements.

### **Evaluation Metrics**

A trial is successful if the SV:

- Successfully accelerates and merges between the two POVs with a minimum separation distance of  $\geq X$  m with each POV.
- Successfully decelerates and merges behind POV\_2 with a minimum separation distance of  $\geq X$  m with POV\_2.
- Successfully accelerates and merges ahead of POV\_1 with a minimum separation distance of  $\geq X$  m with POV\_1 and does not exceed Y kph of the specified speed limit.

Categories of Behavioral Competencies	NHTSA Testable Cases	Waymo Voluntary Safety Self-Assessment	California PATH Behavior Competencies	NHTSA Pre-Crash Scenarios
<b>Tactical Maneuvers</b>				
Parking	• Parking	<ul style="list-style-type: none"> <li>• Navigate a Parking Lot and Locate Spaces</li> <li>• Make Appropriate Reversing Maneuvers</li> </ul>	<ul style="list-style-type: none"> <li>• Navigate a Parking Lot and Locate Open Spaces</li> </ul>	<ul style="list-style-type: none"> <li>• Vehicles Parking</li> </ul>
Lane Maintenance & Car Following	<ul style="list-style-type: none"> <li>• Car Following</li> <li>• Speed Maintenance</li> <li>• Lane Centering</li> <li>• Enhancing Conspicuity (headlights)</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Speed Limit Changes and Speed Advisories</li> <li>• Detect and Respond to Encroaching Oncoming Vehicles</li> <li>• Perform Car Following (Including Stop and Go)</li> </ul>	<ul style="list-style-type: none"> <li>• Perform Car Following Including Stop &amp; Go and Emergency Braking</li> <li>• Detect &amp; Respond to Speed Limit Changes (Including Advisory Speed Zones)</li> </ul>	<ul style="list-style-type: none"> <li>• Lead Vehicle Stopped</li> <li>• Vehicles Turning at Non-Signalized Junctions</li> <li>• Lead Vehicle Decelerating</li> <li>• Vehicles Changing Lanes</li> <li>• Straight Crossing paths at Non-Signalized Junctions</li> </ul>
Lane Change (e.g., overtake, merge)	<ul style="list-style-type: none"> <li>• Lane Switching/ Overtaking</li> <li>• Enhancing Conspicuity (e.g., blinkers)</li> <li>• Merge (high &amp; low speed)</li> </ul>	<ul style="list-style-type: none"> <li>• Perform High-Speed Merge (e.g., Freeway)</li> <li>• Perform Low-Speed Merge</li> <li>• Move Out of the Travel Lane and Park (e.g., to the Shoulder for Minimal Risk)</li> <li>• Detect and Respond to Encroaching Oncoming Vehicles</li> <li>• Detect Passing and No Passing Zones and Perform Passing Maneuvers</li> <li>• Perform Lane Changes</li> </ul>	<ul style="list-style-type: none"> <li>• Detect Passing and No Passing Zones</li> <li>• Perform High Speed Freeway Merge</li> <li>• Perform a Lane Change or Lower Speed Merge</li> <li>• Park on the Shoulder or Transition the Vehicle to a Minimal Risk State (Not Required for SAE L3)</li> </ul>	<ul style="list-style-type: none"> <li>• Vehicles Turning at Non-Signalized Junctions</li> <li>• Vehicles Changing Lanes</li> <li>• Straight Crossing paths at Non-Signalized Junctions</li> </ul>
Navigate Intersection: <ul style="list-style-type: none"> <li>• Type: Signalized, Non-signalized, Roundabout, Rail Crossing</li> <li>• Turn: Left/ Right/ Straight</li> </ul>	<ul style="list-style-type: none"> <li>• Navigate On/Off Ramps</li> <li>• Roundabouts</li> <li>• Intersection (left, right, straight)</li> <li>• Crosswalk</li> <li>• U-Turn</li> </ul>	<ul style="list-style-type: none"> <li>• Perform Car Following (Including Stop and Go)</li> <li>• Navigate Intersections and Perform Turns</li> <li>• Navigate Roundabouts</li> <li>• Navigate Railroad Crossings</li> </ul>	<ul style="list-style-type: none"> <li>• Navigate Intersections &amp; Perform Turns</li> <li>• Detect and Respond to Traffic Control Devices</li> <li>• Navigate Intersections &amp; Perform Turns</li> </ul>	<ul style="list-style-type: none"> <li>• Running Red Light</li> <li>• Vehicles Turning - Same Direction</li> <li>• LTAP/OD at Signalized Junction</li> <li>• LTAP/OD at Non-Signalized Junction</li> <li>• Running Stop Sign</li> <li>• Vehicle Turning Right at Signalized Intersection</li> </ul>
Navigate Temporary or A-Typical Condition	<ul style="list-style-type: none"> <li>• Detect and Respond to Workzone</li> <li>• N-Point Turn</li> <li>• Detect and Respond to Relevant Safety Officials</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Work Zones and People Directing Traffic in Unplanned or Planned Events</li> <li>• Follow Police/First Responder Controlling Traffic (Overriding or Acting as Traffic Control Device)</li> <li>• Follow Construction Zone Workers Controlling Traffic Patterns (Slow/Stop Sign Holders)</li> <li>• Respond to Citizens Directing Traffic After a Crash</li> <li>• Detect/Respond to Detours and/or Other</li> </ul>	<ul style="list-style-type: none"> <li>• Detect Work Zones, Temporary Lane Shifts, or Safety Officials Manually Directing Traffic</li> </ul>	98

Abbildung 7.5: Vergleich taktische Manöver [43]

		<p>Temporary Changes in Traffic Patterns</p> <ul style="list-style-type: none"> <li>• Navigate Around Unexpected Road Closures (e.g., Lane, Intersection, etc.)</li> </ul>		
<b>OEDR Capabilities</b>				
OEDR: Vehicles	<ul style="list-style-type: none"> <li>• Detect and Respond to Encroaching, Oncoming Vehicles</li> <li>• Vehicle Following</li> <li>• Detect and Respond to Relevant Stopped Vehicle</li> <li>• Detect and Respond to Lane Changes/ Cut-ins</li> <li>• Detect and Respond to Cut-outs/ Reveals</li> <li>• Detect and Respond to School Buses</li> <li>• Detect and Respond to Emergency Vehicles</li> <li>• Detect and Respond to Vehicle Roadway Entry</li> <li>• Detect and Respond to Relevant Adjacent Vehicles</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Encroaching Oncoming Vehicles</li> <li>• Detect and Respond to Stopped Vehicles</li> <li>• Detect and Respond to Lane Changes</li> <li>• Detect and Respond to Emergency Vehicles</li> <li>• Yield for Law Enforcement, EMT, Fire, and Other Emergency Vehicles at Intersections, Junctions, and Other Traffic Controlled Situations</li> <li>• Provide Safe Distance From Vehicles, Pedestrians, Bicyclists on Side of the Road</li> <li>• Detect and Respond to Lead Vehicle</li> <li>• Detect and Respond to a Merging Vehicle</li> <li>• Detect and Respond to Motorcyclists</li> <li>• Detect and Respond to School Buses</li> <li>• Detect and Respond to Vehicles Parking in the Roadway</li> </ul>	<ul style="list-style-type: none"> <li>• Detect Emergency Vehicles</li> <li>• Detect &amp; Respond to Stopped Vehicles</li> <li>• Detect &amp; Respond to Intended Lane Changes/Cut-Ins</li> <li>• Detect &amp; Respond to Encroaching Oncoming Vehicles</li> </ul>	<ul style="list-style-type: none"> <li>• Running Red Light</li> <li>• Lead Vehicle Moving at Lower Constant Speed</li> <li>• Backing Up Into Another Vehicle</li> <li>• Vehicless Not Making A Maneuver - Opposite Direction</li> <li>• Vehicles Drifting - Same Direction</li> <li>• Following Vehicle Making Maneuver</li> <li>• Running Stop Sign</li> <li>• Lead Vehicle Accelerating</li> <li>• Vehicles Making a Maneuver - Opposite Direction</li> </ul>

Abbildung 7.6: Vergleich OEDR1 [43]

OEDR: Traffic Control Devices & Infrastructure	<ul style="list-style-type: none"> <li>• Follow Driving Laws</li> <li>• Detect and Respond to Speed Limit Changes</li> <li>• Detect and Respond to Relevant Access Restrictions</li> <li>• Detect and Respond to Relevant Dynamic Traffic Signs</li> </ul>	<ul style="list-style-type: none"> <li>• Detect Traffic Signals and Stop/Yield Signs</li> <li>• Respond to Traffic Signals and Stop/Yield Signs</li> <li>• Detect and Respond to Access Restrictions (One-Way, No Turn, Ramps, etc.)</li> <li>• Make Appropriate Right-of-Way Decisions</li> <li>• Follow Local and State Driving Laws</li> <li>• Detect and Respond to Temporary Traffic Control Devices</li> <li>• Detect/Respond to Detours and/or Other Temporary Changes in Traffic Patterns</li> <li>• Detect and Respond to Faded or Missing Roadway Markings or Signage</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Access Restrictions such as One-Way Streets, No-Turn Locations, Bicycle Lanes, Transit Lanes, and Pedestrian Ways</li> <li>• Detect and Respond to Traffic Control Devices</li> </ul>	
OEDR: Vulnerable Road Users (VRU), Objects, Animals	<ul style="list-style-type: none"> <li>• Detect and Respond to Relevant Static Obstacles in Lane</li> <li>• Detect and Respond to Pedestrians, Pedalcyclists, Animals</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Static Obstacles in the Path of the Vehicle</li> <li>• Yield to Pedestrians and Bicyclists at Intersections and Crosswalks</li> <li>• Provide Safe Distance From Vehicles, Pedestrians, Bicyclists on Side of the Road</li> <li>• Detect and Respond to Pedestrians in Road (Not Walking Through Intersection or Crosswalk)</li> <li>• Provide Safe Distance from Bicyclists Traveling on Road (With or Without Bike Lane)</li> <li>• Detect and Respond to Animals</li> </ul>	<ul style="list-style-type: none"> <li>• Detect &amp; Respond to Static Obstacles in Roadway</li> <li>• Detect &amp; Respond to Bicycles, Pedestrians, Animals, or Other Moving Objects</li> </ul>	

Abbildung 7.7: Vergleich OEDR2 [43]

Failure Modes				
ODD Boundary	<ul style="list-style-type: none"> <li>• Detect and Respond to ODD Boundary Transition</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Unanticipated Weather or Lighting Conditions Outside of Vehicle's Capability (e.g., rainstorm)</li> </ul>	<ul style="list-style-type: none"> <li>• Detect System Engagement/Disengagement Conditions Including Limitations by Location, Operating Condition, or Component Malfunction</li> </ul>	<ul style="list-style-type: none"> <li>• </li> </ul>
Degraded Performance/Health Monitoring	<ul style="list-style-type: none"> <li>• Fail-Safe/Fail-Operational Mechanisms</li> </ul>	<ul style="list-style-type: none"> <li>• Detect and Respond to Non-Collision Safety Situations (e.g., vehicle doors ajar)</li> <li>• Detect and Respond to Conditions Involving Vehicle, System, or Component-Level Failures or Faults (e.g., power failure, sensing failure, sensing obstruction, computing failure, fault handling or response)</li> <li>• Detect and Respond to Vehicle Control Loss (e.g., reduced road friction)</li> <li>• Moving to a Minimum Risk Condition When Exiting the Travel Lane is Not Possible</li> </ul>	<ul style="list-style-type: none"> <li>• Park on the Shoulder or Transition the Vehicle to a Minimal Risk State (Not Required for SAE L3)</li> </ul>	<ul style="list-style-type: none"> <li>• Control Loss Without Prior Vehicle Action</li> <li>• Evasive Action Without Prior Vehicle Maneuver</li> <li>• Control Loss With Prior Vehicle Action</li> <li>• Non-Collision Incident</li> <li>• Evasive Action With Prior Vehicle Maneuver</li> <li>• Vehicle Failure</li> <li>• Animal Crash Without Prior Vehicle Maneuver</li> <li>• Road Edge Departure Without Prior Vehicle Maneuver</li> <li>• Pedestrian Crash Without Prior Vehicle Maneuver</li> <li>• Road Edge Departure With Prior Vehicle Maneuver</li> <li>• Pedestrian Crash With Prior Vehicle Maneuver</li> <li>• Pedalcyclist Crash Without Prior Vehicle Maneuver</li> </ul>

Abbildung 7.8: Vergleich Ausfall Modi [43]

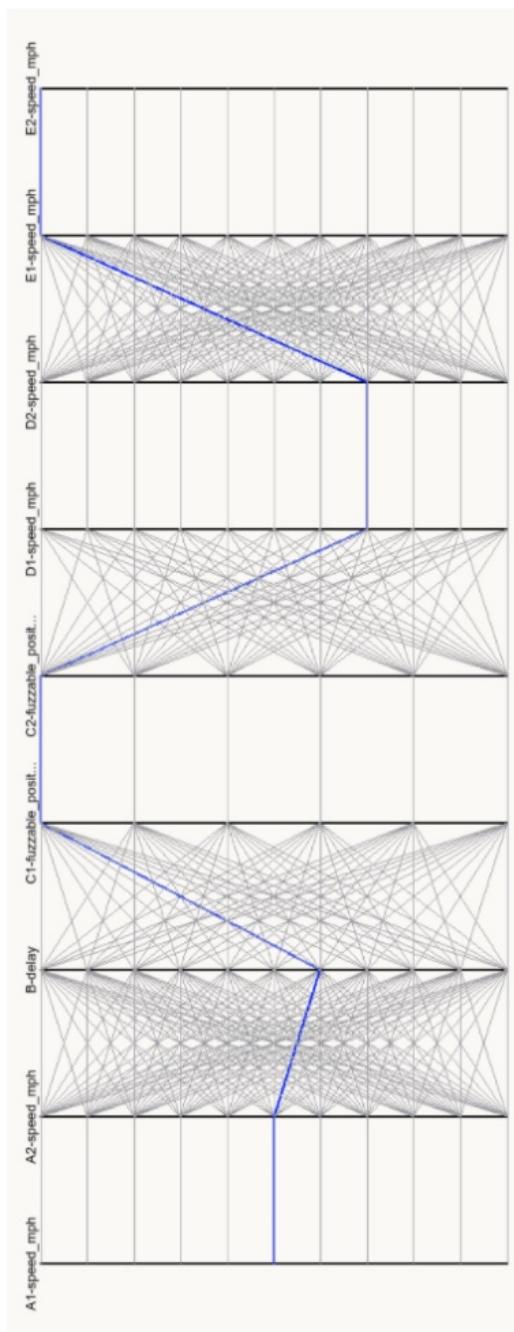


Abbildung 7.9: waymo Fuzzing Prozess [5]

# PEGASUS

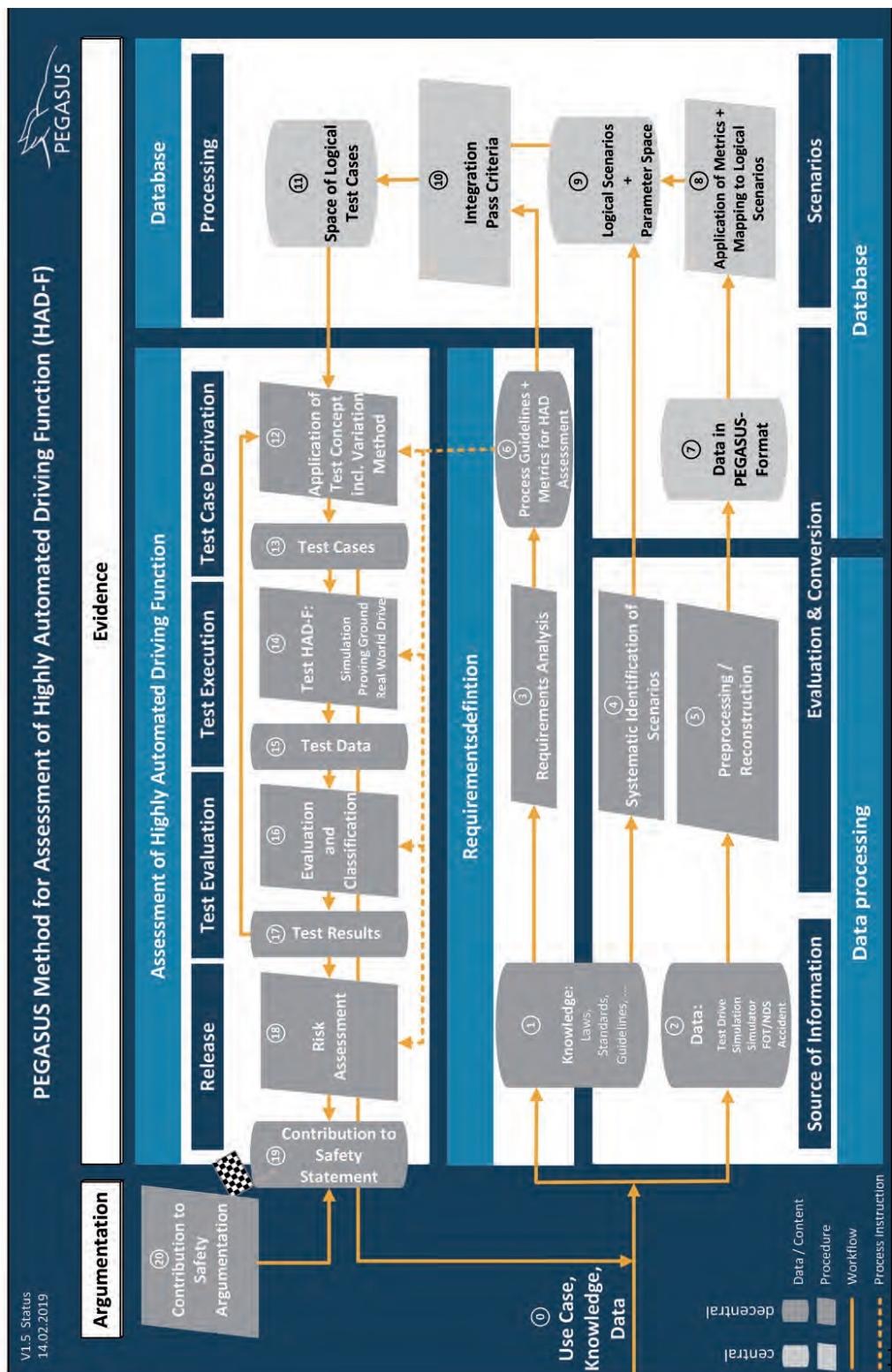


Abbildung 7.10: PEGASUS methode for assessment of highly automated driving function (HAD-F) [10]

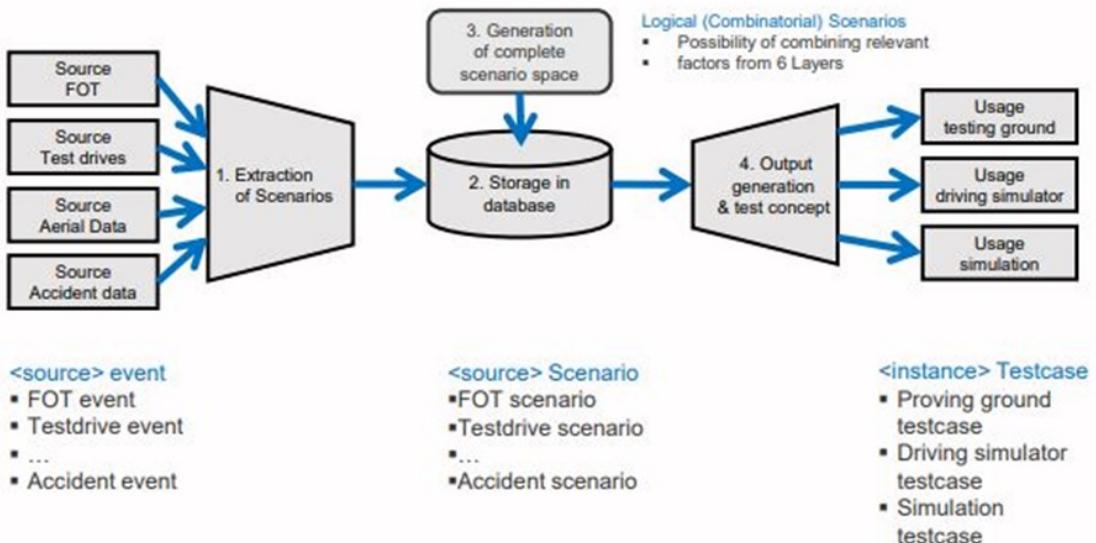


Abbildung 7.11: Pegasus, Datenbank und Szenarien extrahieren [10]

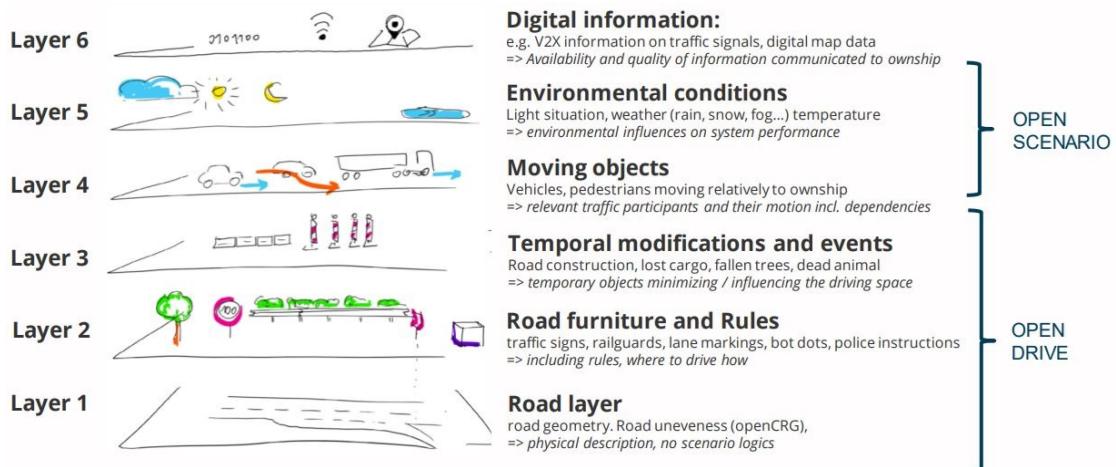


Abbildung 7.12: Pegasus, 6 Layers Szenario [10]

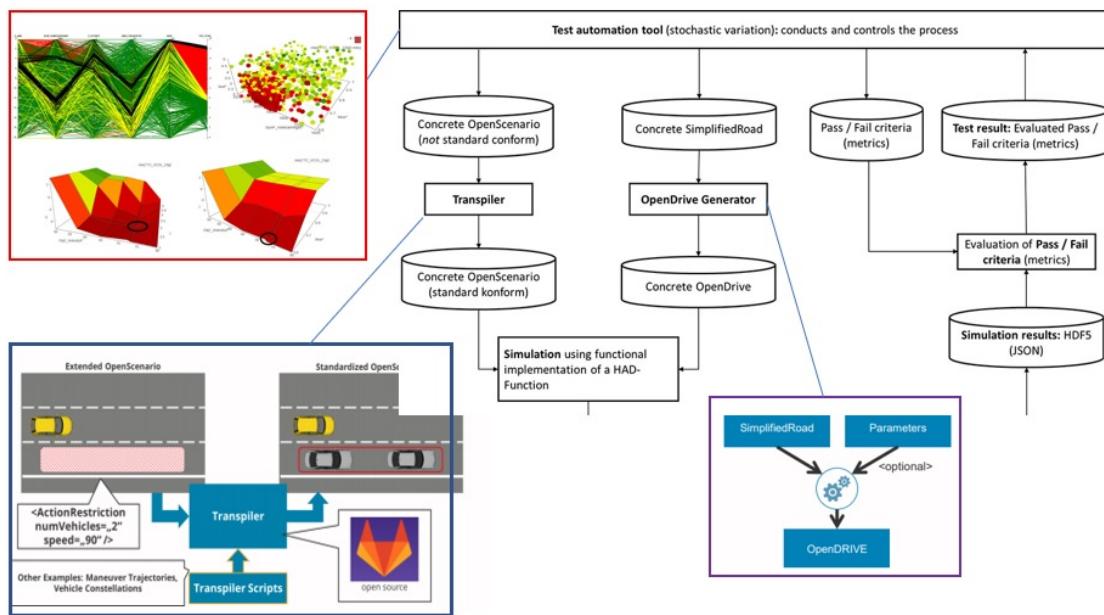


Abbildung 7.13: Pegasus, test automation tool [10]

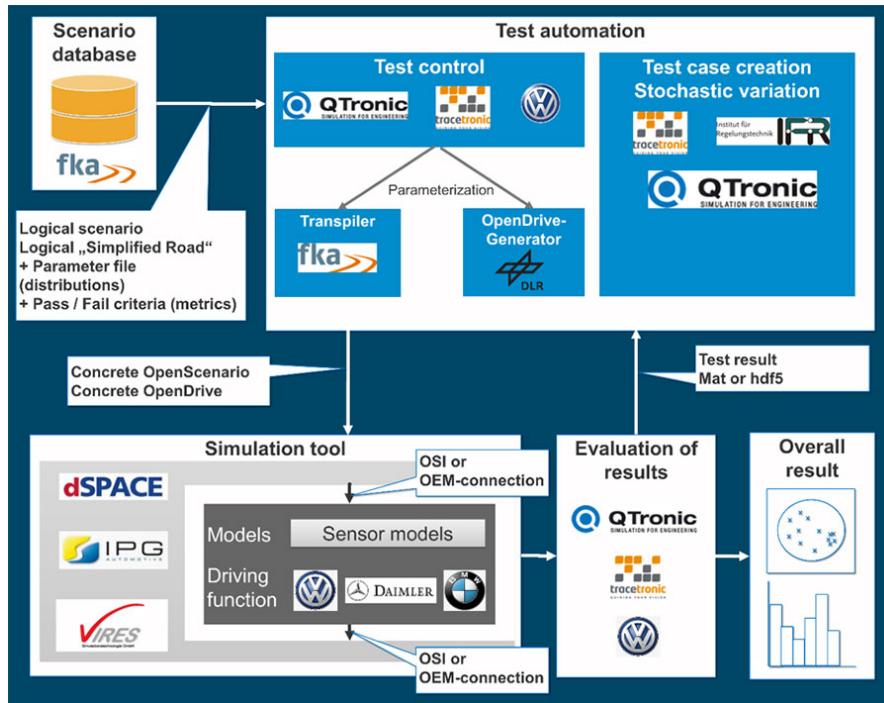


Abbildung 7.14: Übersicht der PEGASUS SW-Architektur inklusive der Umgebungssimulation [10]

# ADORe

Categories of Behavioral Competencies	NHTSA Testable Cases	Waymo Voluntary Safety Self-Assessment	ADORe Demos & Test Scenarios(current state)
<b>Tactical Maneuvers</b>			
1-Parking	• Parking	• Navigate a Parking Lot and Locate Spaces • Make Appropriate Reversing Maneuvers	• test001_parking.launch <a href="#">🔗</a>
2-Lane Maintenance & Car Following	• Car Following • Speed Maintenance • Lane Centering • Enhancing Conspicuity (headlights)	• Detect and Respond to Speed Limit Changes and Speed Advisories • Detect and Respond to Encroaching Oncoming Vehicles • Perform Car Following (Including Stop and Go)	• test002_lane_following.launch <a href="#">🔗</a>
3-Lane Change (e.g., overtake, merge)	• Lane Switching/ Overtaking • Enhancing Conspicuity (e.g., blinkers) • Merge (high & low speed)	• Perform High-Speed Merge (e.g., Freeway) • Perform Low-Speed Merge • Move Out of the Travel Lane and Park (e.g., to the Shoulder for Minimal Risk) • Detect and Respond to Encroaching Oncoming Vehicles • Detect Passing and No Passing Zones and Perform Passing Maneuvers • Perform Lane Changes	• test003_lane_change.launch <a href="#">🔗</a> • test003_lane_change_merge.launch <a href="#">🔗</a>
4-Navigate Intersection: • Type: Signalized, Nonsignalized, Roundabout, Rail Crossing • Turn: Left/ Right/ Straight	• Navigate On/Off Ramps • Roundabouts • Intersection (left, right, straight) • Crosswalk • U-Turn	• Perform Car Following (Including Stop and Go) • Navigate Intersections and Perform Turns • Navigate Roundabouts • Navigate Railroad Crossings	• test004_navigate_intersection_onramp.launch <a href="#">🔗</a> • test004_navigate_intersection_offramp.launch <a href="#">🔗</a> • test004_navigate_intersection_Roundabouts.launch <a href="#">🔗</a> • test004_navigate_intersection_lrs.launch <a href="#">🔗</a> • test004_navigate_intersection_crosswalk.launch <a href="#">🔗</a> • test004_navigate_intersection_uturn.launch <a href="#">🔗</a>
Test Procedure and Goal Description	Status	Evaluation Metrics	Parameter variation
Please refer to the documentation <a href="#">🔗</a>	3	B-C-D-E-F-G	
Please refer to the documentation <a href="#">🔗</a>	3	B-C-D-E-F-G	
Please refer to the documentation <a href="#">🔗</a>	3	B-C-D-E-F-G	
Please refer to the documentation <a href="#">🔗</a>	3	B-C-D-E-F-G	

Tabelle 7.7: NHTSA, Waymo und ADORe Vergleich Tabelle

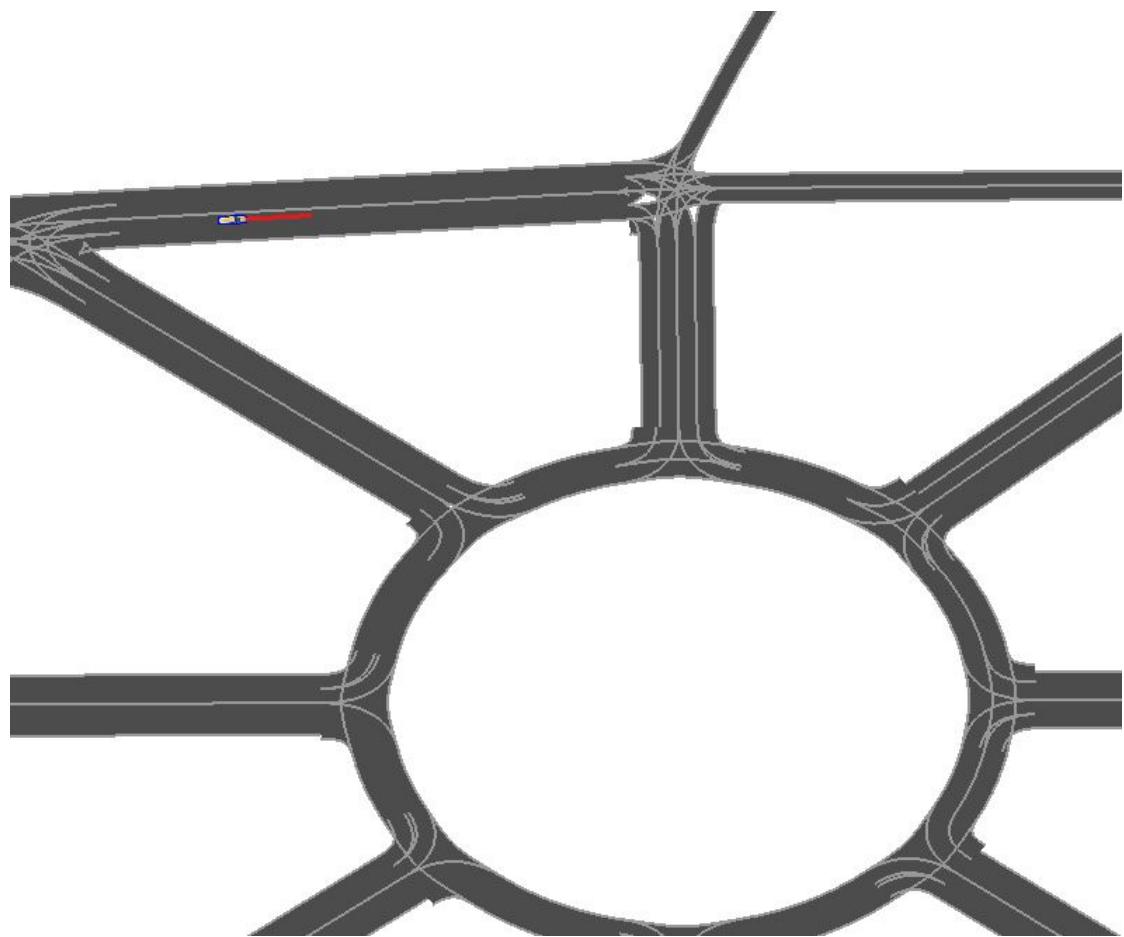
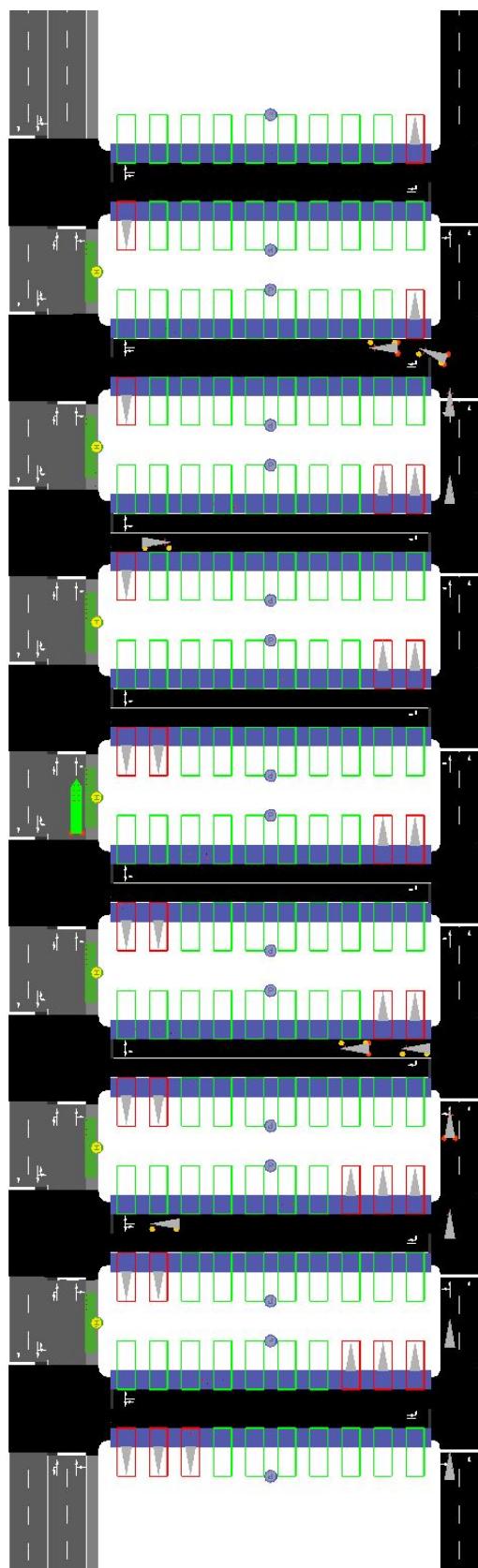


Abbildung 7.15: Überblick test000\_template.launch



108

Abbildung 7.16: Überblick test001\_parking.launch

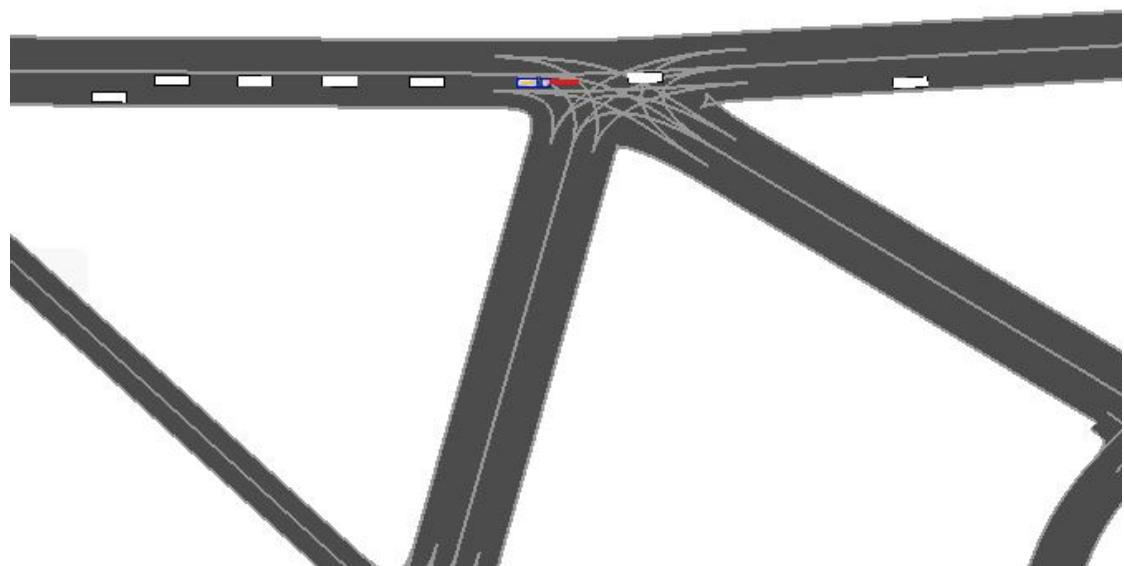


Abbildung 7.17: Überblick test002\_lane\_following.launch

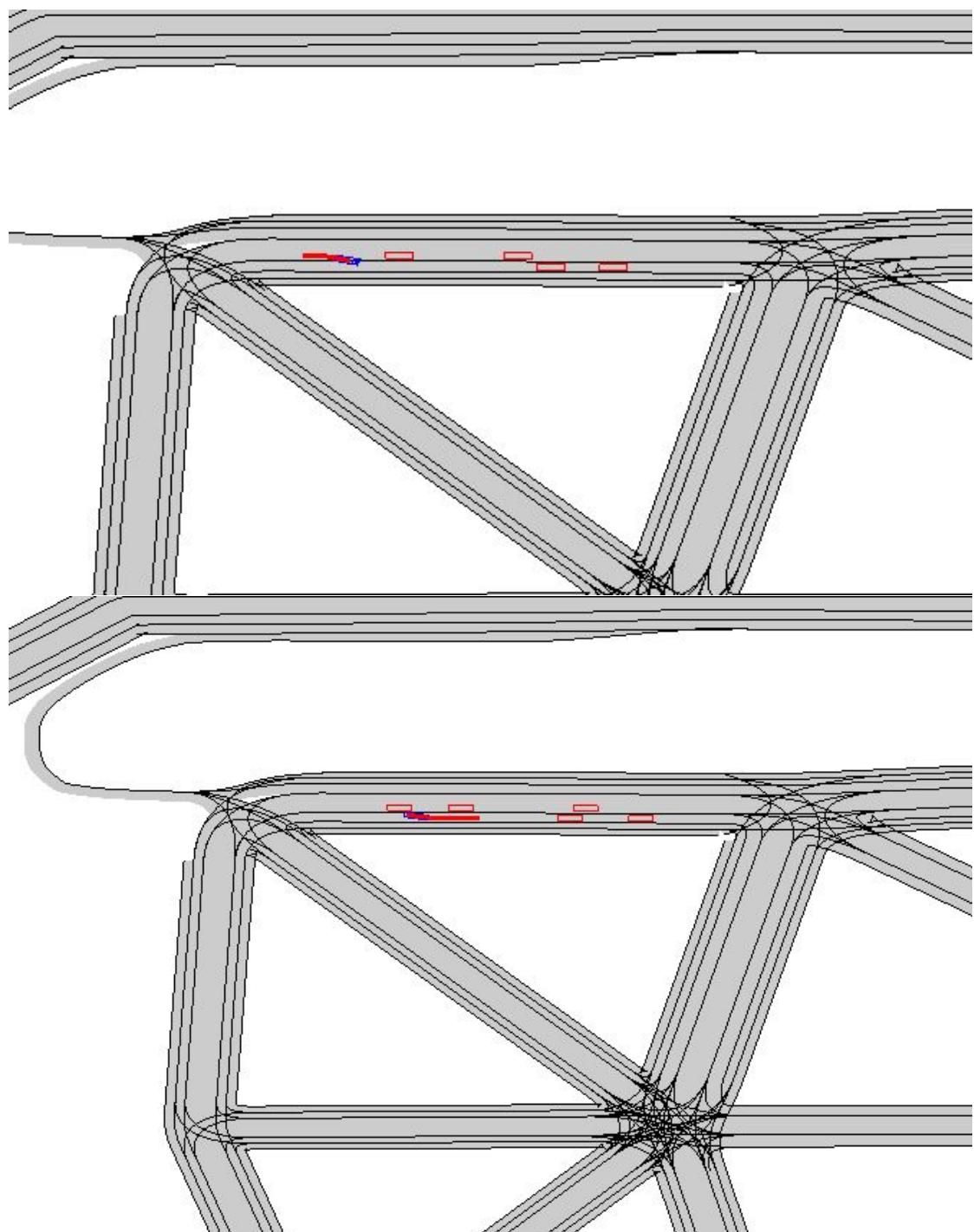


Abbildung 7.18: Überblick test003\_lane\_change

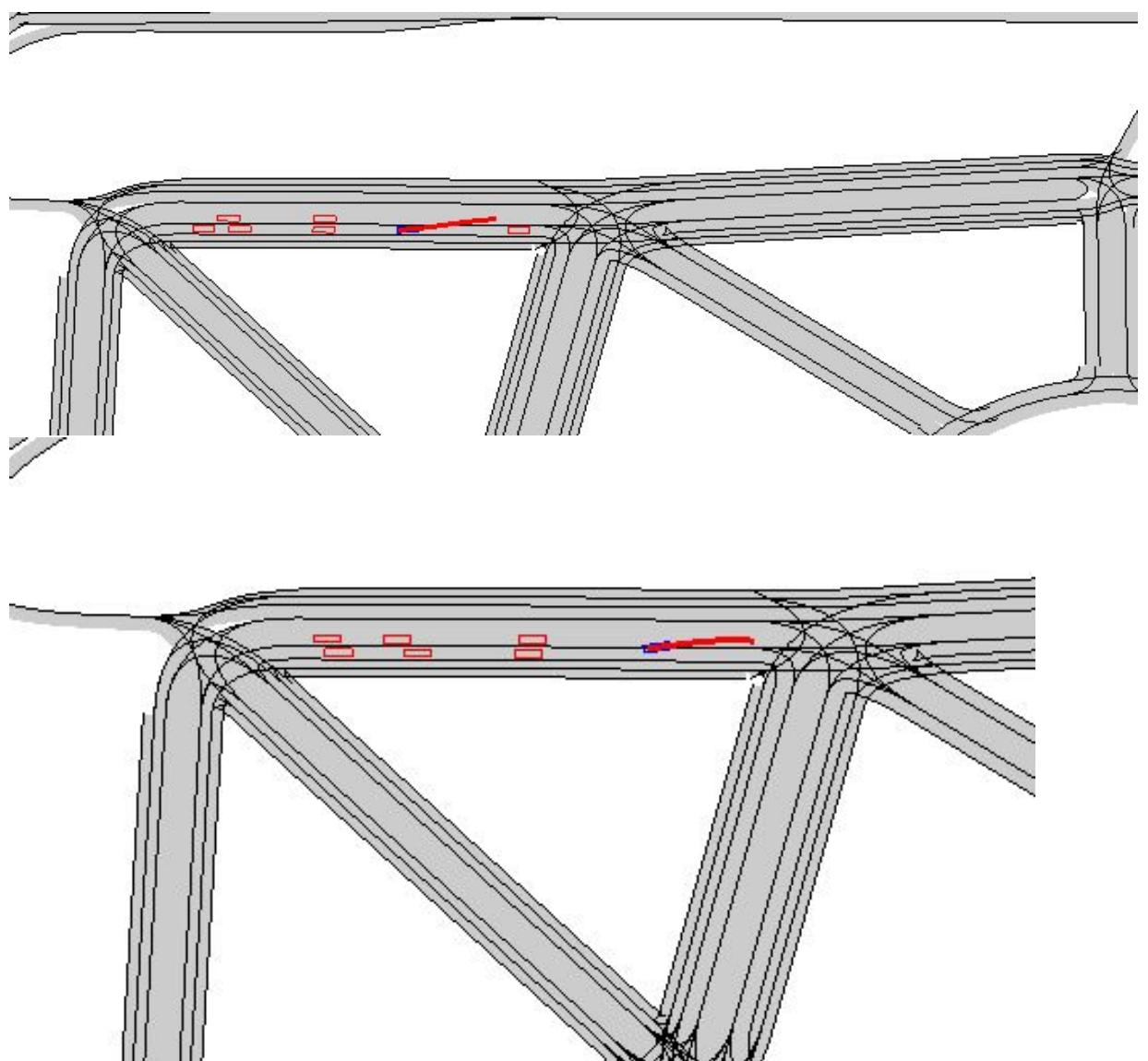


Abbildung 7.19: Überblick test003\_lane\_change\_merge

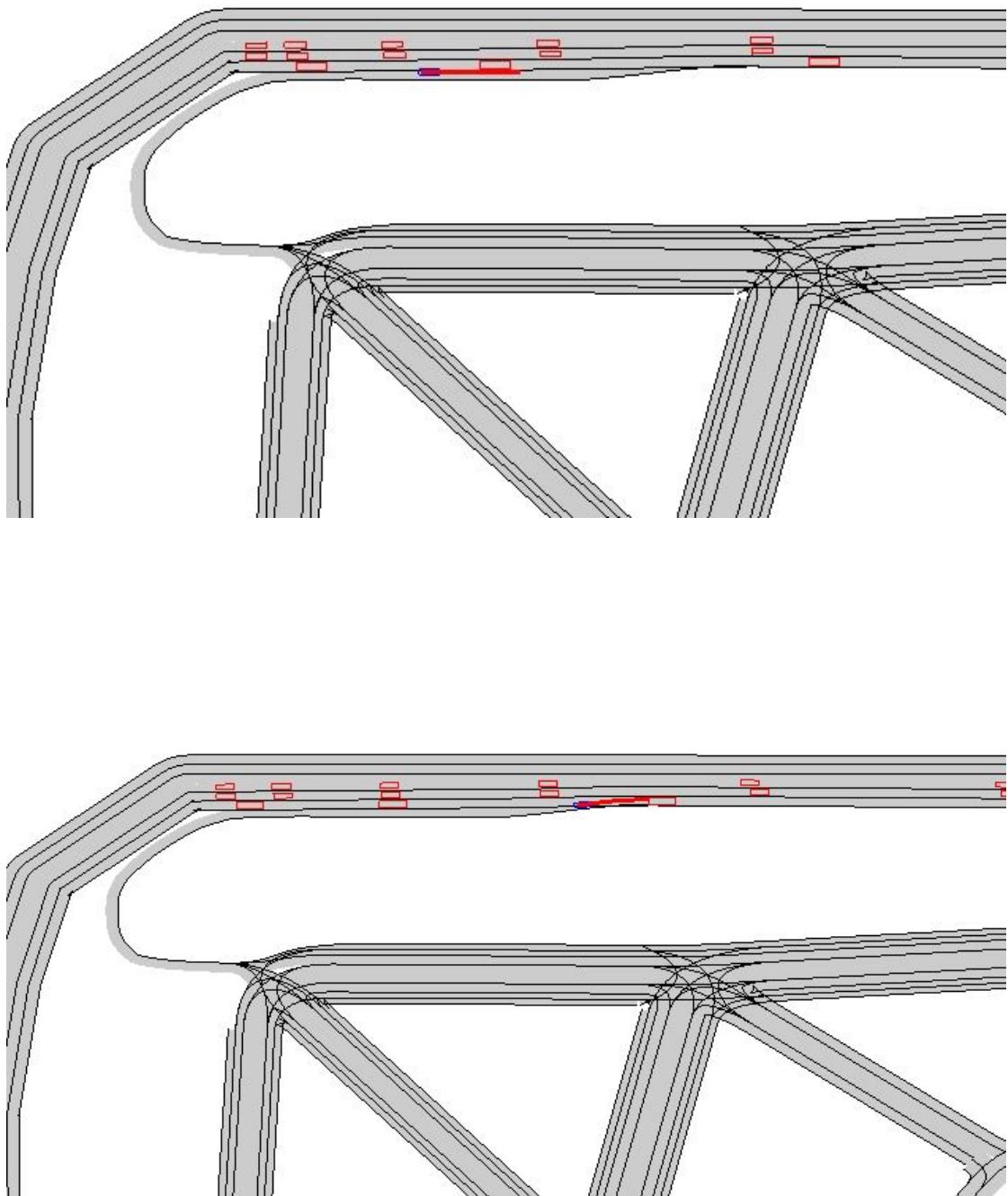


Abbildung 7.20: Überblick test004\_navigate\_intersection\_onramp

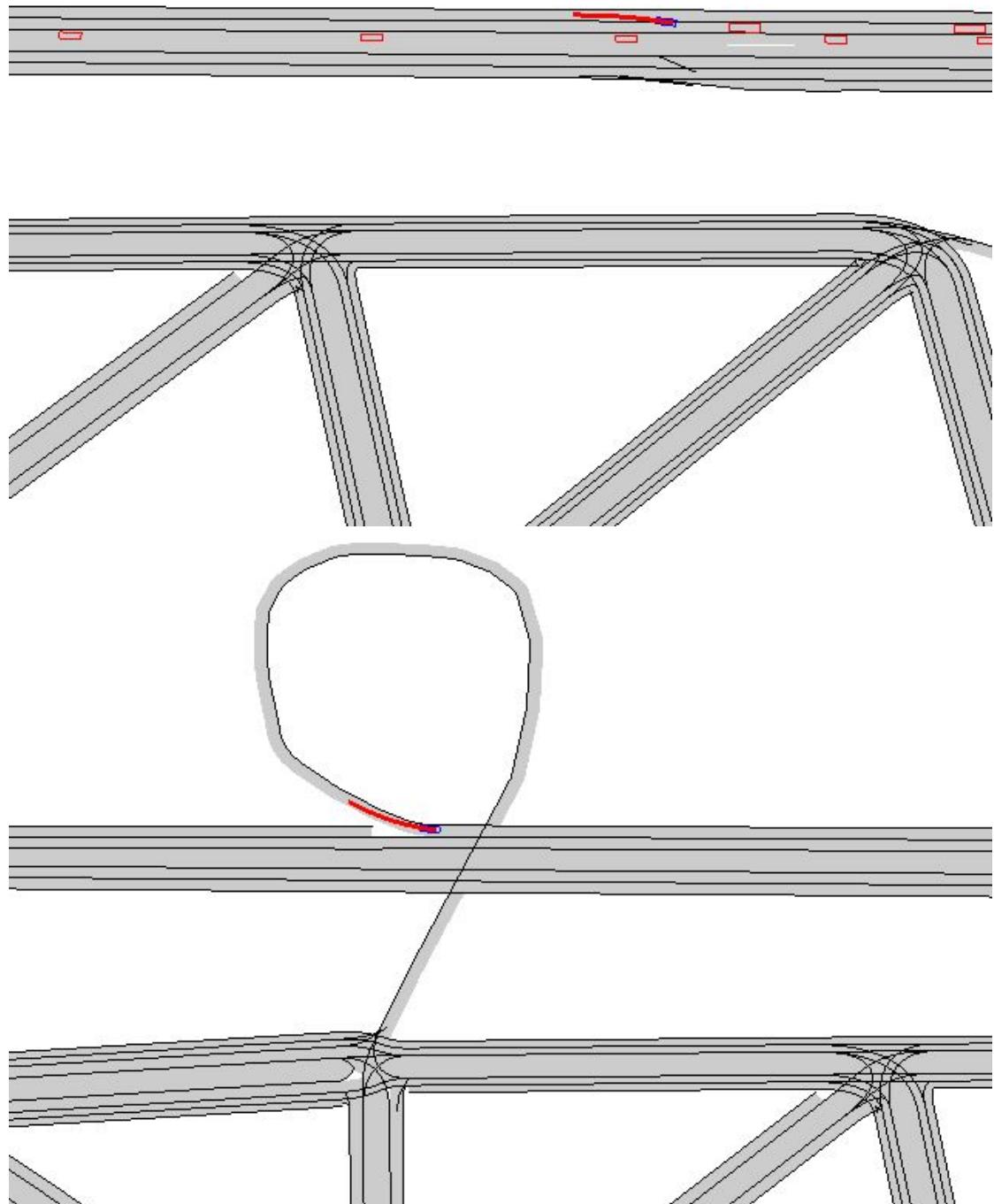


Abbildung 7.21: Überblick test004\_navigate\_intersection\_offramp

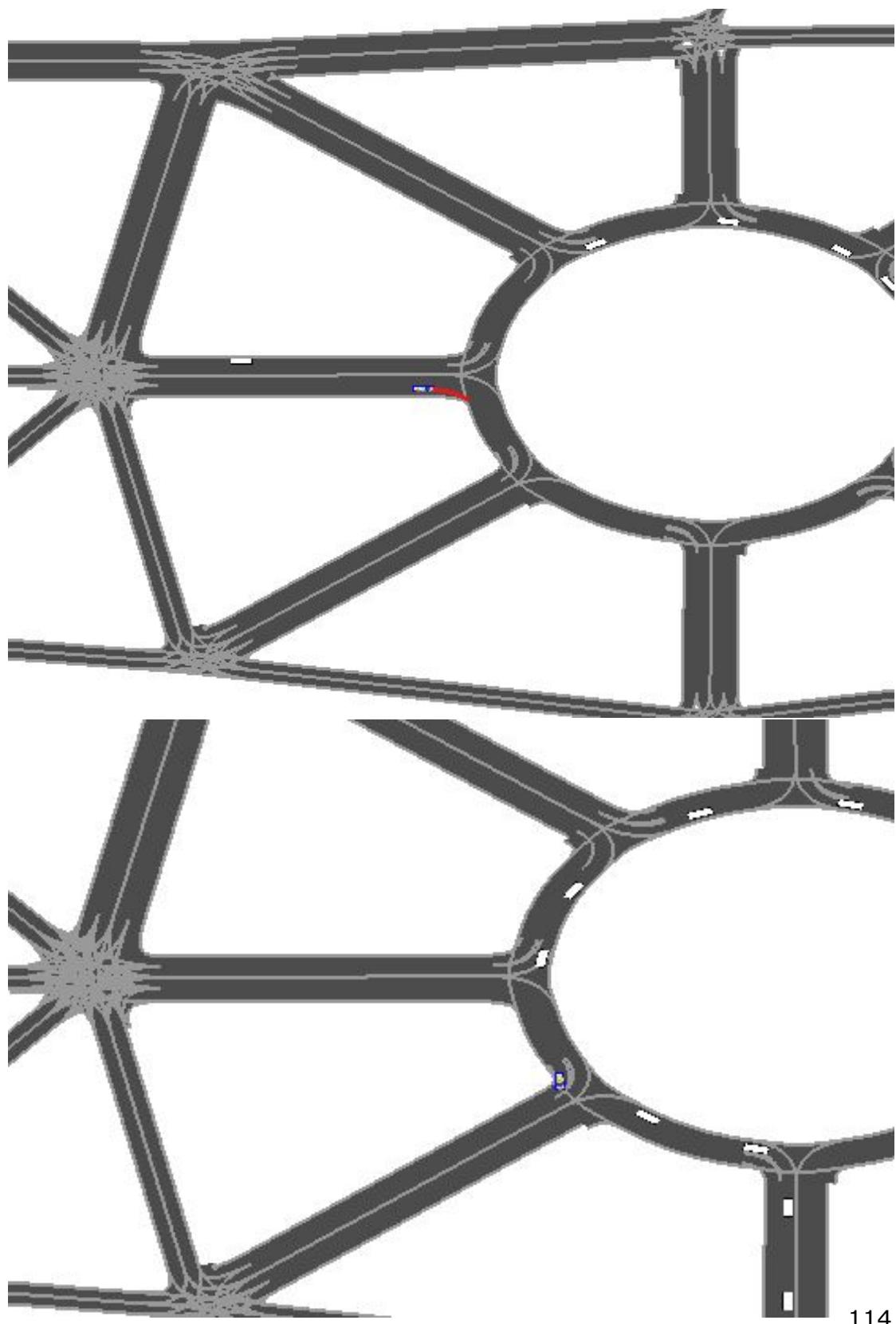
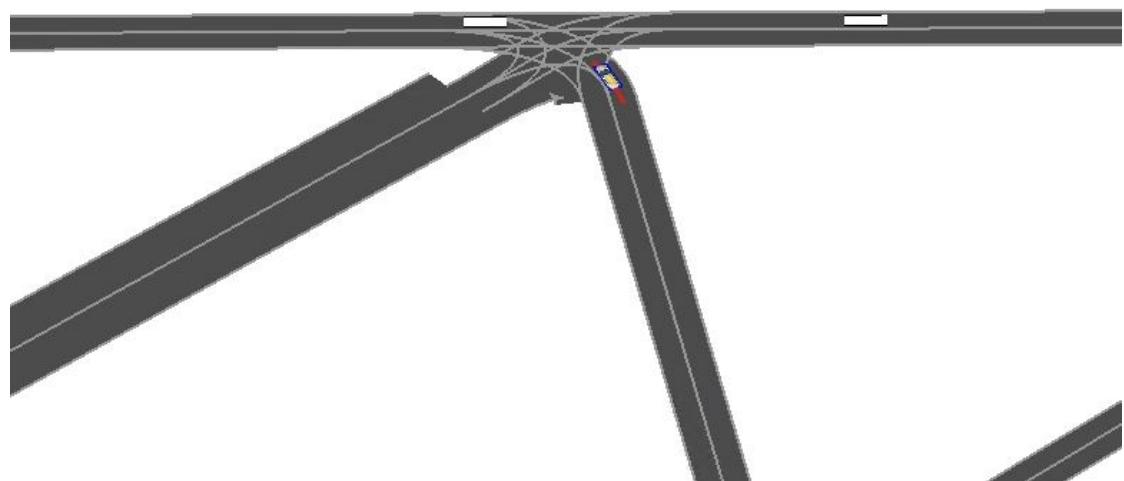
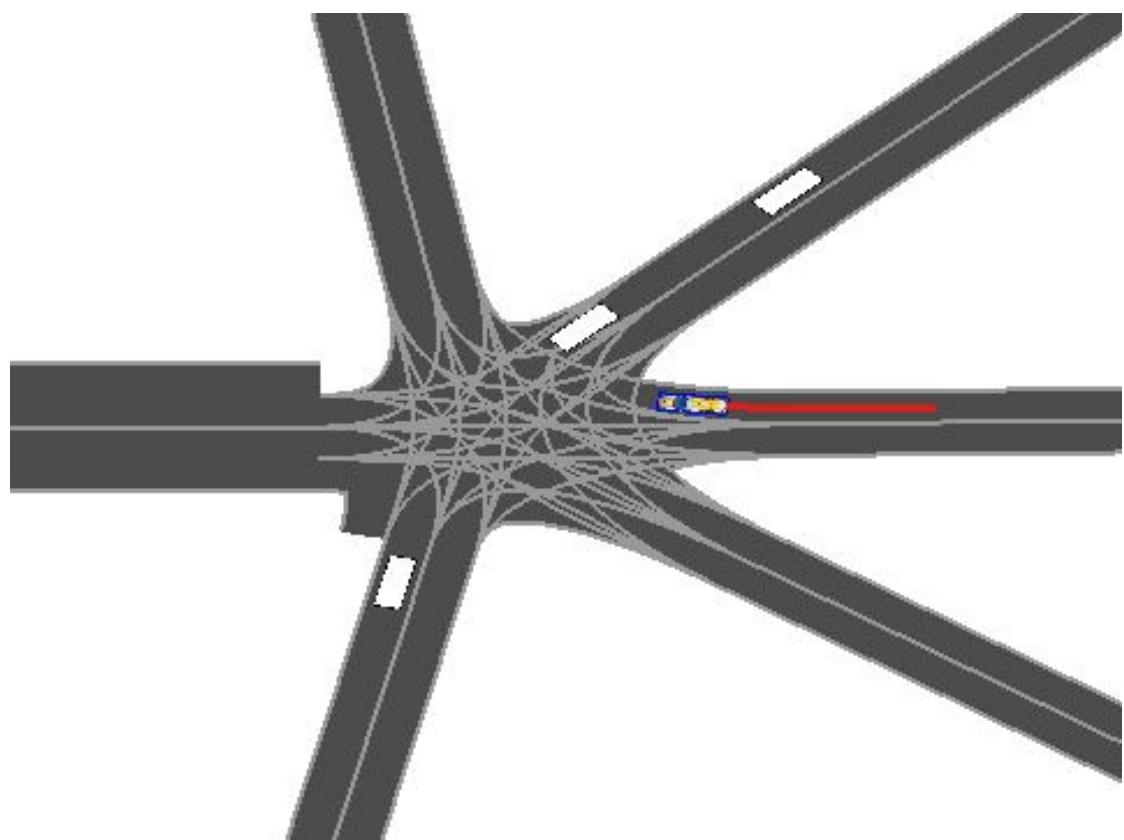


Abbildung 7.22: Überblick test004\_navigate\_intersection\_Roundabouts



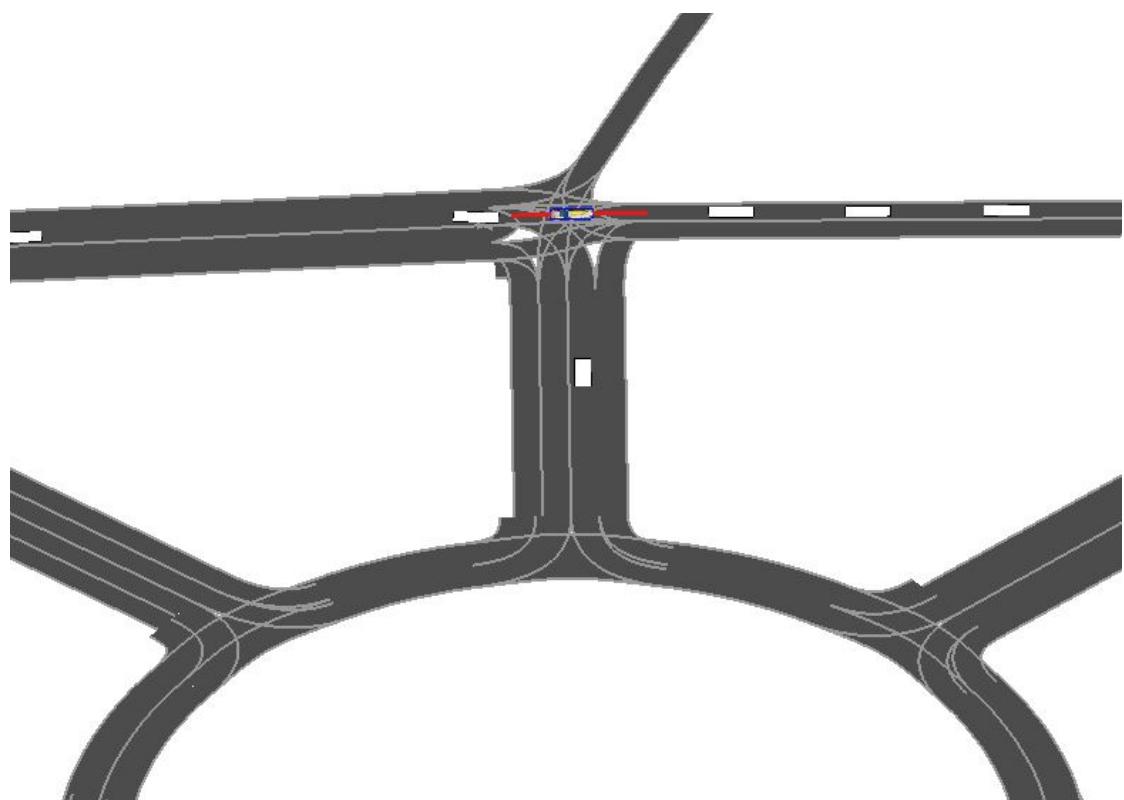


Abbildung 7.23: Überblick test004\_navigate\_intersection\_lrs

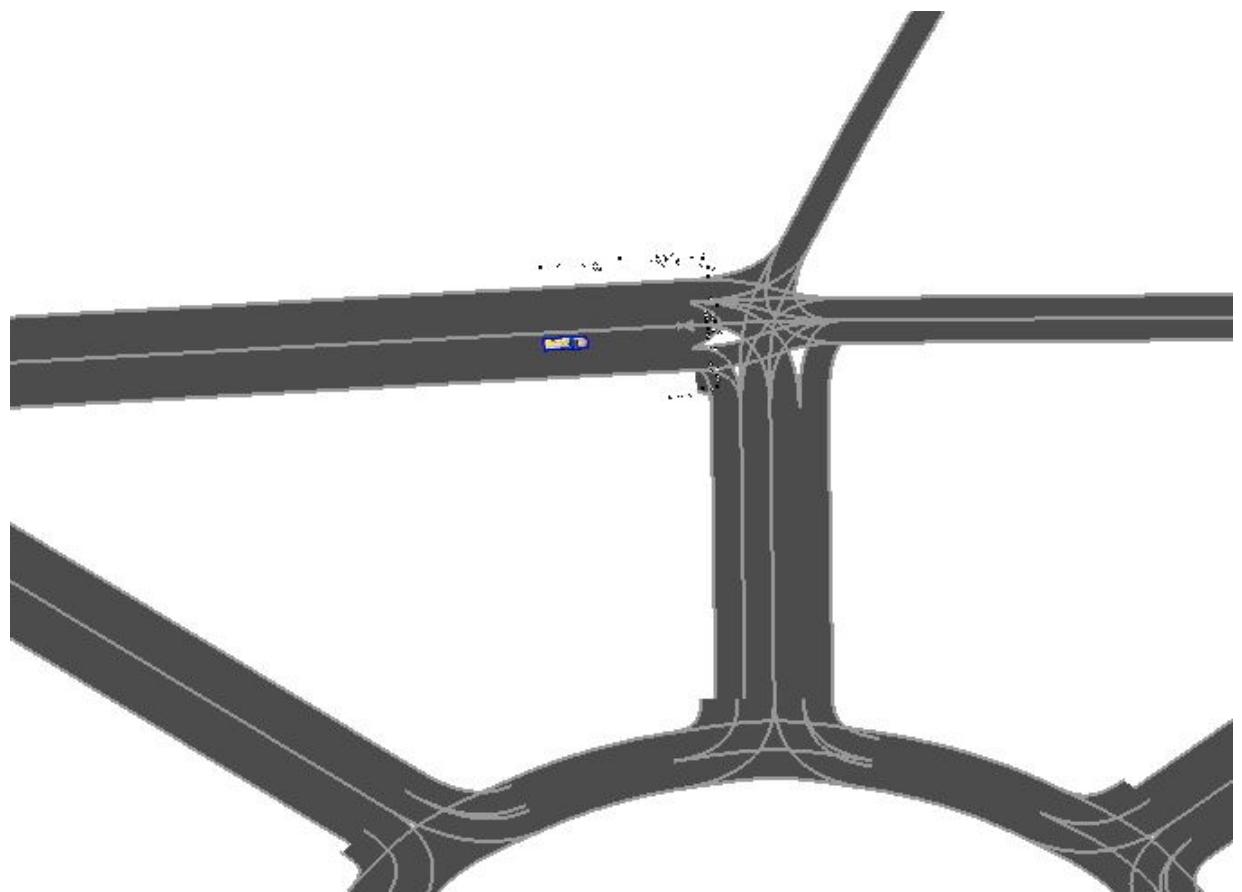


Abbildung 7.24: Überblick test004\_navigate\_intersection\_crosswalk

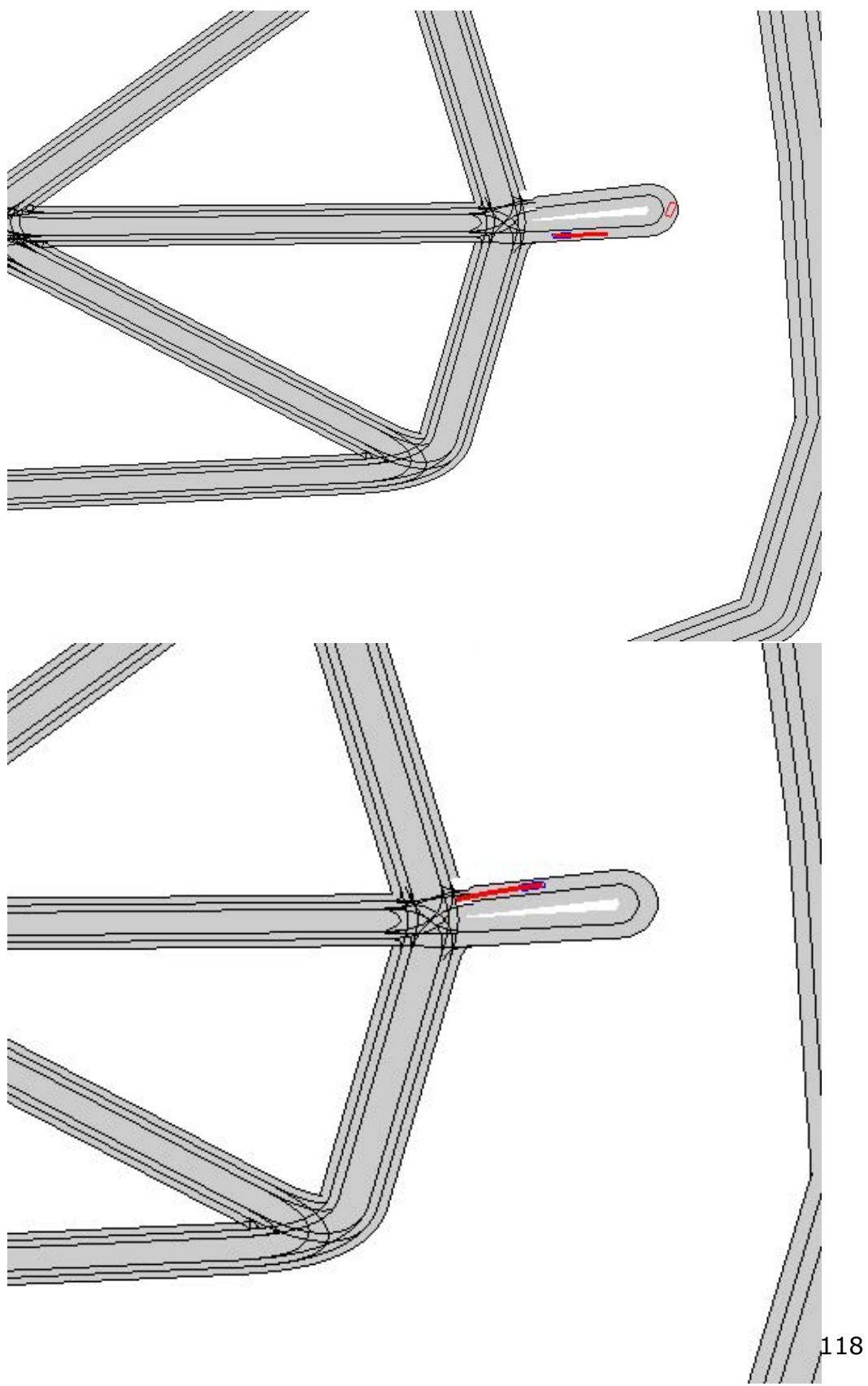


Abbildung 7.25: Überblick test004\_navigate\_intersection\_uturn

	<b>Symbol</b>	<b>Erklärung</b>
not A	$\neg A$	Negation (not)
A und B	$A \wedge B$	Konjunktion (AND)
A oder B	$A \vee B$	Disjunktion (OR)
wenn A dann B	$A \Rightarrow B$	Implikation (if A then B)
nur wenn A dann B	$A \Leftrightarrow B$	Äquivalenz (Bidirektional: if and only if)
auf alle Pfade folgt $\varphi$	$\forall \varphi$	für alle Pfade, $\varphi$ ist wahr
auf mindestens einem Pfad folgt $\varphi$	$\exists \varphi$	es existiert einen Pfad, sodass $\varphi$ ist wahr
<b>NeXt:</b> N/X $\varphi$	$\circ \varphi$	<b>NeXt:</b> $\varphi$ gilt am nächsten Zustand(State)
<b>Globally/always:</b> G $\varphi$	$\Box \varphi$	<b>Global:</b> $\varphi$ gilt auf dem kompletten nachfolgenden Pfad
<b>Finally/Future:</b> F $\varphi$	$\diamond \varphi$	<b>Finally:</b> $\varphi$ gilt irgendwann auf dem nachfolgenden Pfad.
<b>Until:</b> $\Psi U \varphi$	$\Psi U \varphi$	<b>Until:</b> $\Psi$ gilt mindestens so lange bis $\varphi$ im nächsten Zustand gilt, keine gemeinsame Zustand erlaubt.
<b>Release:</b> $\Psi R \varphi$	$\Psi R \varphi$	<b>Release:</b> $\Psi$ gilt einschließlich bis zur ersten Position, an der $\varphi$ gilt ( $\varphi$ wird an dieser Position freigegeben "released"), oder für immer, wenn eine solche Position nicht existiert.

Tabelle 7.8: Logik, temporale und Pfad-Operatoren [82, 81]

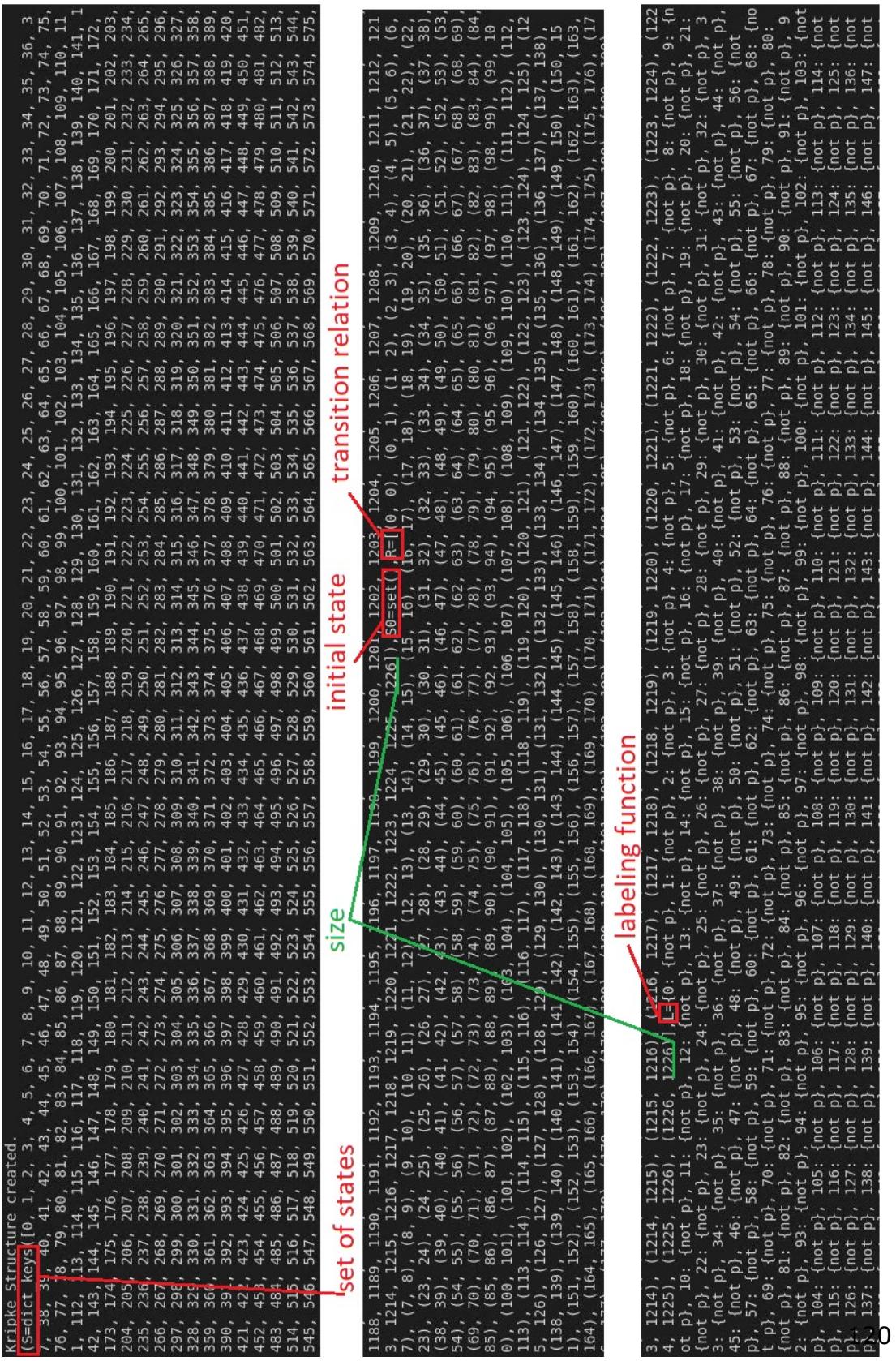


Abbildung 7.26: Kripke-Struktur df1

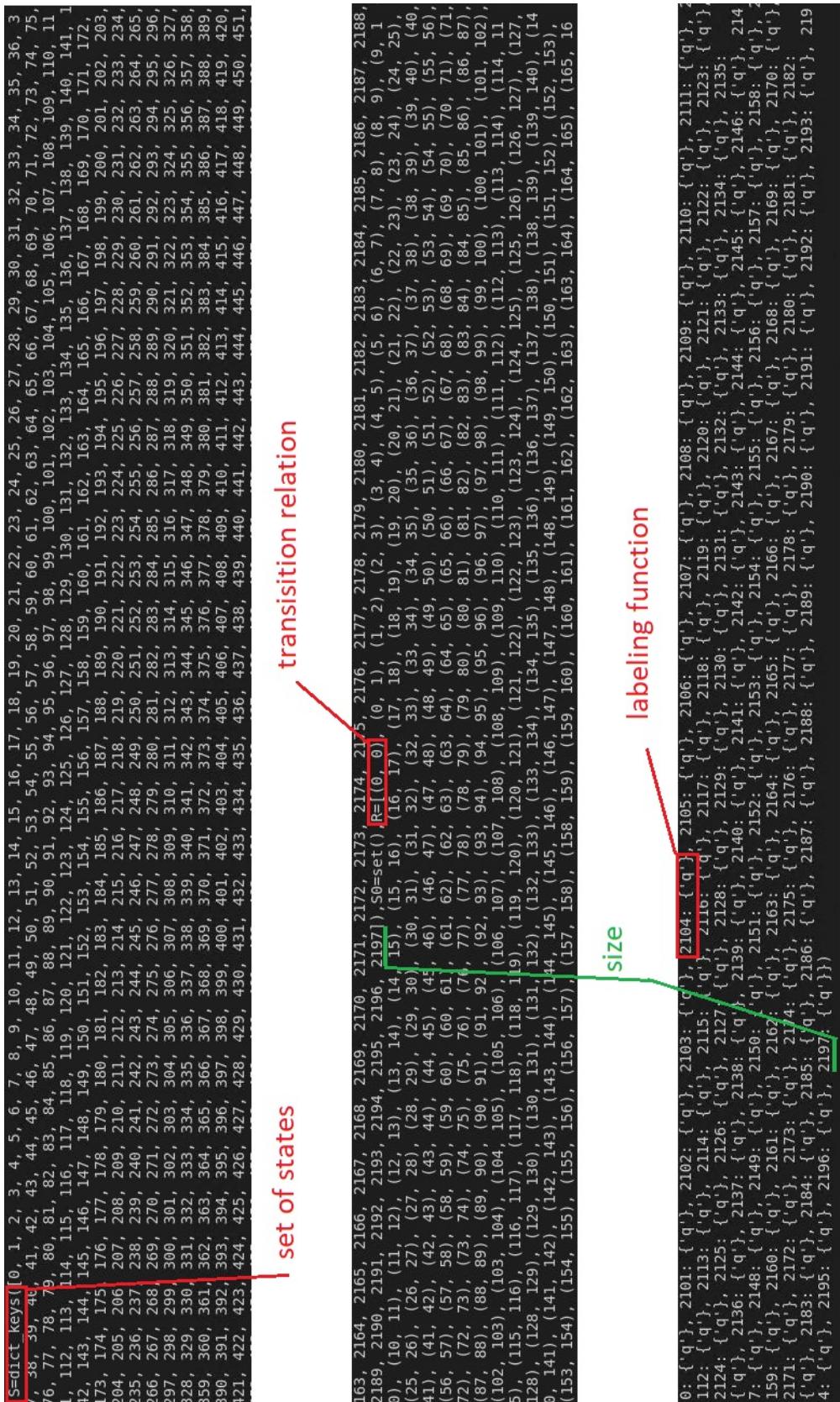


Abbildung 7.27: Kripke-Struktur  $\text{df2}$

# Codeauflistung

```
1 <!--
*****Copyright (C) 2017-2020 German Aerospace Center (DLR).
2 * Eclipse ADORe, Automated Driving Open Research https://eclipse.org/adore
3 *
4 * This program and the accompanying materials are made available under the
5 * terms of the Eclipse Public License 2.0 which is available at
6 * http://www.eclipse.org/legal/epl-2.0.
7 *
8 * SPDX-License-Identifier: EPL-2.0
9 *
10 *
11 * Contributors:
12 * Sami Dhiab - Test Scenarios Generation
13 * test000:
14 * template for test generation
15 *
*****-->
16 <launch>
17   <group ns="vehicle0">
18     <arg name="record_rosbag" default="true" />
19     <!-- this starts the rosbag recording - name of the bag should be same as
        launchfile -->
20     <!--node pkg="rosbag" type="record" name="sim_rosbag_recorder" args="
        record -0 /home/sd/outputdir/test_ci.bag -a"
        if="$(arg record_rosbag)"-->
21     <!-- Load Timer Node , output onto Terminal -->
22     <node name="timer" pkg="adore_if_ros" type="adore_timer_node" output="
        screen" launch-prefix="xterm -e"/>
23     <node name="sumotraffic2ros_node" pkg="sumo_if_ros" type="
        sumotraffic2ros_node" launch-prefix="xterm -e "/> <!-- output="screen"-->
24     <!-- Load sut Parameter -->
25     <include file="$(dirname)/sut.launch"/>
26     <!-- Load Track Parameter -->
27     <param name="PARAMS/track" value="$(dirname)/../tracks/complex/complex.
        xodr" type="str" />
28     <param name="PARAMS/map_provider/activate_plotting" value="true" type="
        bool" />
29     <node name="plotter" pkg="adore_if_ros" type="adore_fancybird_node"/>
```

```

31    <!-- assign sim id -->
32    <param name="simulationID" value="0" type="int" />
33    <!-- publish startpos and orientation on topic -->
34    <node pkg="rostopic" type="rostopic" name="posereset" args="pub /vehicle0
/SIM/ResetVehiclePose geometry_msgs/Pose '{position: {x: -170.60, y:
155.40, z: 0},orientation: {w: 1}}'"/>
35    <!-- change speed parameter of ego vehicle -->
36    <param name="PARAMS/tactical_planner/global_speed_limit" value="10" type=
"double" /> <!---22.66,117.29, 107.41,1.92, -135.75,48.10-->
37    <!-- set the goal position -->
38    <node pkg="rostopic" type="rostopic" name="navgoal" args="pub /vehicle0/
ENV/NavigationGoal adore_if_ros_msg/NavigationGoal '{target: {x: 36.25,
y: 163.22, z: 0}}'"/>
39    <!-- Load Sumo params-->
40
41    <arg name="sumocfg" value="$(dirname)../../tracks/complex/complex.sumocfg"/
>
42    <arg name="sumohome" value="$(env SUMO_HOME)"/>
43    <arg name="traciport" value="1337"/>
44    <arg name="sumostep" value="0.01"/>
45    <node name="sumo" pkg="sumo_if_ros" type="start_sumo.sh" output="screen"
launch-prefix="xterm -e " args="$(arg sumohome) $(arg sumocfg) $(arg
traciport) $(arg sumostep)"/>
46  </group>
47 </launch>

```

Listing 7.1: test000\_template.launch

```

1 data:
2   time: 45.0
3   motion_state:
4     header:
5       seq: 0
6       stamp:
7         secs: 45
8         nsecs: 0
9       frame_id: ''
10      child_frame_id: ''
11      pose:
12        pose:
13          position:
14            x: 40.76871233860872
15            y: 163.01981344545305
16            z: 0.0
17          orientation:
18            x: 0.0
19            y: 0.0
20            z: 0.03545783638575293
21            w: 0.9993711732078533
22          covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
23        twist:
24          twist:

```

```

25     linear:
26         x: 4.999278685454047
27         y: 0.0
28         z: 0.0
29     angular:
30         x: 0.0
31         y: 0.0
32         z: 0.0
33     covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
34     0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
35     0.0, 0.0, 0.0, 0.0, 0.0]
36     shape:
37         type: 0
38         dimensions: [5.0, 1.8, 1.5]
39     dimensions_variance: 0.0
40     classification:
41         type_id: 0
42         classification_certainty: 0
43         existance_certainty: 100
44     leftIndicatorOn: False
45     leftIndicator_certainty: 0
46     rightIndicatorOn: False
47     rightIndicator_certainty: 0
48     brakeLightOn: False
49     brakeLight_certainty: 0
50     lowBeamOn: False
51     lowBeam_certainty: 0
52     highBeamOn: False
53     highBeam_certainty: 0
54     v2xStationID: 0
55     v2xStationID_certainty: 0
56     trackingID: 1001
57     detection_by_sensor: 0
58
59     -
60     data:
61         time: 45.0
62         motion_state:
63             header:
64                 seq: 0
65                 stamp:
66                     secs: 45
67                     nsecs: 0
68                     frame_id: &apos;&apos;
69                     child_frame_id: &apos;&apos;;
70             pose:
71                 pose:
72                     position:
73                         x: 65.67909104357948
74                         y: 163.80164469456452
75                         z: 0.0
76                     orientation:
77                         x: 0.0
78                         y: 0.0

```

```

76         z: 0.005216518541127433
77         w: 0.9999863938745918
78         covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
79          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
80          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
81         twist:
82             twist:
83                 linear:
84                     x: 4.996935874406481
85                     y: 0.0
86                     z: 0.0
87                 angular:
88                     x: 0.0
89                     y: 0.0
90                     z: 0.0
91                     covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
92                      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
93                      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
94         shape:
95             type: 0
96             dimensions: [5.0, 1.8, 1.5]
97             dimensions_variance: 0.0
98             classification:
99                 type_id: 0
100                classification_certainty: 0
101                existance_certainty: 100
102                leftIndicatorOn: False
103                leftIndicator_certainty: 0
104                rightIndicatorOn: False
105                rightIndicator_certainty: 0
106                brakeLightOn: False
107                brakeLight_certainty: 0
108                lowBeamOn: False
109                lowBeam_certainty: 0
110                highBeamOn: False
111                highBeam_certainty: 0
112                v2xStationID: 0
113                v2xStationID_certainty: 0
114                trackingID: 1000
115                detection_by_sensor: 0

```

Listing 7.2: json traffic

```

1 ## gitlab-ci yaml: launch a pipeline on every new push (see Pipeline Gui)
2 ## Stages: 1-Build Adore,2-unit-test, 3-test Scenarios , 4-evaluation deploy
3 ## 1: build adore in docker container
4 ## 2: run unit test after building
5 ## 3: launch all test scenarios and evaluate the results
6 ## 4: based onevaluation deploy the scenario:
7 ##     - publish scenario to production
8 ##     - derivate paramter for scenario
9 ## Jobs: Jobs are run once in stage 1 and 2, in parallel in stage 3 and 4
10    using the same runner
11 ## Branch: developCi

```

```

11 ## Ressources: Gitlab-ci, gitlab-runner, Docker
12 ## Version: Latest
13
14 variables:
15   SharedVol: "/home/sd/SharedVol"
16
17 stages:
18   - build
19   - unit_test
20   - ci_scenarios
21   - evaluation_deploy
22
23 build_adore:
24   stage: build
25   tags: [ci_test]
26   rules:
27     - if: '$CI_COMMIT_BRANCH == "developCi"'
28   script:
29     - sudo chmod 777 -R /home/sd/gitlab-runner/ || true
30     - docker stop $(docker ps -aq) || true ##stop running container: || true
31     =1 if no container already found
32     - docker rm $(docker ps -aq) || true ## remove them
33     - docker container prune ## Removes all stopped containers.
34     - sudo rm -r $SharedVol/ || true
35     - sudo mkdir -p $SharedVol/ || true
36     - sudo cp -r $CI_PROJECT_DIR/ $SharedVol/
37     - export adore_libadore_branch=developCi
38     - >
39       docker run
40         --name test_cont -idt --entrypoint=/bin/bash
41         -e adore_libadore_branch=$adore_libadore_branch
42         -e CI_JOB_TOKEN=$CI_JOB_TOKEN
43         -v /usr/share/adore_external_cache:/home/adore_external_cache
44         -v $SharedVol/adore/:/home/adore_repo
45         -v $SharedVol/adore/:/home/outputdir
46           gitlabciadore/adoreci:v0.1
47
48     - docker exec -t test_cont "/home/adore_repo/test/ci/runner_script.bash"
49     - sudo mkdir -p $CI_PROJECT_DIR/build/
50     - sudo cp -r $SharedVol/adore/build/* $CI_PROJECT_DIR/build/
51     - sudo chmod 777 -R /home/sd/gitlab-runner/ || true
52
53 artifacts:
54   when: always
55   paths:
56     - build/*.txt
      - build/*.log

```

Listing 7.3: Stage Build

```

1 class Analyser:
2
3   """Class for Parsing Arguments
4   and analysing all topics from command line"""

```

```

5
6     def __init__(self):
7         self.inputfile = None
8         self.outputfile = None
9         self.topics = None
10        self.results = []
11
12    def ParseArgument(self):
13        """Arguments parsing and storing in class variables"""
14
15        parser = argparse.ArgumentParser(description=description)
16        parser.add_argument('-i', '--input', type=str,
17                            help='select a bag file')
18        parser.add_argument('-o', '--output', type=str,
19                            help='write result to a file')
20        parser.add_argument('-t', '--topics', nargs='+',
21                            type=str, help='select topics from list, see help
for more infos')
22
23        args = parser.parse_args()
24        print('Parsing Arguments ..')
25        self.inputfile = args.input
26        self.outputfile = args.output
27        self.topics = args.topics

```

Listing 7.4: class Analyser: ParseArgument

```

1
2     def AnalyseTopics(self):
3         """ iterate throw all topics from the list"""
4         for i in range(50):
5             for topic in self.topics:
6                 if topic == '/vehicle'+str(i)+'/ENV/propositions':
7                     print('Analysing first topic:', topic)
8                     M = Monitor(self.inputfile, self.outputfile, topic)
9                     df = M.ReadBagFileByTopic(self.inputfile, topic)
10                    print('extracting topic from bag file')
11                    df1 = M.DataFrameExtractByTerm(df, 'term', IN_COLLISION)
12                    df2 = M.DataFrameExtractByTerm(df, 'term', NEAR_GOAL)
13                    K1 = M.GenerateKrypkeFromDataFrame(df1, IN_COLLISION, p)
14                    K2 = M.GenerateKrypkeFromDataFrame(df2, NEAR_GOAL, q)
15                    if K1 is False:
16                        print(
17                            'cannot check model for collision, either
dataframe is empty or no relevant data found')
18                        if K2 is False:
19                            print(
20                                'cannot check model for goal, either dataframe is
empty or no relevant data found')
21                        if K1 is not False:
22                            phi = M.GetFormula(IN_COLLISION, p)
23                            print('The No Collision Formula is: ', phi)
24                            M.CheckModel(K1, phi)
25                            self.results.append(M.Result)

```

```

26     if K2 is not False:
27         psi = M.GetFormula(NEAR_GOAL, q)
28         print('The Goal Achieving Formula is: ', psi)
29         M.CheckModel(K2, psi)
30         self.results.append(M.Result)

```

Listing 7.5: class Analyser: Topic 1

```

1     elif topic == '/vehicle'+str(i)+'/odom':
2         print('Analysing third topic:', topic)
3         P = Monitor(self.inputfile, self.outputfile, topic)
4         df = P.ReadBagFileByTopic(self.inputfile, topic)
5         posx_df = P.DataFrameExtractByTerm(
6             df, 'pose.pose.position.x', None)
7         posy_df = P.DataFrameExtractByTerm(
8             df, 'pose.pose.position.y', None)
9         twistx_df = P.DataFrameExtractByTerm(
10            df, 'twist.twist.linear.x', None)
11        twisty_df = P.DataFrameExtractByTerm(
12            df, 'twist.twist.linear.y', None)
13        Ego_df = pd.DataFrame(posx_df.values, columns=['Ego.x'])
14        Ego_df['Ego.y'] = posy_df.values
15        Ego_df['Ego.Vx'] = twistx_df.values
16        Ego_df['Ego.Vy'] = twisty_df.values
17        Kv = P.GenerateKrypkeFromDataFrame(twistx_df, EGO_SPEED,
18                                         v)
19        if Kv is False:
20            print(
21                'cannot check model for speed, either dataframe
22    is empty or no relevant data found')
23        if Kv is not False:
24            taw = P.GetFormula(EGO_SPEED, v)
25            print('The Speed Limit Formula is: ', taw)
26            P.CheckModel(Kv, taw)
27            self.results.append(P.Result)

```

Listing 7.6: class Analyser: Topic 3

```

1     elif topic == '/vehicle'+str(i)+'/traffic':
2         print('Analysing fourth topic:', topic)
3         T = Monitor(self.inputfile, self.outputfile, topic)
4         print('extracting data from bag file..')
5         dt = T.ReadBagFileByTopic(self.inputfile, topic)
6         dx = T.DataFrameExtractByTerm(dt, 'data', None)
7         print('converting json data to dataframe..')
8         dataset = T.ConvertStringListToDataFrame(dx)
9         if dataset.empty:
10             print('dataframe is empty, skipping..')
11             break
12         traffic_data = T.FilterAndJoinPOVs(dataset)
13         print('Creating final full dataset..')
14         full_set = pd.concat(
15             [Ego_df, ax_df, traffic_data], axis=1, join="outer")
16         Sx = T.GenerateKrypkeFromDataFrame(

```

```

17             full_set, SAFE_DISTANCE_X, x)
18         if Sx is False:
19             print(
20                 'cannot check model for x axis safe distance,
21                 either dataframe is empty or no relevant data found')
22             if Kv is not False:
23                 omega = T.GetFormula(SAFE_DISTANCE_X, x)
24                 print('The Longitudinal Safe Distance Formula is: ',
25                     omega)
26
27                 T.CheckModel(Kv, omega)
28                 self.results.append(T.Result)
29
30
31             Sy = T.GenerateKripkeFromDataFrame(
32                 full_set, SAFE_DISTANCE_Y, y)
33             if Sy is False:
34                 print(
35                     'cannot check model for y axis safe distance,
36                     either dataframe is empty or no relevant data found')
37                 if Sy is not False:
38                     gamma = T.GetFormula(SAFE_DISTANCE_Y, y)
39                     print('The Lateral Safe Distance Formula is: ', gamma
40             )
41
42                 T.CheckModel(Sy, gamma)
43                 self.results.append(T.Result)
44
45             Ka = T.GenerateKripkeFromDataFrame(
46                 full_set, Deceleration, a)
47             if Ka is False:
48                 print(
49                     'cannot check model for brake deceleration,
50                     either dataframe is empty or no relevant data found')
51                 if Ka is not False:
52                     Lambda = T.GetFormula(Deceleration, a)
53                     print('The Lateral Safe Distance Formula is: ',
54                         Lambda)
55
56                 T.CheckModel(Ka, Lambda)
57                 self.results.append(T.Result)
58
59 print('checking overall test Result..')
60 if all(self.results):
61     Monitor.WriteResultToFile(self.outputfile, True)
62 else:
63     Monitor.WriteResultToFile(self.outputfile, False)

```

Listing 7.7: class Analyser: Topic 4

```

1 Parsing Arguments ..
2 Analysing first topic: /vehicle0/ENV/propositions
3 [INFO] Data folder ../Bagfiles/test_ci_1_old already exists. Not creating.
4 extracting topic from bag file
5 Checking if there is a Collision..
6 Kripke Structure created.
7 Checking if Goal is reached..
8 Kripke Structure created.
9 The No Collision Formula is: A(G(not p))

```

```

10 The Kripke Structure satisfies the defined formula, saving test result
11 The Goal Achieving Formula is: A(E(G(F(q))))
12 The Kripke Structure satisfies the defined formula, saving test result
13 Analysing second topic: /vehicle0/VEH/ax
14 [INFO] Data folder ../Bagfiles/test_ci_1_old already exists. Not creating.
15 Analysing third topic: /vehicle0/odom
16 [INFO] Data folder ../Bagfiles/test_ci_1_old already exists. Not creating.
17 Checking Speed limit..
18 Kripke Structure created.
19 The Speed Limit Formula is: A(G(v))
20 The Kripke Structure satisfies the defined formula, saving test result
21 Analysing fourth topic: /vehicle0/traffic
22 extracting data from bag file..
23 [INFO] Data folder ../Bagfiles/test_ci_1_old already exists. Not creating.
24 converting json data to dataframe..
25 dataframe is empty, skipping..
26 checking overall test Result..
27 test passed, Result saved to: /home/sd/outputdir/scripts/outputfile.txt

```

Listing 7.8: Resultat1: Auswertung einer Bag-Datei

```

1 Parsing Arguments ..
2 Analysing first topic: /vehicle0/ENV/propositions
3 [INFO] Data folder ../Bagfiles/test_ci already exists. Not creating.
4 extracting topic from bag file
5 Checking if there is a Collision..
6 Kripke Structure created.
7 Checking if Goal is reached..
8 Kripke Structure created.
9 The No Collision Formula is: A(G(not p))
10 The Kripke Structure doesn't satisfy the defined formula, saving test result
11 The Goal Achieving Formula is: A(E(G(F(q))))
12 The Kripke Structure doesn't satisfy the defined formula, saving test result
13 Analysing second topic: /vehicle0/VEH/ax
14 [INFO] Data folder ../Bagfiles/test_ci already exists. Not creating.
15 Analysing third topic: /vehicle0/odom
16 [INFO] Data folder ../Bagfiles/test_ci already exists. Not creating.
17 Checking Speed limit..
18 Kripke Structure created.
19 The Speed Limit Formula is: A(G(v))
20 The Kripke Structure doesn't satisfy the defined formula, saving test result
21 Analysing fourth topic: /vehicle0/traffic
22 extracting data from bag file..
23 [INFO] Data folder ../Bagfiles/test_ci already exists. Not creating.
24 converting json data to dataframe..
25 Creating final full dataset..
26 Checking longitudinal safe distance..
27 Kripke Structure created.
28 The Longitudinal Safe Distance Formula is: A(G(x))
29 The Kripke Structure doesn't satisfy the defined formula, saving test result
30 Checking lateral safe distance..
31 Kripke Structure created.
32 The Lateral Safe Distance Formula is: A(G(y))
33 The Kripke Structure doesn't satisfy the defined formula, saving test result

```

```

34 Checking Deceleration if brake light is on..
35 no data found, nothing to create..
36 cannot check model for brake deceleration, either dataframe is empty or no
    relevant data found
37 checking overall test Result..
38 test failed, Result saved to: /home/sd/outputdir/scripts/outputfile.txt

```

Listing 7.9: Resultat2: Auswertung einer Bag-Datei

```

1 import sys
2 import argparse
3 import csv
4 import os
5 import bagpy
6 from bagpy import bagreader
7 import pandas as pd
8 import numpy as np
9 from collections import defaultdict, Iterable
10 from pyModelChecking import *
11 from pyModelChecking.CTLS import *
12
13
14 # ModelChecker.py
15 description = 'ModelChecker.py |\n \
16 Read bag file, extract data by topic |\n \
17 Create a krypke Structure for model |\n \
18 Model checking with CTL* |\n \
19 topics: i >=0 [/vehicle{i}/ENV/propositions, /vehicle{i}/VEH/ax, /vehicle{i}/
    odom, /vehicle{i}/traffic]|\n \
20 shell: python3 ModelChecker.py -i|--input inputfile.bag -o|--output
    outputfile.txt -t|--topic |\n \
21 example: python3 ModelChecker.py -i inputfile.bag -o outputfile.txt -t /
    vehicle0/ENV/propositions /vehicle0/VEH/ax /vehicle0/odom /vehicle0/
    traffic|\n \
22 Author: Sami Dhiab |\n \
23 Topics to analyse: |\n \
24 topic1 = '/vehicle0/ENV/propositions' -> Criteria {in collision, near goal}
    |\n \
25 topic2 = '/vehicle0/VEH/ax' -> Criteria {Ego acceleration, deceleration} |\n \
    \
26 topic3 = '/vehicle0/odom' -> Criteria {Ego position(x,y), Ego Speed(x,y),
    ...} |\n \
27 topic4 = '/vehicle0/traffic' -> Criteria {sumo ID, sumo position, sumo Speed
    , signal status(brake, left, right indicator) ...} |\n \
28 Kriteria to check: |\n \
29 -twist: Ego Speed |\n \
30 -ax: Ego Acceleration, Deceleration Rate |\n \
31 -safety Separation Distance(gap) Lateral/longitudinal |\n \
32 -Signal Status (brake is on, left or right indicator)'
33
34 IN_COLLISION = 'IN_COLLISION'
35 NEAR_GOAL = 'NEAR_GOAL'
36 EGO_SPEED = 'EGO_SPEED'
37 MAX_SPEED = 30

```

```

38 SAFE_DISTANCE_X = 'SAFE_DISTANCE_X'
39 SAFE_DISTANCE_Y = 'SAFE_DISTANCE_Y'
40 Deceleration = 'Deceleration'
41 p = 'p' # IN Collision
42 q = 'q' # Near goal
43 v = 'v' # EGO_SPEED
44 x = 'x' # X_Safe_dis
45 y = 'y' # Y_SAfe_dis
46 a = 'a' # EGO_ax
47
48
49 class Monitor:
50     """Class for monitoring a model\n
51     *Read bag file, extract data by topic\n
52     *Creating a krypke Structure for model\n
53     *Model checking with CTL* \n
54     *Args: see constructor: Monitor.__init__\n
55     """
56
57     def __init__(self, inputfile, outputfile, topic):
58         """Monitor constructor\n
59         Args:\n
60             inputfile (.bag): choose a bag file to treat\n
61             outputfile (.txt): choose a file to write result to\n
62             topic (string): choose a topic string to parse\n
63         """
64         self.inputfile = inputfile
65         self.topic = topic
66         self.outputfile = outputfile
67         self.Result = None
68
69     def ReadBagFileByTopic(self, inputfile, topic):
70         """Read bag file and filter by topic\n
71         write content to csv file\n
72         get the content in pandas dataframe"""
73
74         b = bagreader(inputfile)
75         data = b.message_by_topic(topic)
76         dataframe = pd.read_csv(data)
77
78         return dataframe
79
80     @staticmethod
81     def WriteResultToFile(outputfile, result):
82         """write test Result to a file"""
83         with open(outputfile, 'w') as the_file:
84             if result:
85                 print('test passed, Result saved to: ',
86                       os.path.realpath(outputfile))
87                 the_file.write('test passed\n')
88
89             else:
90                 print('test failed, Result saved to: ',

```

```

91             os.path.realpath(outputfile))
92             the_file.write('test failed\n')
93
94     def DataFrameExtractByTerm(self, BaseDf, Term, Value):
95         """Extract dataframe from dataframe
96         by specifying Term and Value
97         resetting index from zero"""
98
99     if Value is None:
100         ExtractedDf = BaseDf[[Term]]
101     else:
102         ExtractedDf = BaseDf.loc[BaseDf[Term] == Value]
103         # Reset indexing from zero
104         ExtractedDf = ExtractedDf.reset_index(drop=True)
105
106     return ExtractedDf
107
108 def GetColumnIndex(self, Df, ColumnName):
109     """return the Index of column in dataframe """
110     IndexColumn = Df.columns.get_loc(ColumnName)
111
112     return IndexColumn
113
114 def CheckDecelerationToBrakeStatus(self, brakeStatusOn,
115                                   acutalAcceleration, lastAcceleration):
116     """Check deceleration rate based on brake light status"""
117     if brakeStatusOn == True: # if brakelight is on
118         if (acutalAcceleration - lastAcceleration) < 0:
119             return True
120         else:
121             return False
122
123 def GenerateKrypkeFromDataFrame(self, DataFrame, PropositionString,
124                                 AtomicProposition):
125     """Iterate dataframe and update propositions
126     generate krypke from propositions"""
127     from distutils.util import strtobool
128     H = []
129     R = [(0, 0)]
130     L = dict()
131     Brake = None
132
133     if PropositionString == SAFE_DISTANCE_X:
134         print('Checking longitudinal safe distance..')
135     if PropositionString == SAFE_DISTANCE_Y:
136         print('Checking lateral safe distance..')
137     if PropositionString == Deceleration:
138         print('Checking Deceleration if brake light is on..')
139         b = self.GetColumnIndex(DataFrame, 'brakeLightStatusOn')
140         l = [i for i, x in enumerate(b) if x]
141         a = self.GetColumnIndex(DataFrame, 'Ego.ax')
142     if PropositionString == EGO_SPEED:
143         print('Checking Speed limit..')

```

```

142     if PropositionString == IN_COLLISION:
143         print('Checking if there is a Collision..')
144     if PropositionString == NEAR_GOAL:
145         print('Checking if Goal is reached..')
146
147     if DataFrame.dropna().empty:
148         print('empty dataframe, nothing to create..')
149         return False
150
151     for i, j in DataFrame.iterrows():
152         B = (i, i+1)
153         # AtomicProposition should be String like 'p', 'q'...
154         p = str(AtomicProposition)
155
156         if PropositionString == IN_COLLISION or PropositionString ==
157             NEAR_GOAL:
158             t = self.GetColumnIndex(DataFrame, 'term')
159             v = self.GetColumnIndex(DataFrame, 'value')
160             if str(j[t]) == PropositionString and j[v] == False: # incollision is false
161                 H = {i: set([Not(p)])}
162
163             elif str(j[t]) == PropositionString and j[v] == True: # incollision is true
164                 H = {i: set([p])}
165
166             elif PropositionString == EGO_SPEED:
167                 e = self.GetColumnIndex(DataFrame, 'twist.twist.linear.x')
168                 if int(j[e]) > MAX_SPEED: # ego_speed > MAx_speed -> p false
169                     H = {i: set([Not(p)])}
170
171                 elif int(j[e]) <= MAX_SPEED: # ego_speed =< MAx_speed -> p true
172                     H = {i: set([p])}
173
174             elif PropositionString == SAFE_DISTANCE_X:
175                 for index, value in enumerate(self.GetColumnIndex(DataFrame,
176                     'POV_X')):
177                     if value:
178                         if pd.isnull(j[self.GetColumnIndex(DataFrame, 'Ego.x'
179                         ]]) or pd.isnull(j[self.GetColumnIndex(DataFrame, 'Ego.Vx')]) or pd.
180                         isnull(j[index]):
181                             continue
182                         if self.CheckSafeDistance(
183                             j[self.GetColumnIndex(DataFrame, 'Ego.Vx')],
184                             j[self.GetColumnIndex(DataFrame, 'Ego.x')], j[index]):
185                                 H = {i: set([p])}
186                         else:
187                             H = {i: set([Not(p)])}
188
189             elif PropositionString == SAFE_DISTANCE_Y:
190                 for index, value in enumerate(self.GetColumnIndex(DataFrame,
191                     'POV_Y')):
192                     if value:
193                         if pd.isnull(j[self.GetColumnIndex(DataFrame, 'Ego.y'

```

```

        )]) or pd.isnull(j[index]):
                continue
                if self.CheckLateralDistance(j[self.GetColumnIndex(
                    DataFrame, 'Ego.y')], j[index]):
                    H = {i: set([p])}
                else:
                    H = {i: set([Not(p)])}

191
192     elif PropositionString == Deceleration:
193         if i == 0:
194             continue
195         for idx in l:
196             if pd.isnull(j[idx]):
197                 continue
198             if strtobool(j[idx]):
199                 Brake = True
200             else:
201                 Brake = False
202             if self.CheckDecelerationToBrakeStatus(Brake, j[a], DataFrame
203 .at[i-1, 'Ego.ax']):
204                 H = {i: set([p])}
205             elif self.CheckDecelerationToBrakeStatus(Brake, j[a],
206 DataFrame.at[i-1, 'Ego.ax']) is False:
207                 H = {i: set([Not(p)])}

208         if i == DataFrame.index[-1]:
209             B = (i, i)

210         R.append(B)
211         L.update(H)

212
213     if H == {}:
214         R = []
215         L = dict()
216         print('no data found, nothing to create..')
217         return False
218
219     K = Kripke(R=R, L=L)
220     print('Kripke Structure created.')
221     return K
222
223 def FullMergeSet(self, R1, R2):
224     """Merge content of different sets
225     using the larger set"""
226     if len(R1) > len(R2):
227         R = R1
228     else:
229         R = R2
230     return R
231
232 def FullMergeDict(self, L1, L2):
233     """Merge value of multiple dictionaries
234     using the same key to new dict"""

```

```

235     L = defaultdict(list)
236     for dct in [L1, L2]:
237         for k, v in dct.items():
238             if isinstance(v, Iterable):
239                 L[k].extend(v)
240             else:
241                 L[k].append(v)
242     return L
243
244 def MergeKrypke(self, K1, K2):
245     """Merge content of two Krypke Structures
246     by merging sets and dicts"""
247     R1 = K1.transitions()
248     R2 = K2.transitions()
249     L1 = K1.labelling_function()
250     L2 = K2.labelling_function()
251
252     K = Kripke(R=self.FullMergeSet(R1, R2), L=self.FullMergeDict(L1, L2))
253     return K
254
255 def GetFormula(self, PropositionSet, AtomicProposition):
256     """Get the right formula for a defined proposition"""
257     p = AtomicProposition
258     phi = A(G(Not(p))) # phi: for all paths always not true
259     # psi: for all paths exist in future a path and from this path always
260     # true
261     psi = A(E(G(F(p))))
262     taw = A(G(p)) # taw: true for all paths always
263     omega = A(G(p)) # omega: always true for all paths
264     gamma = A(G(p)) # gamma: always true for all paths
265     Lambda = A(G(p)) # lambda: always true for all paths
266     if PropositionSet == IN_COLLISION:
267         return phi
268     elif PropositionSet == NEAR_GOAL:
269         return psi
270     elif PropositionSet == EGO_SPEED:
271         return taw
272     elif PropositionSet == SAFE_DISTANCE_X:
273         return omega
274     elif PropositionSet == SAFE_DISTANCE_Y:
275         return gamma
276     elif PropositionSet == Deceleration:
277         return Lambda
278
279 def CheckModel(self, K, F):
280     """ Check the model of a krypke structure by a defined formula
281     return the states that satisfies that formula"""
282
283     m = modelcheck(K, F)
284     if len(m) == len(K.labelling_function()):
285         print(
286             "The Kripke Structure satisfies the defined formula, saving
287             test result")

```

```

286         self.Result = True
287
288     else:
289         print(
290             "The Kripke Structure doesn't satisfy the defined formula,
291             saving test result")
292         self.Result = False
293
294     return m
295
296 def CombineFormulas(self, Formula1, Formula2):
297     """Combine two Formulas with And to build a general Formula"""
298     Combined = And(Formula1, Formula2)
299     return Combined
300
301 def CheckSafeDistance(self, SpeedEgo, PosEgo, PosPov):
302     """Check Longitudinal Safe Distance between Ego and POVs
303     Can be upgraded regarding ODD"""
304     s = SpeedEgo
305     xEgo = PosEgo
306     xPov = PosPov
307     Delta = abs(xEgo-xPov)
308     # check if speed smaller than 15m/s or 50 km/h inside urban (for 50km
309     /h distance should be 15m)
310     if s <= 15:
311         SafeDistance = int((s*3.6) / 3.33)
312         if Delta >= SafeDistance:
313             # print('Ego Speed:', s, 'm/s, Position Ego:', xEgo, 'm,
314             Position POV:', xPov,
315             #      'm, Safe Distance kepted:', SafeDistance, 'm, actual
316             Distance:', Delta, 'm')
317             return True
318         elif Delta < SafeDistance:
319             # print('Ego Speed:', s, 'm/s, Position Ego:', xEgo, 'm,
320             Position POV:', xPov,
321             #      'm, Safe Distance required:', SafeDistance, 'm, actual
322             Distance:', Delta, 'm')
323             return False
324
325     # check if speed bigger than 15m/s or 50 km/h outside urban (for 100
326     km/h distance should be 50m)
327     else:
328         SafeDistance = int((s*3.6) / 2)
329         if Delta >= SafeDistance:
330             # print('Ego Speed:', s, 'm/s, Position Ego:', xEgo, 'm,
331             Position POV:', xPov,
332             #      'm, Safe Distance kepted:', SafeDistance, 'm, actual
333             Distance:', Delta, 'm')
334             return True
335         elif Delta < SafeDistance:
336             # print('Ego Speed:', s, 'm/s, Position Ego:', xEgo, 'm,
337             Position POV:', xPov,
338             #      'm, Safe Distance required:', SafeDistance, 'm, actual

```

```

329     Distance:', Delta, 'm')
330         return False
331
332     def CheckLateralDistance(self, YEgo, YPov):
333         """Check Lateral Safe Distance between Ego and POVs
334         Can be upgraded regarding ODD"""
335         Delta = abs(YEgo-YPov)
336         SafeLateral = 1.5 # depending on object classification(0,5->1,5)
337         if Delta >= SafeLateral:
338             # print('Safe Lateral Distance keeped: ', Delta, 'm')
339             return True
340         elif Delta < SafeLateral:
341             # print('Safe Distance required: ', SafeLateral,
342             #       'm, actual Distance: ', Delta, 'm')
343             return False
344
345     def ConvertStringListToDataframe(self, datafram):
346         """Convert list of strings like json format
347         inside a Dataframe to a sorted Dataframe"""
348         datas = []
349         for index, value in datafram.iterrows():
350             data_array = datafram.iloc[index, 0].split()
351             datas.extend(data_array)
352
353         Px = []
354         Py = []
355         ID = []
356         Vx = []
357         BrakeStatus = []
358         for i, data in enumerate(datas):
359             if data == 'position:':
360                 x = float(datas[i+2])
361                 Px.append(x)
362                 y = float(datas[i+4])
363                 Py.append(y)
364             if data == 'linear:':
365                 v = float(datas[i+2])
366                 Vx.append(v)
367             if data == 'brakeLightOn:':
368                 Brake = datas[i+1]
369                 BrakeStatus.append(Brake)
370             if data == 'trackingID:':
371                 t = int(datas[i+1])
372                 ID.append(t)
373
374         df = pd.DataFrame(ID, columns=['POV_ID'])
375         df['POV_X'] = Px
376         df['POV_Y'] = Py
377         df['POV_Vx'] = Vx
378         df['brakeLightStatusOn'] = BrakeStatus
379         return df
380
381     def FilterAndJoinPOVs(self, dataset):
382         """ extract POVs dataframe filtered
383             by ID, join dataframes together for next step"""

```

```

381     max_col = (dataset["POV_ID"]).max()
382     ID_max = max_col % 1000
383     #Columns = ['POV_ID_'+str(x) for x in range(ID_max)]
384     list_df = []
385     for i in range(ID_max):
386         extracted_df = self.DataFrameExtractByTerm(
387             dataset, 'POV_ID', 1000+i)
388         list_df.append(extracted_df)
389     joined_df = pd.concat(list_df, axis=1, join="outer")
390     return joined_df
391
392
393 class Analyser:
394
395     """Class for Parsing Arguments
396     and analysing all topics from command line"""
397
398     def __init__(self):
399         self.inputfile = None
400         self.outputfile = None
401         self.topics = None
402         self.results = []
403
404     def ParseArgument(self):
405         """Arguments parsing and storing in class variables"""
406
407         parser = argparse.ArgumentParser(description='description')
408         parser.add_argument('-i', '--input', type=str,
409                             help='select a bag file')
410         parser.add_argument('-o', '--output', type=str,
411                             help='write result to a file')
412         parser.add_argument('-t', '--topics', nargs='+',
413                             type=str, help='select topics from list, see help
for more infos')
414
415         args = parser.parse_args()
416         print('Parsing Arguments ..')
417         self.inputfile = args.input
418         self.outputfile = args.output
419         self.topics = args.topics
420
421     def AnalyseTopics(self):
422         """ iterarte throw al topics from the list"""
423         for i in range(50):
424             for topic in self.topics:
425                 if topic == '/vehicle'+str(i)+'/ENV/propositions':
426                     print('Analysing first topic:', topic)
427                     M = Monitor(self.inputfile, self.outputfile, topic)
428                     df = M.ReadBagFileByTopic(self.inputfile, topic)
429                     print('extracting topic from bag file')
430                     df1 = M.DataFrameExtractByTerm(df, 'term', IN_COLLISION)
431                     df2 = M.DataFrameExtractByTerm(df, 'term', NEAR_GOAL)
432                     K1 = M.GenerateKrypkeFromDataFrame(df1, IN_COLLISION, p)

```

```

433         K2 = M.GenerateKrypkeFromDataFrame(df2, NEAR_GOAL, q)
434         if K1 is False:
435             print(
436                 'cannot check model for collision, either
437                 dataframe is empty or no relevant data found')
438             if K2 is False:
439                 print(
440                     'cannot check model for goal, either dataframe is
441                     empty or no relevant data found')
442                     if K1 is not False:
443                         phi = M.GetFormula(IN_COLLISION, p)
444                         print('The No Collision Formula is: ', phi)
445                         M.CheckModel(K1, phi)
446                         self.results.append(M.Result)
447                     if K2 is not False:
448                         psi = M.GetFormula(NEAR_GOAL, q)
449                         print('The Goal Achieving Formula is: ', psi)
450                         M.CheckModel(K2, psi)
451                         self.results.append(M.Result)

452             elif topic == '/vehicle'+str(i)+'/VEH/ax':
453                 print('Analysing second topic:', topic)
454                 N = Monitor(self.inputfile, self.outputfile, topic)
455                 df = N.ReadBagFileByTopic(self.inputfile, topic)
456                 ax_df = (N.DataFrameExtractByTerm(df, 'data', None)
457                             .rename(columns={"data": "Ego.ax"}))

458             elif topic == '/vehicle'+str(i)+'/odom':
459                 print('Analysing third topic:', topic)
460                 P = Monitor(self.inputfile, self.outputfile, topic)
461                 df = P.ReadBagFileByTopic(self.inputfile, topic)
462                 posx_df = P.DataFrameExtractByTerm(
463                     df, 'pose.pose.position.x', None)
464                 posy_df = P.DataFrameExtractByTerm(
465                     df, 'pose.pose.position.y', None)
466                 twistx_df = P.DataFrameExtractByTerm(
467                     df, 'twist.twist.linear.x', None)
468                 twisty_df = P.DataFrameExtractByTerm(
469                     df, 'twist.twist.linear.y', None)
470                 Ego_df = pd.DataFrame(posx_df.values, columns=['Ego.x'])
471                 Ego_df['Ego.y'] = posy_df.values
472                 Ego_df['Ego.Vx'] = twistx_df.values
473                 Ego_df['Ego.Vy'] = twisty_df.values
474                 Kv = P.GenerateKrypkeFromDataFrame(twistx_df, EGO_SPEED,
475                                         v)
476                     if Kv is False:
477                         print(
478                             'cannot check model for speed, either dataframe
479                             is empty or no relevant data found')
480                             if Kv is not False:
481                                 taw = P.GetFormula(EGO_SPEED, v)
482                                 print('The Speed Limit Formula is: ', taw)
483                                 P.CheckModel(Kv, taw)

```

```

482             self.results.append(P.Result)
483
484     elif topic == '/vehicle'+str(i)+'/traffic':
485         print('Analysing fourth topic:', topic)
486         T = Monitor(self.inputfile, self.outputfile, topic)
487         print('extracting data from bag file..')
488         dt = T.ReadBagFileByTopic(self.inputfile, topic)
489         dx = T.DataFrameExtractByTerm(dt, 'data', None)
490         print('converting json data to dataframe..')
491         dataset = T.ConvertStringListToDataframe(dx)
492         if dataset.empty:
493             print('dataframe is empty, skipping..')
494             break
495         traffic_data = T.FilterAndJoinPOVs(dataset)
496         print('Creating final full dataset..')
497         full_set = pd.concat(
498             [Ego_df, ax_df, traffic_data], axis=1, join="outer")
499         Sx = T.GenerateKrypkeFromDataFrame(
500             full_set, SAFE_DISTANCE_X, x)
501         if Sx is False:
502             print(
503                 'cannot check model for x axis safe distance,
504                 either dataframe is empty or no relevant data found')
505             if Kv is not False:
506                 omega = T.GetFormula(SAFE_DISTANCE_X, x)
507                 print('The Longitudinal Safe Distance Formula is: ',
508                      omega)
509
510                 T.CheckModel(Kv, omega)
511                 self.results.append(T.Result)
512
513         Sy = T.GenerateKrypkeFromDataFrame(
514             full_set, SAFE_DISTANCE_Y, y)
515         if Sy is False:
516             print(
517                 'cannot check model for y axis safe distance,
518                 either dataframe is empty or no relevant data found')
519             if Sy is not False:
520                 gamma = T.GetFormula(SAFE_DISTANCE_Y, y)
521                 print('The Lateral Safe Distance Formula is: ', gamma)
522
523                 T.CheckModel(Sy, gamma)
524                 self.results.append(T.Result)
525
526         Ka = T.GenerateKrypkeFromDataFrame(
527             full_set, Deceleration, a)
528         if Ka is False:
529             print(
530                 'cannot check model for brake deceleration,
531                 either dataframe is empty or no relevant data found')
532             if Ka is not False:
533                 Lambda = T.GetFormula(Deceleration, a)
534                 print('The Lateral Safe Distance Formula is: ',
535                      Lambda)

```

```
529             T.CheckModel(Ka, Lambda)
530             self.results.append(T.Result)
531
532         print('checking overall test Result..')
533         if all(self.results):
534             Monitor.WriteResultToFile(self.outputfile, True)
535         else:
536             Monitor.WriteResultToFile(self.outputfile, False)
```

Listing 7.10: Gesamtes Skript

## Eidesstattliche Erklärung

Ich versichere, dass ich diese Abschlussarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort, Datum

Unterschrift

berlin, den 6. April 2021

A handwritten signature in black ink, appearing to be a stylized 'W' or a similar character.