TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Elektrotechnik und Informationstechnik

Praktikumsbericht

Entwicklung und Implementierung einer automatisierten szenariobasierten Unit-Test Strategie für einen modellprädiktiven Pfadfolgeregler in einer GitLab CI Pipeline

vorgelegt von

Georg Ehrler

Matrikel-Nr.: 521446

Studiengang: Energie- und Automatisierungssysteme

Betreuer: Francisco Moreno, M.Sc.
Robert Ritschel, M.Sc. (IAV)

Prüfer: Prof. Dr.-Ing. habil. Stefan Streif

Datum: 14. März 2024

IAV GmbH

Regelungstechnik und Systemdynamik
Prof. Dr.-Ing. habil. Stefan Streif

Inhaltsverzeichnis

Abbildungsverzeichnis IV

1 Einleitung 1

2 Grundlagen 3
2.1 Modellprädiktive Pfadfolgeregelung . . . 3
2.1.1 Modellprädiktive Regelung . . . 3
2.1.2 Pfadfolgeregelung . . . 4
2.2 Szenariobasiertes Testen im Automobilbereich . . . 5
2.3 Softwartests und CI/CD . . . 6

3 Implementierung 7
3.1 Anforderungen . . . 7
3.2 Simulationsumgebung . . . 7
3.3 MATLAB® Unit Test Framework . . . 7
3.4 Aufbau und Ablauf eines Szenarios . . . 9
3.4.1 Simulationsparameter . . . 9
3.4.2 Ablauf eines Testskripts . . . 10
3.4.3 Parametergenerierung . . . 11
3.4.4 Definition von Key Performance Indicators und Testkriterien . . . 12
3.5 Integration in die GitLab CI Pipeline . . . 15
3.5.1 Darstellung der Pipeline . . . 15
3.5.2 Darstellung der Testergebnisse . . . 16

4 Vorstellung ausgewählter Szenarien 19
4.1 Szenario: Gerade mit initialem Versatz . . . 19
4.1.1 Parametrierung . . . 19
4.1.2 Abgetestete KPIs . . . 20
4.2 Szenario: Kurve negativ . . . 21
4.2.1 Parametrierung . . . 22
4.2.2 abgetestete KPIs . . . 22
4.3 Szenario: ACC mit konstanter Objektgeschwindigkeit . . . 23
4.3.1 Parametrierung . . . 23
4.3.2 Abgetestete KPIs . . . 24

5 Zusammenfassung und Ausblick 27

Literatur 29

Abbildungsverzeichnis

Abbildung 1.1 Bei Verkehrsunfällen Getötete pro Jahr in Tausend [4] 1

Abbildung 2.1 Ablauf einer modellprädiktiven Regelung [1] 3
Abbildung 2.2 schematischer Aufbau der MPFC [17] 4
Abbildung 2.3 Einteilung des Informationsgehalts eines Szenarios nach Schichten [2] 5
Abbildung 2.4 Erhöhung der Anzahl an Szenarios mit zunehmener Konkretisierung der Szenarien [12] 6
Abbildung 2.5 Ablauf eines Testprozesses [23] 6
Abbildung 2.6 Ablauf eines CI/CD Prozesses [8] 6

Abbildung 3.1 Beispiel einer komplexen Simulationsstrecke 8
Abbildung 3.2 Ablauf von klassenbasierten Tests 9
Abbildung 3.3 schematischer Ablauf eines Testskripts zur Ausführung eines Szenarios 11
Abbildung 3.4 Vergleich der Verteilung von zufälligen Sampling mit Latin Hypercube Sampling [16] 12
Abbildung 3.5 laterale Beschleunigung in Kurven eines typischen Fahrers [18] 14
Abbildung 3.6 maximal zulässige longitudinale Beschleunigung, Durchschnitt über 2 s [18] 14
Abbildung 3.7 maximal zulässiger negativer Ruck, Durchschnitt über 1 s [18] 15
Abbildung 3.8 Darstelllung der Jobs in der Gitlab CI Pipeline im Browser. “Alice“ und “IAVShuttle“ sind Codenamen von Prototypenfahrezeugen 16
Abbildung 3.9 Übersichtliche Darstellung der Testergebnisse eines Szenarios auf der ersten Seite des Testberichts 17
Abbildung 3.10 Darstellung eines fehlgeschlagenen Tests im Testbericht. Der Wert der KPI und der tatsächliche Wert werden verglichen. Ein kurzer Text gibt dem Entwickler wieder, was fehlgeschlagen ist. Wenn sinnvoll wird ein Plot mit den relevanten Daten erzeugt. Die Angabe der Simulationsparameter genutzt werden um diese eine Simulation lokal zu wiederholen. . 18

Abbildung 4.1 Darstellung des Szenarios: Gerade mit initialem Versatz 20
Abbildung 4.2 KPI Pfadabweichung abhängig der Simulationsparameter . . 20
Abbildung 4.3 KPI Geschwindigkeitsabweichung abhängig der Simulationsparameter 21
Abbildung 4.4 KPI laterale Beschleunigung abhängig der Simulationsparameter 21
Abbildung 4.5 KPI lateraler Ruck abhängig der Simulationsparameter 22

Abbildung 4.6 Darstellung des Szenarios: Kurvenfahrt negativ 22
Abbildung 4.7 KPI Pfadabweichung bei verschiedenen Kurvenradien 23
Abbildung 4.8 Darstellung des Szenarios: ACC mit konstanter Objektgeschwindigkeit 23
Abbildung 4.9 Simulationsdaten für Abstands- und Distanzfehler bei unterschiedlichen Parametrierungen 24
Abbildung 4.10 Simulationsdaten für longitudinale Beschleunigung und Ruck bei unterschiedlichen Parametrierungen 25

1 Einleitung

Gestzliche Bestimmungen zur Erhöhung der Sicherheit von sowohl den Insassen eines Fahrzeugs als auch anderer Verkehrsteilnehmer haben in den letzten Jahrzehnten zu einer signifikanten Senkung der Mortalitätsrat bei Verkehrsunfällen beigetragen.

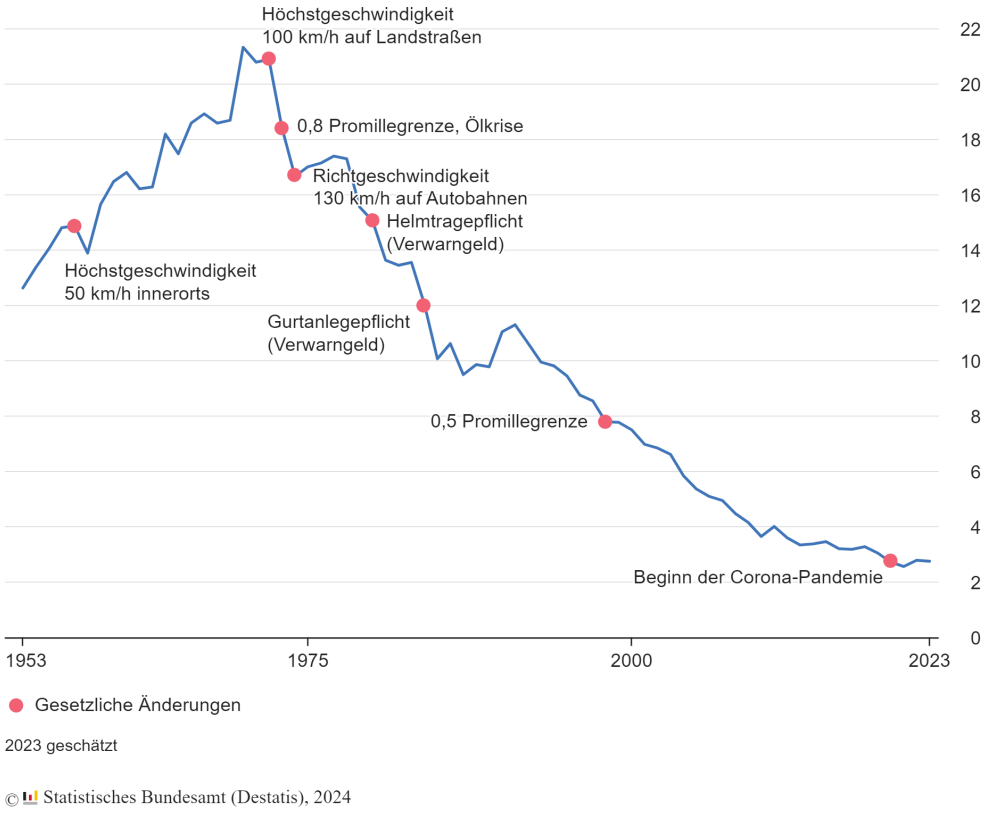


Abbildung 1.1: Bei Verkehrsunfällen Getötete pro Jahr in Tausend [4]

Im Jahr 2021 gab es über 250.000 Unfälle mit Personenschaden auf deutschen Straßen. In 88% der Fälle ist die Ursache auf ein Fehlverhalten der Fahrzeugführer zurückzuführen [4]. Der Einsatz von automatisierten Fahrzeugen kann zu einem weiteren Rückgang der in Abbildung 1.1 gezeigten Statistik führen, indem es den Fahrer durch Einsatz von intelligenten Asstenzsystemen unterstützt oder die menschliche Fehlerkomponente gänzlich beseitigt, indem ein Fahrzeug vollständig autonom fährt.

Um die Sicherheit der Fahrzeuge zu gewährleisten muss sowhl das Gesamtsystem als auch alle Teilsysteme intensiv getestet werden. Eines dieser Teilsystem für das hochautomatisierte Fahren ist die Regelung der Längs- und Querführung des Fahrzeugs. Bei der IAV[1] geschieht dies durch einen modellprädiktiven Pfadfolgeregler.

1 https://www.iav.com/

Aufgabenstellung

Im Rahmen des Praktikums soll eine automatisierte Teststrategie für einen modellprädiktiven Pfadfolgeregler (MPFC) konzeptioniert und implementiert werden.

Modellprädiktive Regler (MPC) eignen sich hervorragend für die Vorhersage und Regelung dynamischer System, weshalb sie in vielen Industriezweigen, einschließlich der Automobilindustrie, Anwendung finden. Im Bereich des hochautomatisierten Fahrens können mithilfe von MPC in Echtzeit Fahrentscheidungen, die sowohl sicherheits- als auch komfortrelevante Anforderungen erfüllen, getroffen werden. Um die Sicherheit und Zuverlässigkeit des Systems sicherzustellen sollen realitätsnahe Fahrszenarien ermittelt und anhand von festgelegten Key Performance Indicators (KPIs) bewertet werden. Dieser Prozess soll automatisiert in einer CI Pipeline ablaufen.

Es sind die folgenden Teilaufgaben umzusetzen:

- Konzeptentwicklung für das automatisierte, szenariobasierte Testen der MPFC unter Verwendung der bestehenden Simulationsumgebung
- Definition und Parametrierung geeigneter Fahrszenarien
- Definition von KPIs zur Beurteilung der Leistung der Regelung
- Einbindung in eine Gitlab CI Pipeline
- Dokumentation der Ergebnisse

2 Grundlagen

In diesem Kapitel wird zunächst eine Einführung in alle Themenbereiche gegeben. Insbesondere wird der modellprädiktive Pfadfolgeregler näher erläutert, um ein besseres Systemverständnis zu erlangen. Außerdem wird auf Strategien für das Überprüfen von Software und Testabläufe, speziell in der Automobilindustrie, näher eingegangen und ein Einblick in aktuelle Softwarentwicklungsabläufe gegeben.

2.1 Modellprädiktive Pfadfolgeregelung

2.1.1 Modellprädiktive Regelung

Die modellprädiktive Regelung (engl. Model Predictive Control, MPC) ist ein Verfahren zur Regelung von dynamischen Systemen. Dabei wird ein mathematisches Modell des Systems erstellt und zur Vorhersage der zukünftigen Systemzustände verwendet. Das prädizierte Systemverhalten wird dann verwendet, um ein Optimalsteuerungsproblem (engl. Optimal Control Problem - OCP) zu lösen und die optimale Eingabe zu finden um das System in einen gewünschten Zustand zu bringen. Der Ablauf einer modellprädiktiven Regelung, wie in Abbildung 2.1 zu sehen, kann in drei Punkte unterteilt werden [5]:

1. Mithilfe eines Modells wird der Ausgang eines Systems für $k + n_p$ Zeitschritte (Prädiktionshorizont) vorausgesagt. Die Prädiktion basiert sowohl auf den vergangenen Ein- und Ausgängen als auch auf den zukünftigen Eingaben über die Länge des Stellhorizonts n_c

2. Die Stellgrößensequenz u wird berechnet. Dabei wird eine Kostenfunktion minimiert, welche so gewählt wird, dass sich ein gewünschtes Systemverhalten ein-

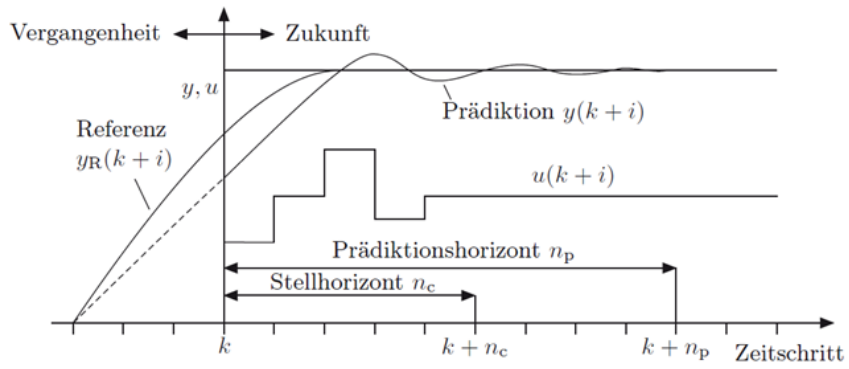


Abbildung 2.1: Ablauf einer modellprädiktiven Regelung [1]

stellt. Der Stellaufwand wird in der Regel in der Kostenfunktion mit berücksichtigt. Ebenfalls können Beschränkungen, von z.B. den Eingangsgrößen, berücksichtigt werden. Für jeden Zeitschritt wird so das OCP gelöst.

3. Das erste Element des Stellhorizonts wird auf das System angewendet. Stell- und Prädiktionshorizont verschieben sich um einen Zeitschritt in die Zukunft. Der Zyklus beginnt von vorn.

2.1.2 Pfadfolgeregelung

Die Implementierung der Pfadfolgeregelung für die in diesem Praktikum eine Teststrategie implementiert werden soll, ist in Abbildung 2.2 dargestellt. Als Eingänge dienen der MPFC eine Sollgeschwindigkeit, Pfaddaten, Fahrzeugzustände, vorgegebene Beschränkungen und ein Fahrzeugmodell. Daraus werden eine Sollbeschleunigung und eine Sollenkradwinkelgeschwindigkeit berechnet.

Die Grundidee der Pfadfolgeregelung ist in [6] beschrieben: Der Regler soll einem vorgegebenem Pfad möglichst gut folgen. Die Geschwindigkeit entlang des Pfades wird, im Gegensatz zu trajektorienbasierten Ansätzen, bei denen die Geschwindigkeit an jedem Punkt des Pfades vorgegeben ist und eingeregelt werden soll, vom Regler selbst festgelegt. Durch die Implementierung wird ein konvergieren des Pfades auf den Sollpfad sichergestellt [17].

Eine Methode zur Ermittlung geeigneter Parameter für das Optimierungsproblem der Pfadfolgeregelung ist in [19] dokumentiert. Darin steht die Erhöhung der Sicherheit und des Fahrkomfort im Fokus. Durch eine bayessche Optimierung werden Parameterwerte, welche eine gutes Verfolgen der Pfadgeschwindigkeit, geringe Abweichung vom Pfad und möglichst geringe laterale sowie longitudinale Beschleunigungen erzeugen, gefunden. Letzteres ist ein großer Faktor wenn es um Fahrkomfort geht [3].

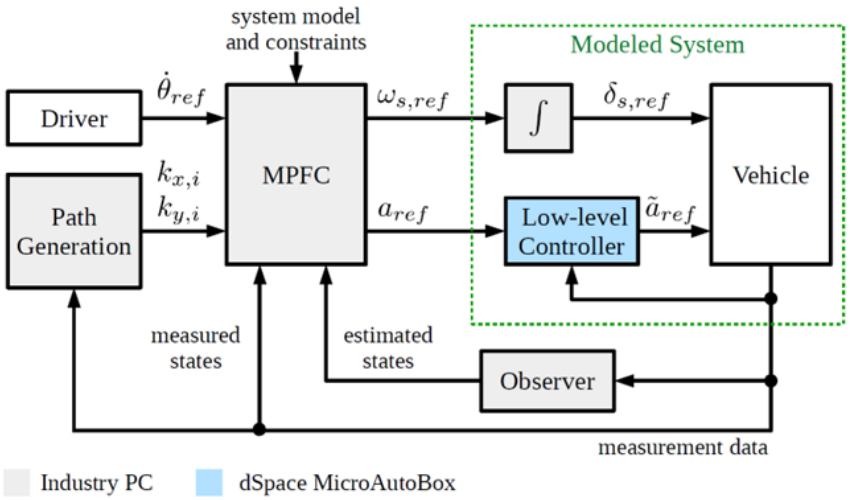


Abbildung 2.2: schematischer Aufbau der MPFC [17]

2.2 Szenariobasiertes Testen im Automobilbereich

Um ein autonomes Fahrsystem zu realisieren und Sicherheit zu gewährleisten sind ausgiebige Test- und Verifikationsprozesse erforderlich. Dabei ist es nicht mehr ausreichend, Testdaten durch reale Testfahrten auf der Straße zu sammeln. Ein großer Anteil der Daten ist schlichtweg uninteressant, da keine kritischen Fahrsituation auftreten. Es bietet sich daher an, in einer Datenbank die kritischen Szenarien zu erfassen und für spätere Verifikationsprozesse wieder heranzuziehen [14].

Um Szenarien zu beschreiben und in eine Datenbank einordnen zu können müssen diese klassifiziert werden. Eine Möglichkeit dafür ist die Einteilung nach Informationslevel [14][2].

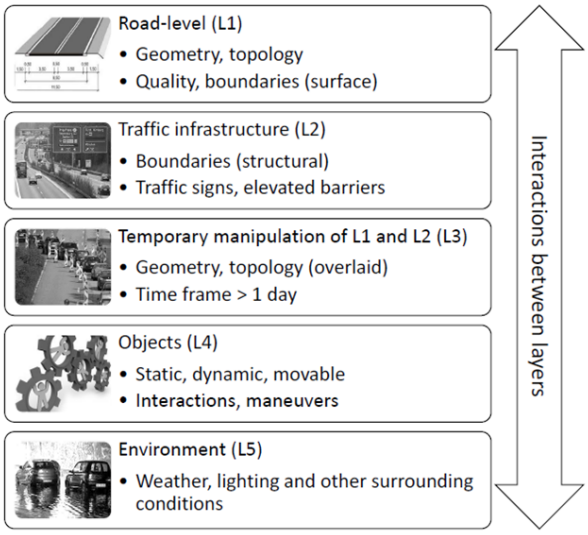


Abbildung 2.3: Einteilung des Informationsgehalts eines Szenarios nach Schichten [2]

Die erste Ebene definiert dabei grundlegende Eigenschaften der Straße, wie etwa deren Beschaffenheit oder Grenzen. Die nächste Ebene fügt Infrastrukturobjekte wie Straßenschilder hinzu. Ebene drei enthält Informationen über temporäre Veränderungen der darunterliegenden Ebenen. Ebene vier fügt einem Szenario statische und dynamische Objekte wie andere Verkehrsteilnehmer und Manöverplanung hinzu. Zum Schluss können noch Umweltfaktoren eingefügt werden [2].

Eine weitere Klassifizierungsmöglichkeit ist nach dem Detailgrad, wie es in Abbildung 2.4 [14][12]. Zunächst werden funktionale Szenarios beschrieben. Diese können einfach mit Worten beschrieben werden und enthalten noch keine Information über die Parameter. In einem ersten Abstrahierungsschritt werden diese zu logischen Szenarios. Hier wurden Parameter identifiziert und Grenzen für diese festgelegt. Um ein konkretes Szenario zu erhalten muss für jeden Parameter ein Wert ausgewählt werden. Theoretisch ermöglicht dies eine unendliche Anzahl an konkreten Szenarios, welche aus lediglich einem funktionalen Szenario erstellt werden können. Selbst bei rein simulativen Tests ist das nicht umsetzbar und muss reduziert werden [12].

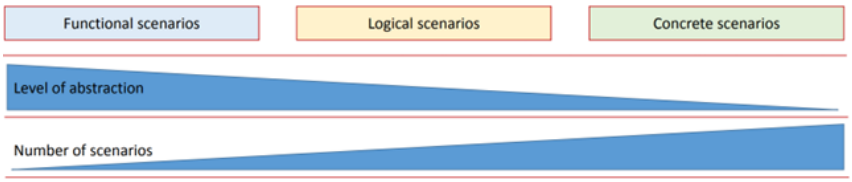


Abbildung 2.4: Erhöhung der Anzahl an Szenarios mit zunehmener Konkretisierung der Szenarien [12]

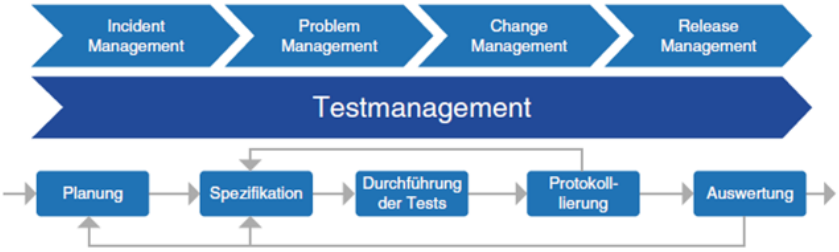


Abbildung 2.5: Ablauf eines Testprozesses [23]

2.3 Softwartests und CI/CD

Ein geeignetes Testmanagement erhöht die Qualität einer Software maßgeblich. Ein Testprozess läuft in der Regel nach dem in Abbildung 2.5 dargestellten Schema ab. Für den Testprozess exitisieren die Normen IEEE 829 und ISO-Standard 9126. Das International Software Testing Qualifications Board (ISTQB) arbeitet auf Grundlage dieser Normen an der Standardisierung von Systemtests [23].

Eine Möglichkeit Testprozesse Durchzuführen steckt hinter den Schlagworten Continuous Integration (CI) und Continuous Deployment (CD). Darunter versteht sich ein Prozess zum entwickeln, testen und freigeben von Software unter Nutzung von Versionsverwaltungssoftware wie er in Abbildung 2.6 dargestellt ist. Wenn ein Entwickler in einem Softwaremodul Änderungen veranlasst, läuft ein automatischer Testprozess ab. Dieser ist typischerweise in Stages unterteilt, in Abbildung 2.6 sind diese BUILD, TEST, MERGE. Werden darin keine Fehler in der Software gefunden werden die Änderungen eines Moduls zusammen mit den Änderungen an allen Modulen zusammengefasst und und somit die gesamte Software auf den aktuellen Stand gebracht (Continuous Delivery). Werden für die Gesamtsoftware ebenfalls keine Fehler gefunden kann eine neue Softwareversion zum Kunden gebracht werden (Continuous Deployment). Dieser Testprozess läuft in der Regel automatisch ab und veringert die Anzahl an Fehlern in Software [8].

Abbildung 2.6: Ablauf eines CI/CD Prozesses [8]

3 Implementierung

In diesem Kapitel werden die implementierten Funktionen und Klassen dokumentiert. Dabei wird zunächst der Aufbau und Ablauf von klassenbasierten Unit-Tests aus dem MATLAB® Unit Test Framework beschrieben und wie sich damit ein Konzept für das szenariobasierte Testen realisieren lässt. Weiterhin werden die implementierten Szenarien sowie deren Parametrierung und Ergebnisbewertung durch KPIs vorgestellt und auf ausgewählte Szenarien näher eingegangen. Abschließend wird die Implementierung in der Gitlab CI Pipeline erläutert.

3.1 Anforderungen

Aus der Aufgabenstellung ergeben sich bereits die wichtigsten Anforderungen. Zunächst soll ein Konzept für das Testen der MPFC basierend auf dem MATLAB® Unit Test Framework erarbeitet werden. Dafür muss eine geeignete Struktur gefunden werden, welche eine automatisierte Ausführung ermöglicht und Variationen von sowohl Testparametern (bspw. Initialgeschwindigkeiten) als auch MPC-Parametern (bspw. Gewichte der Kostenfunktion) leicht umzusetzen sind.

Es sollen Szenarien gefunden und parametriert werden, welche für den Regelungsalgorithmus eine Herausforderung darstellen und so das Verhalten in kritischen Situationen getestet werden kann.

Um die Simulationsergebnisse bewerten zu können, sollen geeignete Kriterien festgelegt werden, die sowohl Sicherheit als auch Komfort der gefundenen Lösung bewerten. Aus den Testergebnissen soll ein Bericht erstellt werden, welcher dem Entwickler übersichtlich mögliche Fehlschläge aufzeigt.

3.2 Simulationsumgebung

Die Simulationsumgebung und der Regelungsalgorithmus sind bereits in vorherigen Entwicklungsprojekten bei der IAV entstanden und sind in MATLAB®/Simulink implementiert. Aus den Streckeninformationen wird ein gewünschter Pfad mit einer Zielgeschwindigkeit generiert. Abbildung 3.1 zeigt den generierte Live-Plot, welcher bei Start der Simulation das Fahrzeug auf der Strecke darstellt.

3.3 MATLAB® Unit Test Framework

MATLAB®, in der für dieses Praktikum verwendeten Version 2022a und 2023b, stellt in der Basiskonfiguration ein umfangreiches Framework für die Implementierung von

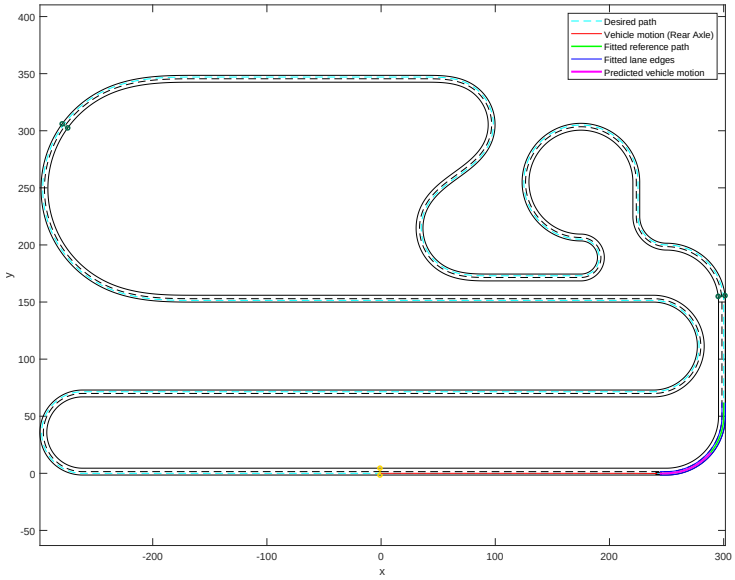


Abbildung 3.1: Beispiel einer komplexen Simulationsstrecke

Unit Tests zur Verfügung [10]. Eine Verwendung dieser Funktionalität für das Testen der MPFC liegt nahe, da diese vollständig in diesem Programm implementiert ist.

Grundsätzlich gibt es drei Arten von Testabläufen: skriptbasierte, funktionsbasierte und klassenbasierte Tests. Nach Befassen mit den entsprechenden Dokumentationsseiten wurde sich für die Nutzung der klassenbasierten Tests entschieden, da diese den größten Funktionsumfang bieten. Ein entscheidender Faktor ist die Möglichkeit, Tests durch externe Parameter zu parametrieren, was eine direkte Anforderung an das zu entwickelnde Konzept ist [10].

Das von Mathworks implementierte Framework ähnelt in vielen Gesichtspunkten der xUnit Architektur indem es die gleichen grundlegenden Strukturen aufweist [21]. Die kleinste Einheit dieser Architektur ist der *test case*. In diesem wird über *assertions*, dies sind meist logische Operationen, festgestellt, ob sich ein System im Normbereich verhält. Ist dies nicht der Fall, wird der Test abgebrochen und als fehlgeschlagen angesehen.

Um sicherzustellen, dass sich das *System Under Test (SUT)* vor einem Test in einem definierten Zustand befindet und vorherige Programmausführungen keine Auswirkungen auf die Testergebnisse haben wird über *test fixtures* dieser definierte Zustand hergestellt [13]. Bei klassenbasierten Tests gibt es mehrere Stellen an denen initiale Zustände hergestellt werden können. Mithilfe der Setup-Methoden können Voraussetzungen für mehrere Klassen (shared fixtures), für alle Testfälle einer Klasse (testclass setup) initial oder vor Ausführung jedes einzelnen Testfalls (testmethod setup) geschaffen werden.

Eine *test suite* ist eine Sammlung von Testfällen, welche die gleichen Initialzustände teilen. Da in der späteren Implementierung eine Testsuite immer aus lediglich einer Testklasse erstellt wird bezeichnen Testsuite und Testklasse hier das gleiche Objekt.

Bei klassenbasierten Tests ist es möglich, externe Parameter in die Testklasse zu laden. Diese können als Testparameter, Testinitialisierungsparameter oder Klasseninitialisierungsparameter übergeben werden. Abhängig davon, welche Parametertypen und wieviel Variationen definiert sind, werden Testfälle und die verschiedenen Setup-

Methoden mehrmals ausgeführt. Eine eigene Darstellung des Ablaufs einer Testklasse ist in Abbildung 3.2 zu sehen.

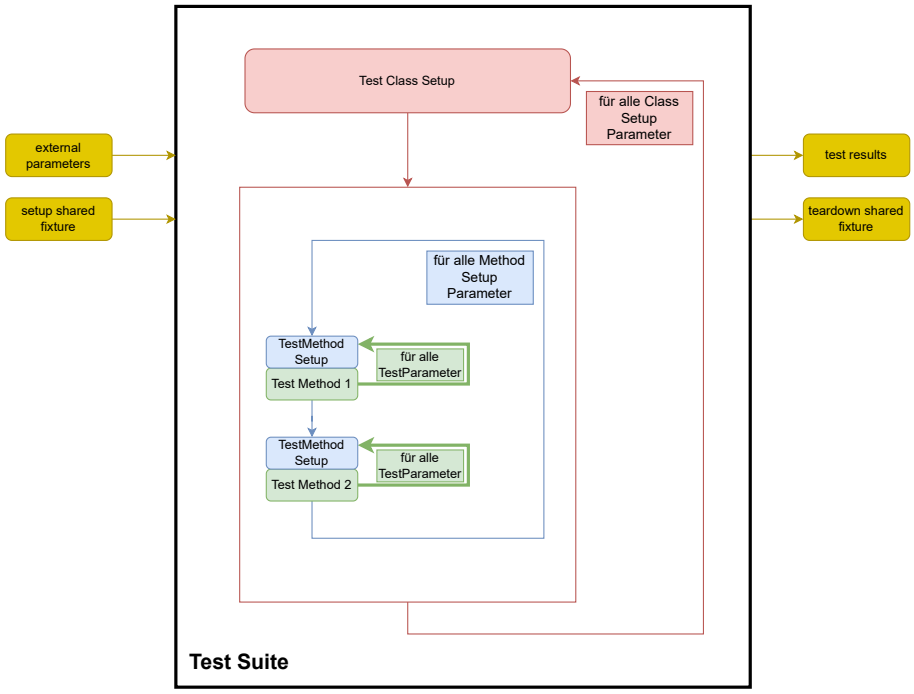


Abbildung 3.2: Ablauf von klassenbasierten Tests

Ein konfigurierbarer *test runner* führt die Tests aus. Das Framework sorgt selbstständig dafür, dass die notwendigen Setup-Methoden an den richtigen Stellen ausgeführt werden.

Für die Darstellung der Testergebnisse existiert ein *test result formatter*, welcher die Ergebnisse in vielen Formaten, beispielsweise PDF order XML, aufbereitet.

Die objektorientierte Natur von MATLAB® und dem Unit Test Framework ermöglichen zusätzlich die Erweiterung der Funktionalität durch das Schreiben von eigenen oder Erweiterung von vorhandenen Plugins.

3.4 Aufbau und Ablauf eines Szenarios

Nachdem die Funktionalitäten des Unit Test Frameworks erfasst wurden kann eine geeignete Implementierung basierend auf den Anforderungen erfolgen. Ein Szenario wird in der Simulation durch das Laden von entsprechenden Parametern bzw. Variablen definiert. Beim Starten der Simulation werden diese in das Simulink Modell geladen. Über die Protokolllierungsfunktion von Simulink werden die Simulationsergebnisse erfasst und für die weitere Verarbeitung abgespeichert.

3.4.1 Simulationsparameter

Zur vollständigen Beschreibung eines Szenarios werden initial folgende Parametersätze bzw. Variablen benötigt:

- Fahrzeugparameter
- Initialzustände
- MPC Parameter
- Streckeninformationen
- (nur für ACC) Objekttrajektorie

Die Fahrzeugparameter, wie zum Beispiel die Länge eines Fahrzeugs oder die Parameter für das Fahrzeugmodell, gehen aus der betriebsinternen Dokumentation hervor. Diese werden über ein Skript geladen und sind fahrzeugspezifisch. Der Parameter hier ist lediglich das verwendete Fahrzeug.

Der Initialzustand eines Fahrzeugs kann mehrere Parameter umfassen. Hier kann beispielsweise die Geschwindigkeit zu Beginn der Simulation oder ein Versatz zum gewünschten Pfad vorgegeben werden.

Die MPFC wurde durch geeignet Methoden, beispielsweise wie in [19] vorgestellt, für die gestellten Anforderungen der Fahrzeugführung parametriert. Es stehen für verschiedene Fahrweisen, von komfortabel bis sportlich, Parametersets zur Verfügung. Das gewählte Parameterset ist damit ein weiterer Simulationsparameter.

Nachdem Fahrzeug und Regler initialisiert sind, wird im nächsten Schritt ein zu folgender Pfad generiert. Dafür steht eine Bibliothek zur Verfügung, mit deren Hilfe ein Pfad aus einfachen geometrischen Objekten - Gerade, Kreisbogen und Klothoid - erzeugt werden kann. Eine Klothoide ist ein Kreisbogen, dessen Krümmung proportional zur Länge ist und somit keinen Sprung in der Krümmung aufweist [22].

Für Szenarien, welche die Funktionalität der MPFC als Abstandsregeltempomat (Adaptive Cruise Control - ACC) testen wird zusätzlich die Trajektorie des vorausfahrenden Fahrzeugs benötigt.

3.4.2 Ablauf eines Testskripts

Ein Testskript vereint die korrekte Initialisierung der Simulation durch Ausführung entsprechender Skripte, Erstellen einer Testsuite aus einer Testklasse und Parameterwerten, Ausführung der Testklasse bzw. Simulation und die Erstellung von Testberichten. Der schematische Ablauf ist in Abbildung 3.3 dargestellt.

Für den Start eines Szenarios wird der Codename für das Szenario und das zu verwendende Fahrzeug benötigt. Nach Ausführung von Initialisierungsskripten werden Parametersätze generiert, worauf in 3.4.3 näher eingegangen wird. Als nächstes wird aus einer Umgebungsklasse eine neue Instanz geschaffen. In dieser wird eine Testsuite erstellt. Wie in 3.3 bereits erwähnt besteht eine Testsuite immer aus einer Testklasse. Eine Testklasse implementiert ein Szenario, indem es die benötigten Variablen korrekt lädt, die Simulation ausführt und die Simulationsergebnisse in ihren Testfällen bewertet. Bei der Erstellung der Testsuite werden die vorher generierten Parameter in die Testklasse geladen.

Der konfigurierte Testrunner kann nun die Testsuite ausführen. Dabei werden die teilweise szenarien- und parameterabhängigen KPIs für die Bewertung der Simulationsergebnisse geladen. In 3.4.4 wird darauf näher eingegangen.

Aus den Ergbenisse der Tests wird schließlich automatisiert ein Testbericht in PDF-Format für den Entwickler und in XML-Format für die Darstellung in der CI Pipeline, siehe 3.5, generiert.

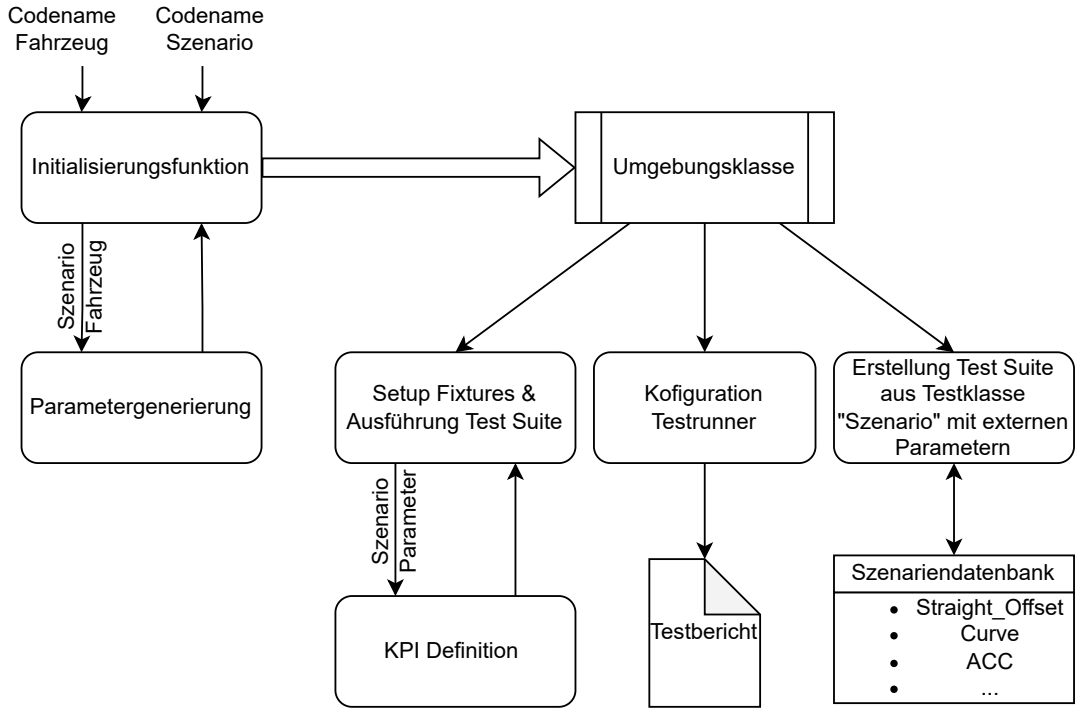


Abbildung 3.3: schematischer Ablauf eines Testskripts zur Ausführung eines Szenarios

3.4.3 Parametergenerierung

In bisherigen Testdurchführungen wurden die Parameter für Szenarien manuell gewählt und in ein Skript geschrieben. Für eine größere Anzahl an Parametern bzw. eine größere Testabdeckung ist dieses Verfahren sehr aufwändig und deckt möglicherweise nicht den gesamten Parameterraum ab. Wie bereits in Kapitel 2.2 geschildert ist es nicht möglich jede Parameterkombination eines ausgewählten Szenarios abzutesten. Besitzt ein Szenario N Parameter und exisitieren für jeden dieser Parameter k verschiedene Werte so werden bei einer vollständigen Abtestung des Parameterraums k^N Simulationen benötigt. Die Anzahl der Simulationen steigt exponentiell mit der Anzahl der Parameter an. Vorallem bei komplexeren Szenarien ist diese Parametrierungsstrategie nicht anwendbar.

Um die Anzahl der notwendigen Simulationen zu reduzieren wurden folgende Strategien angewendet:

- sinnvolle Begrenzung des Parameterraums

- Implementierung einer (zufälligen) Sampling Methode
- Diskretisierung des Parameterraums

Die Begrenzung des Parameterraums geschieht durch festlegen von Ober- und Untergrenzen für Parameterwerte. Diese Grenzen können durch das verwendete Fahrzeug gegeben sein, beispielsweise minimale Lenkradien oder Höchstgeschwindigkeiten oder durch Normen festgelegt. Beispielsweise werden in [7] minimale Kurvenradien bei der Anlage von Straßen beschrieben. Noch fehlende Parametergrenzen wurden durch Testen der Funktionen der aktuellen MPFC Implementierung sinnvoll festgelegt.

Die Implementierung einer Sampling Methode, wie zum Beispiel dem Monte-Carlo-Verfahren, ermöglicht eine Reduzierung der Anzahl der benötigten Simulationen. Für jeden Parameter wird aus dem jeweiligen Parameterraum ein zufälliger Wert gezogen. Dies wird für eine vorher definerte Anzahl an Samples wiederholt.

Es kann allerdings nicht gewährleistet werden, dass der gesamte Parameterraum abgedeckt wird. Deshalb wird in der Implementierung das Latin Hypercube Sampling (LHS) angewendet, welches eine bessere Verteilung der Samples im Parameterraum gewährleistet, wie in Abbildung 3.4 dargestellt ist. Grundidee hierbei ist, dass jeder Parameterraum in k Teilintervalle unterteilt wird. Es werden k Samplesets generiert. Aus jedem Teilintervall wird dabei immer genau ein Wert zufällig gezogen, sodass nach Abschluss der gesamte Parameterraum abgedeckt ist. Aufgrund der zufälligen Ziehung innerhalb eines Intervalls ist das LHS dennoch eine Monte-Carlo Methode [11].

Nach Ausführung dieser Funktion entstehen k-Samplesets, die für alle Parameter in den jeweiligen Grenzen eine gute Abdeckung des Parameterraums bieten.

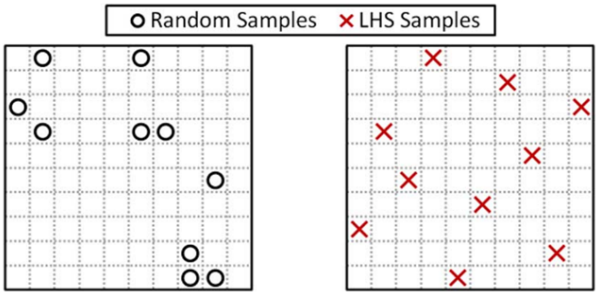


Abbildung 3.4: Vergleich der Verteilung von zufälligen Sampling mit Latin Hypercube Sampling [16]

3.4.4 Definition von Key Performance Indicators und Testkriterien

Um eine Bewertung der Simulationsergebnisse vornehmen zu können werden geeignete, zu bewertende Daten, benötigt und Grenzwerte für diese. Tritt ein Über- bzw. Unterschreiten dieser auf wird ein entsprechender Test als fehlgeschlagen definiert. Diese Kriterien werden als KPI bezeichnet.

Ein KPI kann beispielsweise die Zeit bis zum Abbauen eines initialen lateralen Versatzes sein. Um diese zu ermitteln werden die Pfaddaten und die tatsächliche Position

Tabelle 3.1: Höchstwerte für laterale Beschleunigung, aufgeschlüsselt nach Fahrzeugklassen und Geschwindigkeitszonen [20]

Für Fahrzeuge der Klassen M1 und N1				
Geschwindigkeitsbereich	10-60 km/h	> 60-100 km/h	> 100-130 km/h	> 130 km/h
Höchstwert für die angegebene maximale Querbeschleunigung	3 m/s^3	3 m/s^3	3 m/s^3	3 m/s^3
Mindestwert für die angegebene maximale Querbeschleunigung	0 m/s^3	0 m/s^3	0 m/s^3	0 m/s^3
Für Fahrzeuge der Klassen M2, M3, N2 und N3				
Geschwindigkeitsbereich	10-30 km/h	> 30-60 km/h	> 60 km/h	
Höchstwert für die angegebene maximale Querbeschleunigung	2,5 m/s^3	2,5 m/s^3	2,5 m/s^3	
Mindestwert für die angegebene maximale Querbeschleunigung	0 m/s^3	0,3 m/s^3	0,5 m/s^3	

des Fahrzeugs benötigt. Wird die Differenz von beiden Datensätzen nicht innerhalb einer definierten Zeitspanne abgebaut gilt der Test als fehlgeschlagen.

Es existieren diverse Normen und Studien zur Bewertung von autonomen Fahrzeugen hinsichtlich Sicherheit und Komfort. In [20] werden Bedingungen für die Zulassung von Fahrzeugen, welche in die Querführung des Fahrzeugs eingreifen definiert. [18] etabliert Standards für die Funktionen und Testkriteren für ein ACC-System.

Laterale Beschleunigung

Die laterale Beschleunigung hat großen Einfluss auf den Fahrkomfort. In [18] wurde das Verhalten eines durchschnittlichen Fahrers in Kurven bei verschiedenen Geschwindigkeiten analysiert, siehe Abbildung 3.5. In [20] werden für verschiedene Fahrzeugklassen in verschiedenen Geschwindigkeitszonen maximale Querbeschleunigungen definiert, siehe Tabelle 3.1.

Longitudinale Beschleunigung

Diese KPI ist vorallem im Kontext des ACC von großer Relevanz. In [18] sind dafür einige Grenzwerte definiert. Betrachtet werden hier keine Maximalwerte sondern der gleitende Durchschnitt über zwei Sekunden.

Ruck

Ruck ist ein unerwünschtes Verhalten bei Fahrzeugen, das durch ungleichmäßige Beschleunigung und Bremsen entsteht und maßgeblich für den Komfort mitverantwortlich ist. [18] definiert im ACC Kontext dafür einen Grenzwert für einen negativen longitudinalen Ruck, wie er beim Abbremsen entsteht. Im Kontext dieser Arbeit wird die Bewertung von einem positiven Ruck und der Ruck in lateraler Richtung ähnlich bewertet. In [20]

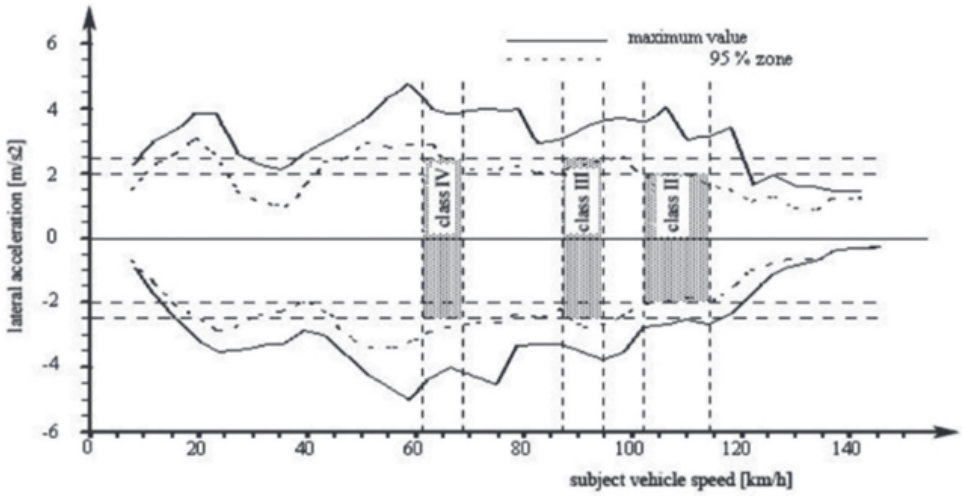
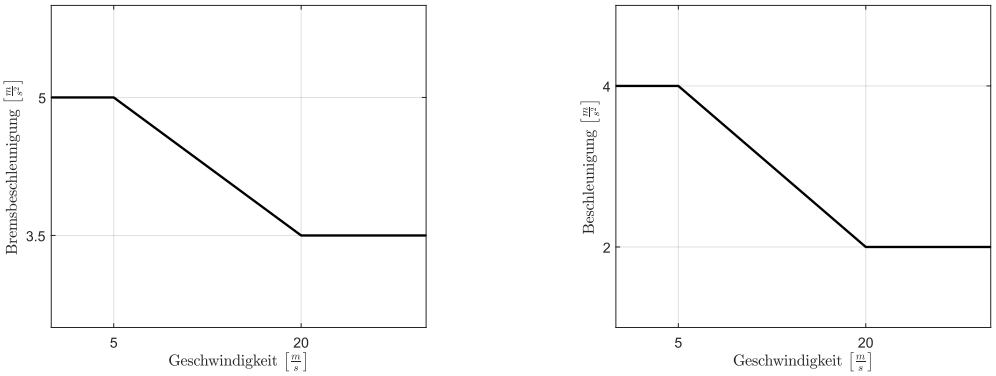


Abbildung 3.5: laterale Beschleunigung in Kurven eines typischen Fahrers [18]



(a) maximale Beschleunigung **(b)** maximale Bremsbeschleunigung

Abbildung 3.6: maximal zulässige longitudinale Beschleunigung, Durchschnitt über 2 s [18]

darf der gleitende Mittelwert des lateralen Rucks über eine halbe Sekunde nicht größer als $5\frac{m}{s^3}$ sein.

Pfadgeschwindigkeitseinhaltung
Die Einhaltung der vorgegebenen Pfadgeschwindigkeit ist aus Sicht der Sicherheit und der Gesetzgebung ein wichtiges Kriterium. Eine Unterschreitung ist hier weniger kritisch zu betrachten. Eine Überschreitung sollte vom System vermieden werden. Im Testkatalog von Euro NCAP bekommt ein System, welches die Geschwindigkeit automatisch anpassen kann, die maximale Punktzahl, wenn es die Geschwindigkeit auf $\pm 2\frac{km}{h}$ anpassen kann, bevor die Frontachse des Fahrzeugs eine Zone mit einer niedrigeren Pfadgeschwindigkeit betritt [15]. Da die Simulation die Hinterachse als Referenz nutzt und darauf regelt wird dieses Kriterum dahingehend angepasst.

Weitere KPIs
Neben den oben genannten KPIs, welche in der Literatur gängige Anwendung finden und quantifiziert werden, wurden für die jeweiligen Szenarien Weitere definiert.

- **laterale Ablage** - Abweichung der Fahrzeugposition von dem gewünschten Pfad

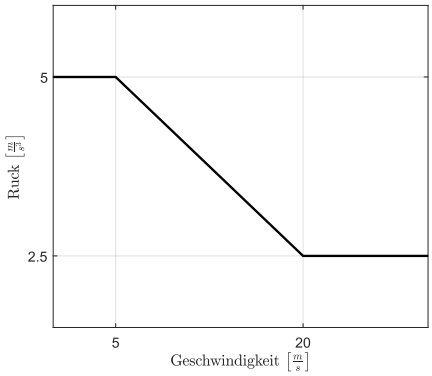


Abbildung 3.7: maximal zulässiger negativer Ruck, Durchschnitt über 1 s [18]

- **Einschwingzeit** - benötigte Zeit, bis ein Einschwingvorgang abgeschlossen ist
- **Reaktionszeit** - benötigte Zeit, bei der eine Sprungantwort zum ersten mal die neue Sollgröße erreicht
- **Differenzgeschwindigkeit** - Geschwindigkeitsdifferenz zwischen Ego-Fahrzeug und dem ACC-Target
- **Distanzfehler** - Abweichung von der gewünschten Distanz zwischen dem Ego und einem Zielobjekt

3.5 Integration in die GitLab CI Pipeline

Für das automatisierte Testen in der Pipeline wird eine .yaml-Datei benötigt [9]. In dieser werden Jobs definiert. Ein Job ist die kleinste Einheit einer Pipeline und definiert eine zu erledigende Aufgabe. Eine Pipeline ist somit eine Ansammlung von zeitgleich gestarteten Jobs. Ein Job besteht mindestens aus der Ausführung eines Skriptes, ähnlich einer Kommandozeilenumgebung. Weiterhin kann definiert werden, wann ein Job gestartet wird und welche Voraussetzungen für diesen geschaffen werden müssen. Für jede Szenario und Fahrzeugkombination kann ein Job erstellt werden. Ein Job führt zunächst Setup-Skripte aus, damit anschließend Matlab® über eine Befehlszeileneingabe gestartet werden kann. Der Ablauf ist ähnlich zu dem in Abschnitt 3.3 beschrieben. Es wird lediglich ein anderes Initialisierungsskript verwendet, um mit den geänderten Gegebenheiten in der Pipeline umzugehen.

3.5.1 Darstellung der Pipeline

Um die Wartung der .yaml-Datei und die Ausführung übersichtlicher zu gestalten wird die Funktionalität von Gitlab ausgenutzt, Jobs in einer Matrix zu parametrisieren. Dabei wird lediglich ein Job für jedes Fahrzeug erstellt. In diesem Job werden über den *matrix* Befehl die gewünschten Szenarien, welche für dieses Fahrzeug abgetestet werden sollen, hinterlegt. Die Pipeline generiert dann automatisch für jede Kombination aus Fahr-

zeug und Szenario einen eigenen Job an. Durch geeignete Wahl der Testklassennamen ist es zusätzlich möglich in einem Job mehrere Testklassen/Szenarien nacheinander laufen zu lassen. Beispielsweise wird im Job “test_Curve“ sowohl ein normales Kurvenfahrtszenario als auch der in Abschnitt 4.2 vorgestellte Negativtest für eine Kurve ausgeführt. Weiterhin können durch die Verwendung des *parallel* Kennworts mehrere Jobs parallel ablaufen, was die Laufzeit der Pipeline enorm verkürzt. Durch diese Einstellung der Pipeline ergibt sich eine sehr übersichtliche Darstellung im Browser (Abbildung 3.8). Aus dieser ist direkt zu entnehmen, welche Szenarios für welches Fahrzeug (Alice, IAVShuttle) ausgeführt wurden und welche Szenarien zu Fehlern geführt haben.

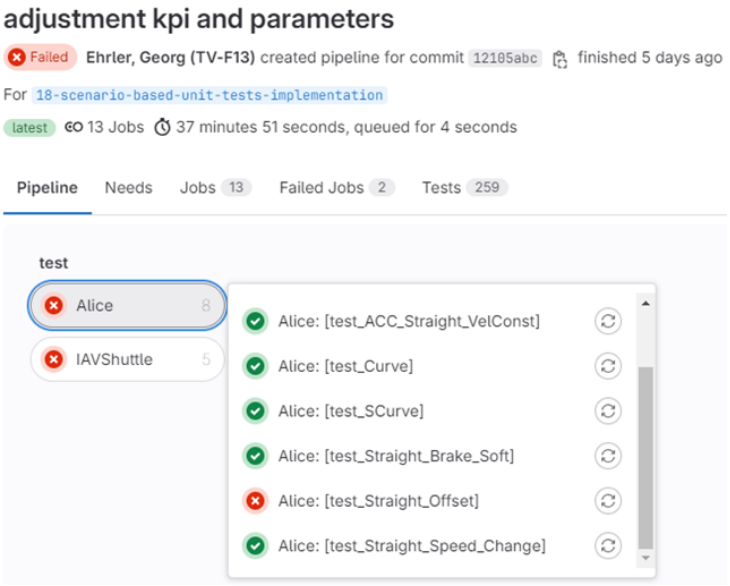


Abbildung 3.8: Darstelllung der Jobs in der Gitlab CI Pipeline im Browser. “Alice“ und “IAVShuttle“ sind Codenamen von Prototypenfahrezeugen

3.5.2 Darstellung der Testergebnisse

Nachdem ein Szenario durchlaufen ist, werden die Testergebnisse sowohl in GitLab, als auch in einer PDF-Datei dargestellt. Dieser Testbericht wird als Artefakt des jeweiligen Jobs behalten und kann vom Entwickler herunterladen werden.

Auf der Startseite befindet sich eine Übersicht (Abbildung 3.9). Schlägt ein Test fehl gibt es einen ausführlicheren Bericht (Abbildung 3.10). Die Testdiagnose teilt dem Entwickler mit, was das Problem ist. Der Vergleich von soll KPI und tatsächlichem Ergebnis folgt darunter. Wenn es für den Test sinnvoll ist wird ebenfalls ein Plot generiert, der den zeitlichen Verlauf von relevanten Größen über die Zeitdauer der Simulation darstellt. Die Parameter des aktuellen Szenarios werden ebenfalls angegeben, um den Test lokal reproduzieren zu können.

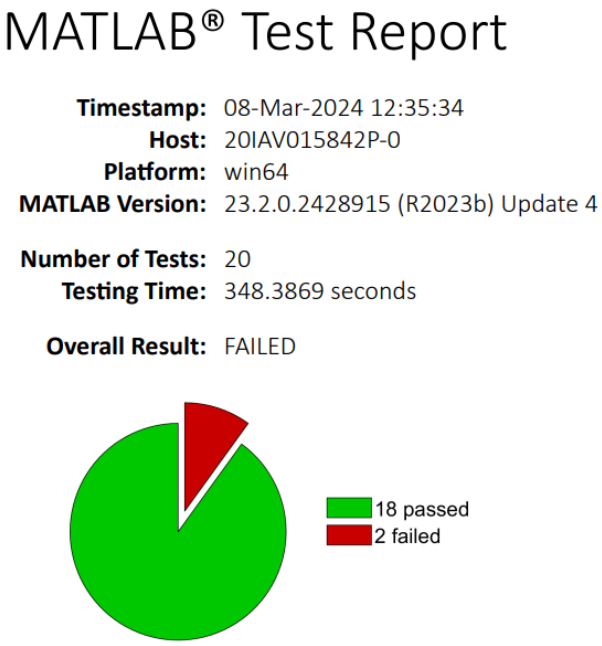


Abbildung 3.9: Übersichtliche Darstellung der Testergebnisse eines Szenarios auf der ersten Seite des Testberichts

s_DistTargetError
Class Setup Parameters: testParams=struct_3

The test failed.
Duration: 45.1431 seconds

Event:
Verification failed.

Test Diagnostic:
```
Vehicle takes too long to establish the minimum distance.
```
Framework Diagnostic:
```
verifyLessThan failed.
--> The value must be less than the maximum value.

Actual Value:
   9.680000000000000
Maximum Value (Exclusive):
     8
```
Additional Diagnostic 1:
```
{"scenario":"test_ACC_Straight_VelConst","vehicle":"Alice","v_egoInit":97.55,"t_gap":-0.85,"v_diff":7.804,"parameter
Set":"SET_2","v_Target":89.746,"v_path":130.06666666666666}
```
Additional Diagnostic 2:
```
Figure saved to:
--> C:\ADPL\Runner\builds\sG-gTE6b\1\AD-platform\projects\ctrl_had\project_ctrl_pathfollowingmpc\Artifacts\857ec331-a
ba2-423b-883a-1de5b6e614d4\s_DistTargetError8c57904c-1bd4-46ee-95b5-545303Ib8ce3.png
```

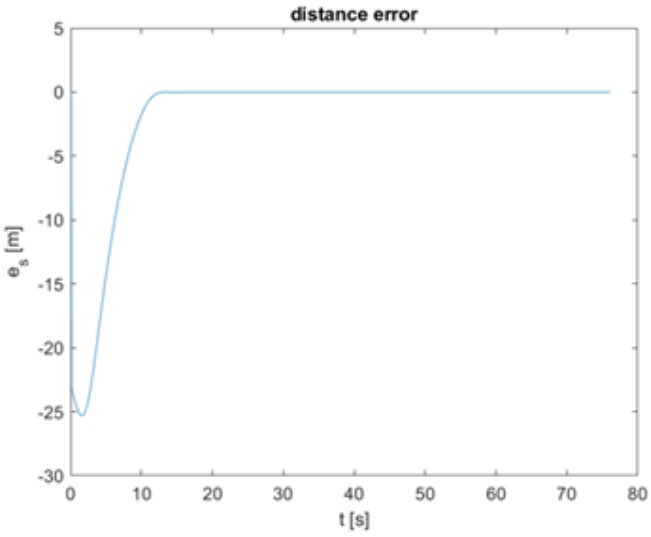


Abbildung 3.10: Darstellung eines fehlgeschlagenen Tests im Testbericht. Der Wert der KPI und der tatsächliche Wert werden verglichen. Ein kurzer Text gibt dem Entwickler wieder, was fehlgeschlagen ist. Wenn sinnvoll wird ein Plot mit den relevanten Daten erzeugt. Die Angabe der Simulationsparameter genutzt werden um diese eine Simulation lokal zu wiederholen.

Vorstellung ausgewählter Szenarien

4

Um geeignete Szenarien für die Simulation zu erstellen, wurden verschiedene Quellen berücksichtigt:

- betriebsinterne Szenariendatenbank
- Normen und Richtlinien
- eigene Überlegungen nach Systemanalyse

Es steht intern eine Szenariendatenbank zur Verfügung. Für diese Simulation sind viele Szenarien allerdings nicht von Relevanz, da die Pfadplanung von einem übergeordneten Modul realisiert wird und bspw. Kreuzungsszenarien zu Kurvenszenarien reduziert werden. In [18] und [15] werden verschiedene Szenarien für die Einhaltung gesetzlicher Vorschriften und ACC Systeme definiert. Weiterhin wurden auch Negativtests implementiert. Dabei wird ein Szenario bewusst so parametriert, dass es der Simulation nicht möglich ist die KPIs einzuhalten. In diesen Fällen ist ein fehlschlagen eines Test als positiv zu werten. Ein Test schlägt fehl, wenn die Grenzwerte nicht überschritten wurden sind.

Im folgenden werden exemplarisch einzelne Szenarien näher erläutert.

4.1 Szenario: Gerade mit initialem Versatz

Das Ego startet auf einer Geraden. Die Initialgeschwindigkeit ist gleich der Pfadgeschwindigkeit. Es besteht ein initialer lateraler Versatz von der Ego Position zum gewünschten Pfad. Das Ego sollte den initialen Versatz abbauen und mit der Pfadgeschwindigkeit auf dem gewünschten Pfad fahren. Die gerine Komplexität und Parameteranzahl des Szenarios ermöglichen eine einfache Analyse von Auswirkungen der Veränderung eines Parameters bei gleichzeitiger Konstanthaltung der anderen Parameter. Dafür wurde in allen Diagrammen das gleiche MPC-Parameterset und das gleiche Fahrzeug verwendet.

4.1.1 Parametrierung

Die Parameter für dieses Szenario sind:

- Offset $[m]$: $s_{offset} \in [-1.5, 1.5]$
- Initialgeschwindigkeit $[km \cdot h^{-1}]$: $v_{Ego} \in [10, v_{vehicle,max}]$
- verwendetes MPC Parameterset

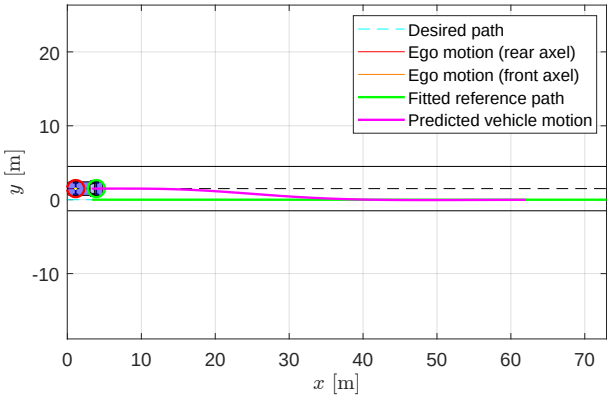
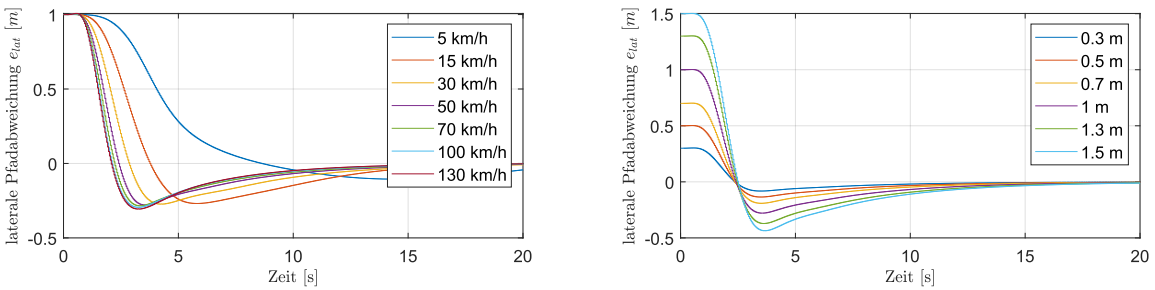


Abbildung 4.1: Darstellung des Szenarios: Gerade mit initialem Versatz

4.1.2 Abgetestete KPIs

Es wurden Test für folgende KPIs mit den dazugehörigen Grenzwerten implementiert:

Einschwingzeit:: der laterale Versatz sollte innerhalb einer bestimmten Zeit abgebaut werden: $t_{settle} < kpi_{t_{settle}}$
Eine Variation des Versatzes bei gleicher Geschwindigkeit von 50 km/h (Abbildung 4.2b) führt zu sehr ähnlichem Verhalten und einer nahezu identischen Reaktionszeit (erster Nulldurchlauf). Je größer der Versatz desto höher ist das Überschwingen auf der anderen Seite des Pfades, was zu einer längeren Einschwingzeit führt. Bei der Variation der Geschwindigkeit (Abbilfung 4.2a) und konstantem Versatz von 1 m verschiebt sich der Zeitpunkt des maximalen Überschwingens nach hinten, je geringer die Geschwindigkeit wird. Durch die Wahl eines Grenzwertes der maximalen Pfadabweichung kann die Einschwingzeit bestimmt werden, indem das letzte Überschreiten des Grenzwertes ermittelt wird.



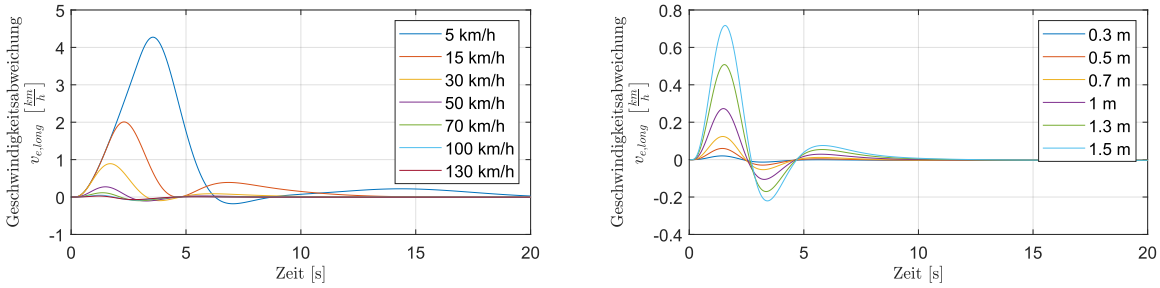
(a) Variation der Initialgeschwindigkeit

(b) Variation des initialen Versatzes

Abbildung 4.2: KPI Pfadabweichung abhängig der Simulationsparameter

longitudinale Geschwindigkeit: die Geschwindigkeitsabweichung zum Pfad darf den Grenzwert nicht überschreiten: $|v_{Long} - v_{Path}| < kpi_{v_{diff}}$
Die Pfadgeschwindigkeit ist in der Implementierung der MPFC ein Optimierungsparameter. Ein verhindern des Überschreitens der Pfadgeschwindigkeit ist nicht garantiert. Um den Versatz möglichst schnell abzubauen beschleunigt der Regler das Fahrzeug.

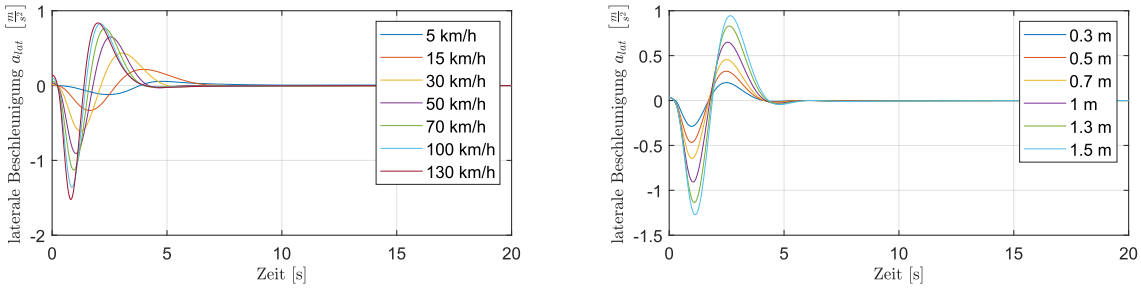
Dies ist vorallem bei niedrigeren Geschwindigkeiten und höheren Versätzen zu beobachten, siehe Abbildungen 4.2.



(a) Variation der Initialgeschwindigkeit **(b)** Variation des initialen Versatzes

Abbildung 4.3: KPI Geschwindigkeitsabweichung abhängig der Simulationsparameter

laterale Beschleunigung: der gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $a_{Lat} < kpi_{a_{Lat}}$
Wie aus Abbildung 4.4 hervorgeht führen höhere Geschwindigkeiten und ein größerer initialer Versatz erwartungsgemäß zu größeren Amplituden.



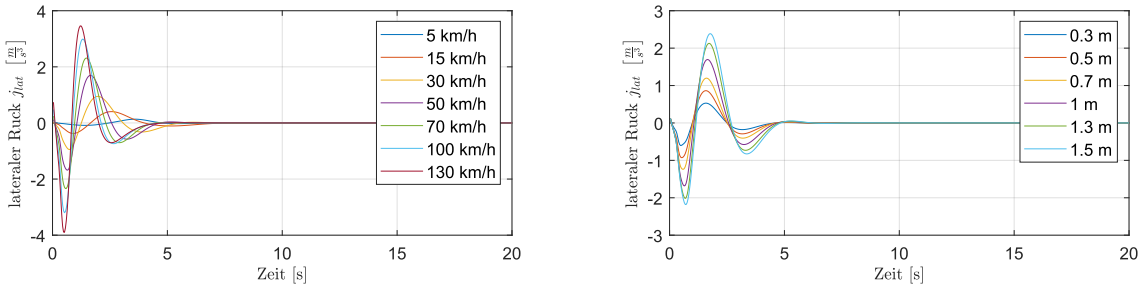
(a) Variation der Initialgeschwindigkeit **(b)** Variation des initialen Versatzes

Abbildung 4.4: KPI laterale Beschleunigung abhängig der Simulationsparameter

lateraler Ruck: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $j_{Lat} < kpi_{j_{Lat}}$
Wie aus Abbildung 4.5 hervorgeht führen höhere Geschwindigkeiten und ein größerer initialer Versatz erwartungsgemäß zu größeren Amplituden.

4.2 Szenario: Kurve negativ

Das Ego startet mit einer Initialgeschwindigkeit, welche der Pfadgeschwindigkeit entspricht auf einer Geraden. An die Gerade schließt sich eine Kurve an, welche in einen Kreis endet. Der Radius des Kreises ist kleiner als der minimale Wendekreis des Fahrzeugs. Das Ego soll die Beschränkungen des Lenkeinschlags nicht verletzen. Dies führt dazu, dass das Fahrzeug die Kurve nicht schaffen kann und der Abstand zum gewünschten Pfad zu groß wird.



(a) Variation der Initialgeschwindigkeit **(b)** Variation des initialen Versatzes

Abbildung 4.5: KPI lateraler Ruck abhängig der Simulationsparameter

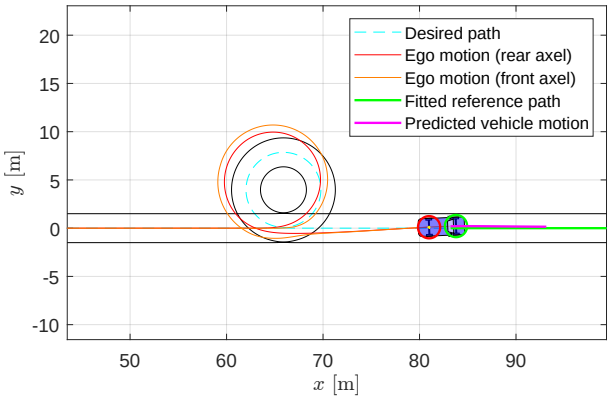


Abbildung 4.6: Darstellung des Szenarios: Kurvenfahrt negativ

4.2.1 Parametrierung

Die Parameter für dieses Szenario sind:

- maximale Querbeschleunigung in der Kurve $[m \cdot s^{-2}]$: $a_{Lat} \in [1, 4]$
- Kurvenradius $[m]$: $s_{radius} \in [0.7, 0.9] \cdot r_{min,turn}$
- verwendetes MPC Parameterset

4.2.2 abgetestete KPIs

Es wurden Test für folgende KPIs mit den dazugehörigen Grenzwerten implementiert:

laterale Ablage: die Abweichung des Fahrzeugs vom gewünschten Pfad muss größer sein als ein definierter Grenzwert: $s_{error,lat} > kpi_{s_{error,lat}}$
Wie erwartet führen kleinere Kurvenradien zu größeren maximalen Pfadabweichungen, Abbildung 4.7. Nachdem die Kurve durchfahren ist kann dieser Fehler wieder abgebaut werden.

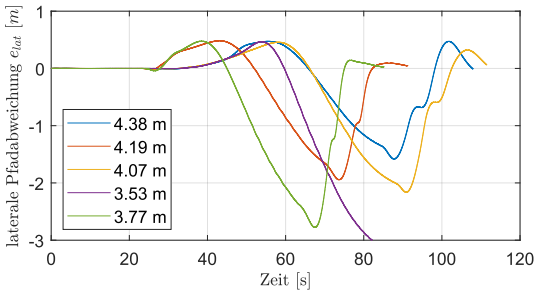


Abbildung 4.7: KPI Pfadabweichung bei verschiedenen Kurvenradien

4.3 Szenario: ACC mit konstanter Objektgeschwindigkeit

Das Ego startet mit einer Initialgeschwindigkeit auf einer Geraden. Ein Target fährt vor dem Ego mit konstanter Geschwindigkeit. Es exisitiert eine Geschwindigkeitsdifferenz zwischen Ego und Target. Ebenfalls kann die initiale Zeitlücke abweichend von der gewünschten Zeitlücke sein. Das Ego sollte die Geschwindigkeitsdifferenz beseitigen und die gewünschte Zeitlücke herstellen.

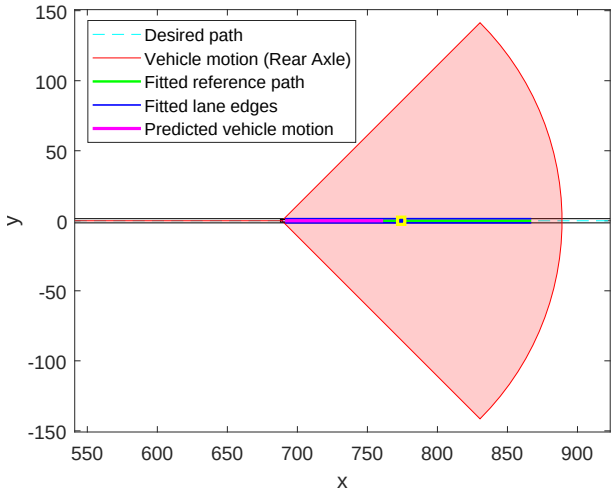


Abbildung 4.8: Darstellung des Szenarios: ACC mit konstanter Objektgeschwindigkeit

4.3.1 Parametrierung

Die Parameter für dieses Szenario sind:

- Initialgeschwindigkeit $[km \cdot h^{-1}]$: $v_{Ego} \in [0.2 \cdot v_{vehicle,max}, 0.8 \cdot v_{vehicle,max}]$
- Differenzgeschwindigkeit $[km \cdot h^{-1}]$: $v_{diff} \in [-0.2, 0.2] \cdot v_{Ego}$
- Zeitlückenabweichung $[s]$: $t_{offset} \in [-1, 1] + t_{gap,desired}$
- verwendetes MPC Parameterset

4.3.2 Abgetestete KPIs

Es wurden Test für folgende KPIs mit den dazugehörigen Grenzwerten implementiert:

longitudinale Beschleunigung: der Gleitende Mittelwert über eine halbe Sekunde darf einen definierten Grenzwert nicht überschreiten: $a_{Long} < kpi_{a_{Long}}$

longitudinaler Ruck: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $j_{Long} < kpi_{j_{Long}}$
Auftretende longitudinale Rücke und Beschleunigungen sind stark abhängig von der gewählten Paramtrierung. Eine Verletzung der Grenzwerte ist hier nicht zulässig. Vorallem darf die maximale Beschleunigung nicht größer sein als ein definierter Grenzwert. Sowohl Beschleunigung als auch Ruck befinden sich in einem akzeptablem Rahmen, siehe Abbildung 4.9.

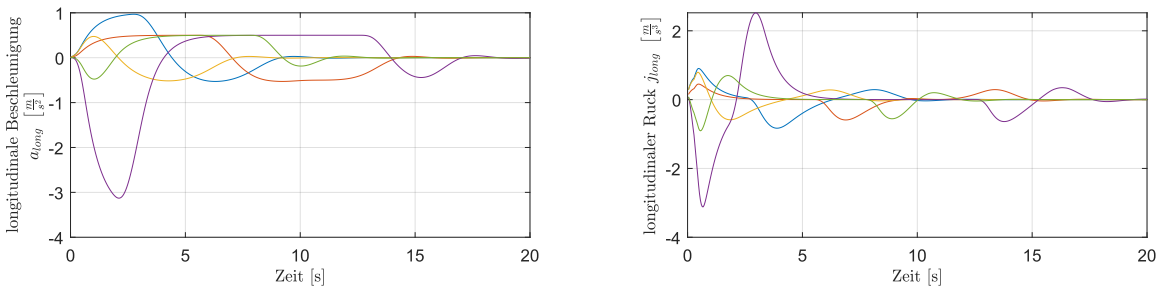


Abbildung 4.9: Simulationsdaten für Abstands- und Distanzfehler bei unterschiedlichen Parametrierungen

Einschwingzeit: der gewünschte Abstand zum Target sollte innerhalb einer definierten Zeit eingestellt werden: $t_{settle} < kpi_{t_{settle}}$
Durch ein optimales Systemverhalten kann die Einschwingzeit deutlich veringert werden. Allerdings ist die Definition eines festen Grenzwertes nicht möglich, da diese stark Parametrierungsabhängig ist (Abbildung 4.10).

Reaktionszeit: der Minimalabstand zum Target soll innerhalb einer definierten Zeit hergestellt werden: $t_{reaction} < kpi_{t_{reaction}}$
Das schnelle einstellen des Mindestabstandes ist wichtiger als die korrekte Abstandseinstellung, weswegen hier ein kürzeres Zeitfenster gewählt wurde.

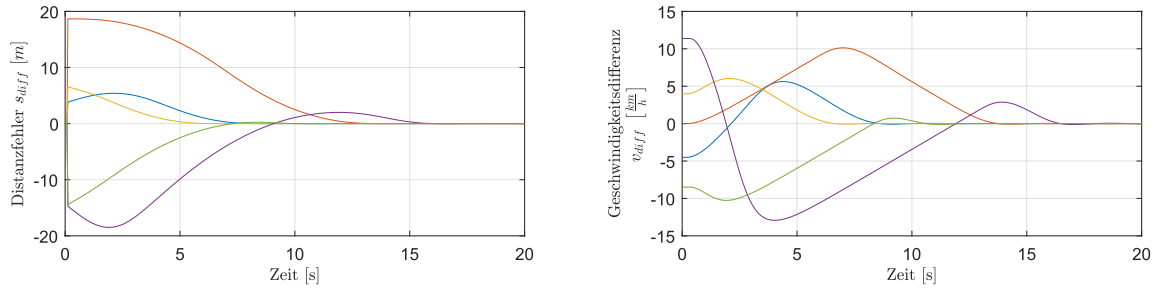


Abbildung 4.10: Simulationsdaten für longitudinale Beschleunigung und Ruck bei unterschiedlichen Parametrierungen

5 Zusammenfassung und Ausblick

Nachdem in der Einleitung die Einordnung und Vorstellung des Themas erfolgte, wurde im Grundlagenkapitel auf die zu testende Architektur eingegangen und eine theoretische Hinführung zum Thema des szenariobasierten Testen gegeben und die Relevanz von Softwaretest über den gesamten Produktlebenszyklus verdeutlicht.

Nach einer Anforderungsdefinition wurden die Möglichkeiten, die des MATLAB® Unit Test Framework für die Umsetzung der Aufgabe bietet, ergründet und daraus ein geeignetes Konzept zur Implementierung erarbeitet. Es ermöglicht die einfache Parametrierung von Szenarien und abtesten von KPIs. Neue Szenarien können durch das Anlegen einer neuen Testklasse, Definition von KPIs und Paramertern hinzugefügt werden und müssen im jeweiligen Job für ein Fahrzeug ergänzt werden. Sollten in zukünftigen Entwicklungen weitere Fahrzeuge hinzukommen ist lediglich ein neuer Job in der .yaml-Datei für die GitLab Pipeline anzulegen.

Nach Analyse der Simulationsumgebung und einer Literaturrecherche wurden ausgewählte Szenarien und die dazugehörigen KPIs implementiert. Um aus den ermittelten funktionalen Szenarien konkrete Szenarien zu generieren wurde das Latin Hypercube Sampling verwendet. Eine zukünftige Verbesserung dieser Sampling-Methode hinsichtlich der Generierung von kritischeren Parameterkombinationen ist denkbar. Derzeit wird, z. B. bei der Kurvenradien, aquidistant über den gesamten Parameterbereich gesampelt. Enge Kurven sind allerdings deutlich kritischer als weite Kurven, weswegen eine engere Abtastung bei kleineren Radien sinnvoller ist. Eine Möglichkeit der Umsetzung dieser Anforderung ist die Äquivalenzklassenbildung. Dabei werden für jeden Parameter Bereiche gesucht, in denen das erwartete Systemverhalten gleich ist. Es wird nun so gesampelt, dass entweder aus jeder Äquivalenzklasse genau ein Wert gebildet wird, analog zum LHS, oder aber die Anzahl an Samples in jeder Aquivalenzklasse gleich groß ist.

Die automatisierte Ausführung der Testskripte und Testklassen in einer CI Pipeline wurde erreicht. Die Testergebnisse werden übersichtlich in ein PDF-Datei verpackt oder können über den Browser im GitLab angezeigt werden. Die Testberichte geben dem Entwickler einen guten Überblick, welche Szenarien fehlgeschlagen sind und warum.

Literatur

[1] Jürgen Adamy. *Nichtlineare Systeme und Regelungen*. Berlin Heidelberg New York: Springer-Verlag, 2014. ISBN: 978-3-642-45013-6. DOI: 10.1007/978-3-642-45013-6.

[2] Gerrit Bagschik, Till Menzel und Markus Maurer. „Ontology based Scene Creation for the Development of Automated Vehicles“. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, S. 1813–1820. DOI: 10.1109/IVS.2018.8500632.

[3] Hanna Bellem, Barbara Thiel, Michael Schrauf und Josef F. Krems. „Comfort in automated driving: An analysis of preferences for different automated driving styles and their dependence on personality traits“. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 55 (2018), S. 90–100. ISSN: 1369-8478. DOI: 10.1016/j.trf.2018.02.036.

[4] Statistisches Bundesamt. „Verkehrsunfälle“. In: *Fachserie 8 Reihe 7* (Sep. 2022).

[5] E.F. Camacho und C.B. Alba. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2007. ISBN: 978-1-85233-694-3. DOI: 10.1007/978-0-85729-398-5.

[6] T. Faulwasser, B. Kern und R. Findeisen. „Model predictive path-following for constrained nonlinear systems“. In: *48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 2009, S. 8642–8647. DOI: 10.1109/CDC.2009.5399744.

[7] Bauformel Verlag GmbH. *Straßenbau Grenz und Richtwerte*. 24. Jan. 2019. URL: https://www.bauformeln.de/strassenbau/grenz-und-richtwerte.

[8] Red Hat. *Was ist CI/CD? Konzepte und CI/CD Tools im Überblick*. 2024. URL: https://www.redhat.com/de/topics/devops/what-is-ci-cd (besucht am 18. 03. 2024).

[9] GitLab Inc. *GitLab Dokumentation*. 2024. URL: https://docs.gitlab.com/.

[10] The MathWorks Inc. *Matlab Unit Test Framework*. 2023. URL: https://de.mathworks.com/help/matlab/matlab-unit-test-framework.html.

[11] M. D. McKay, R. J. Beckman und W. J. Conover. „A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code“. In: *Technometrics* 21.2 (1979), S. 239–245. ISSN: 00401706. DOI: 10.2307/1268522.

[12] Till Menzel, Gerrit Bagschik und Markus Maurer. „Scenarios for Development, Test and Validation of Automated Vehicles“. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, S. 1821–1827. DOI: 10.1109/IVS.2018.8500406.

[13] Gerard Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[14] Đemin Nalić, Tomislav Mihalj, Maximilian Baeumler, Matthias Lehmann, Arno Eichberger und Stefan Bernsteiner. „Scenario Based Testing of Automated Driving Systems: A Literature Survey“. In: *FISITA Web Congress 2020*. Okt. 2020. DOI: 10.46720/f2020-acm-096.

[15] Euro NCAP. *Assisted Driving Highway & Interurban Assist Systems. Test & Assessment Protocol*. Version 2.1. Feb. 2024. URL: https://www.euroncap.com/en/for-engineers/protocols/general/.

[16] Robin Preece und Jovica Milanović. „Efficient Estimation of the Probability of Small-Disturbance Instability of Large Uncertain Power Systems“. In: *IEEE Transactions on Power Systems* 31 (Apr. 2015), S. 1–10. DOI: 10.1109/TPWRS.2015.2417204.

[17] Robert Ritschel, Frank Schrödel, Juliane Hädrich und Jens Jäkel. „Nonlinear Model Predictive Path-Following Control for Highly Automated Driving“. In: *IFAC-PapersOnLine* 52.8 (2019). 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019, S. 350–355. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2019.08.112.

[18] International Organization for Standardization. *ISO 15622:2018. Intelligent transport systems — Adaptive cruise control systems — Performance requirements and test procedures*. Sep. 2018.

[19] David Stenger, Robert Ritschel, Felix Krabbes, Rick Voßwinkel und Hendrik Richter. „What Is the Best Way to Optimally Parameterize the MPC Cost Function for Vehicle Guidance?“ In: *Mathematics* 11.2 (2023). ISSN: 2227-7390. DOI: 10.3390/math11020465.

[20] Wirtschaftskommission der Vereinten Nationen für Europa. *UNECE-R79. Einheitliche Bedingungen für die Genehmigung der Fahrzeuge hinsichtlich der Lenkanlage*. 16. Okt. 2018.

[21] Wikipedia. *XUnit*. Wikimedia Foundation, Inc. 2023. URL: https://en.wikipedia.org/wiki/XUnit (besucht am 18. 03. 2024).

[22] Wikipedia. *Klothoide*. Wikimedia Foundation, Inc. 2024. URL: https://de.wikipedia.org/wiki/Klothoide (besucht am 18. 03. 2024).

[23] Frank Witte. *Testmanagement und Softwaretest: theoretische Grundlagen und praktische Umsetzung*. Springer Fachmedien Wiesbaden, 2016.

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Vorname: geb. am: Matr.-Nr.:	Bitte beachten: 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: ……. Unterschrift: ……

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.