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

Einleitung

1

Gestzlichen Bestimmung zur Erhöhung der Sicherheit von sowohl den Insassen eines Fahrzeugs als auch anderer Verkehrsteilnehmer hat zu einer signifikanten Senkung der Mortalitätsrat bei Verkehrsunfällen beigetragen und den Straßenverkehr in den letzten Jahren wesentlich sicherer gemacht, siehe Abbildung 1.1.

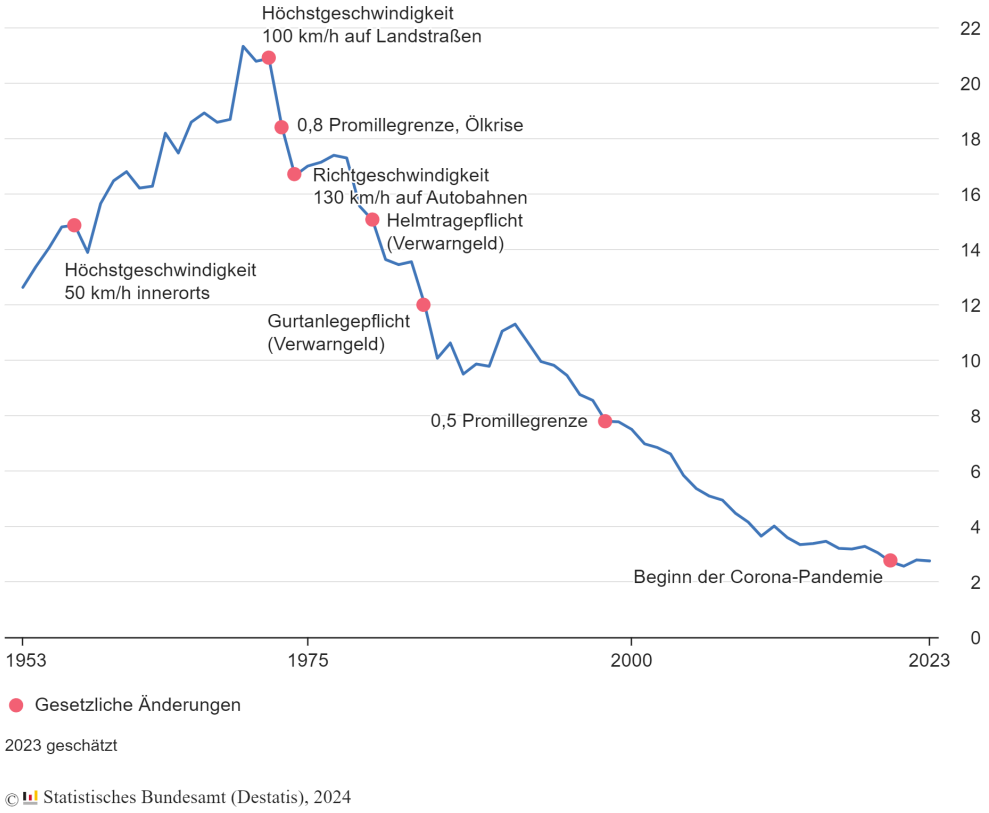


Abbildung 1.1: Bei Verkehrsunfällen Getötete pro Jahr in Tausend [4]

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 "Knickïn den Statistiken 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.

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 Key Performance Indicators zur Beurteilung der Leistung der Regelung
- Einbindung in eine Gitlab CI Pipeline
- Dokumentation der Ergebnisse

1 https://www.iav.com/

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 und so die Qualität der Tests zu steigern. 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

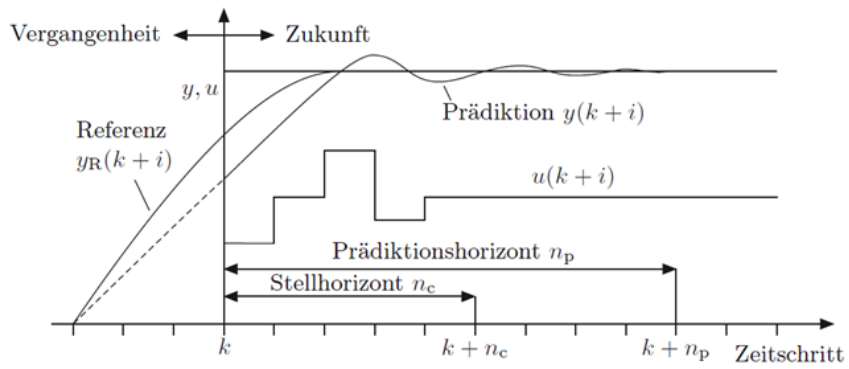


Abbildung 2.1: Ablauf einer modellprädiktiven Regelung [1]

2. Die Stellgrößensequenz u wird berechnet. Dabei wird eine Kostenfunktion minimiert, welche so gewählt wird, dass ein System möglichst gut den gewünschten Eigenschaften entspricht. 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 2.2 dargestellt. Als Eingänge dienen der MPFC eine Sollgeschwindigkeit, Pfaddaten, Fahrzeugzustände, vorgegebene Beschränkungen und ein Modell. Daraus werden eine Sollbeschleunigung und eine Solllenkradwinkelgeschwindigkeit berechnet.

Die Grundidee ist in [7] beschrieben: Der Regelungsalgorithmus soll einem Referenzpfad folgen, es ist jedoch nicht vorgegeben, wann das System an welcher Stelle des Pfades sein muss. Die Geschwindigkeit entlang des Pfades kann also als ein zusätzlicher Freiheitsgrad verwendet werden. Durch eine Modifikation des Ansatzes aus [7] kann eine gewünschte Geschwindigkeit dem Pfad zugeordnet werden, die Fahrzeuggeschwindigkeit wird auf die Pfadgeschwindigkeit konvergieren, die Einhaltung dieser ist allerdings nicht sichergestellt. Ein konvergieren des Fahrzeug auf den Pfad und die Einhaltung von Beschränkungen wird garantiert [16].

Eine Methode zur finden geeigneter Parameter für das Optimierungsproblem der Pfadfolgeregelung ist in [17] 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, wie auch [3] verdeutlicht.

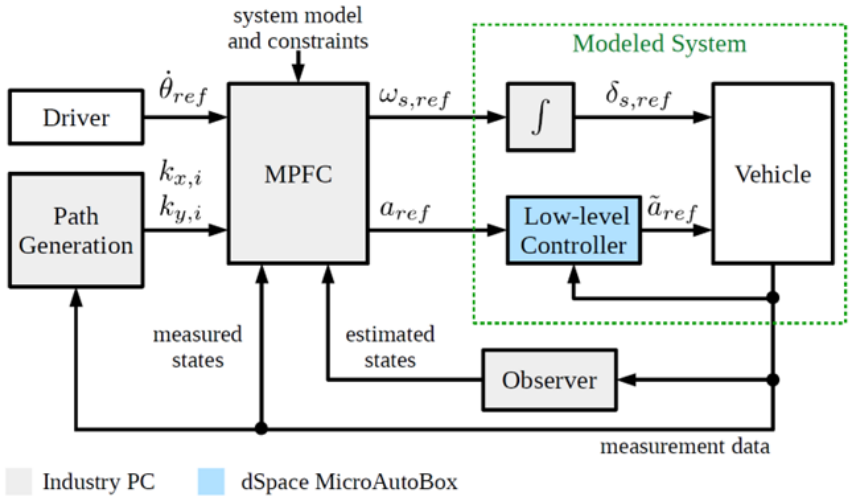


Abbildung 2.2: schematischer Aufbau der MPFC [16]

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].

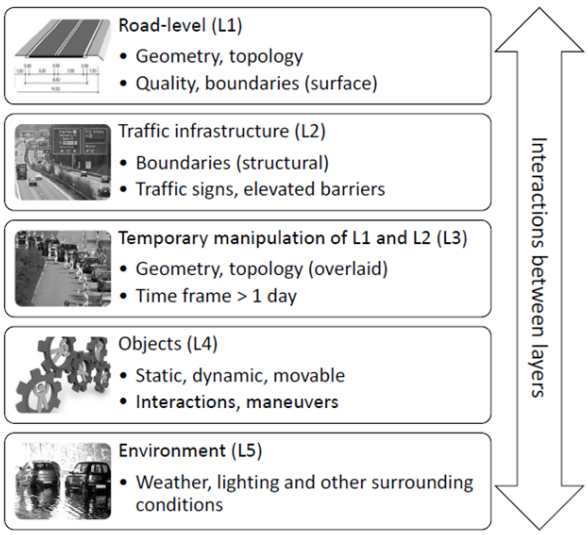


Abbildung 2.3: Einteilung des Informationsgehalts eines Szenarios nach Schichten [2]

Die erste Ebene definiert dabei grundlegende Eigenschaften der Straße, wie etwa deren Beschaffenheit oder Grenzen. Die nächste Ebene Fügt Infrastrukturobjekte wie Stra-

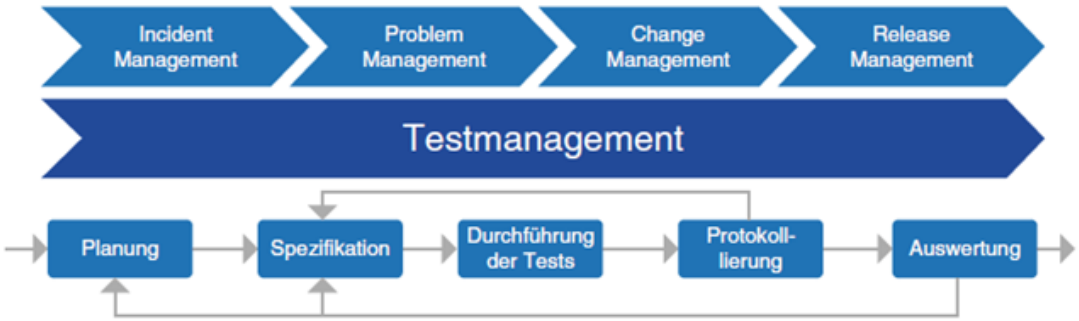


Abbildung 2.5: Ablauf eines Testprozesses [22]

ßenschilder hinzu. Eben 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 Grenze 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 einem funktionalen Szenario entstehen. Selbst bei rein simulativen Tests ist das nicht Umsetzbar und muss reduziert werden [12].

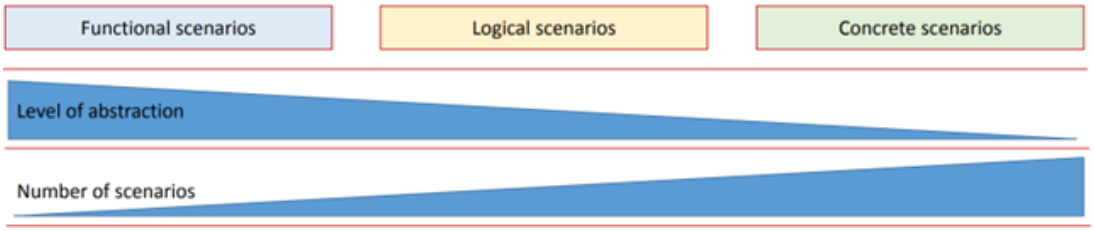


Abbildung 2.4: Erhöhung der Anzahl an Szenarios mit zunehmener Konkretisierung der Szenarien [12]

2.3 Softwartests und CI/CD

Ein geeignetes Testmanagement erhöht die Qualitätvon 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 [22].

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 Versi-

Abbildung 2.6: Ablauf eines CI/CD Prozesses [19]

onsverwaltungssoftware 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 [19].

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.

In nächsten Schritt 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

Der Regelungsalgorithmus und die Simulationsumgebung sind in MATLAB®/Simulink implementiert. Aus den Streckeninformationen wird ein gewünschter Pfad mit einer Zielgeschwindigkeit generiert. Abbildung 3.1 zeigt den generierten Plot, welcher bei Start der Simulation das Fahrzeug auf der Strecke simuliert.

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 Unit Tests zur Verfügung [9]. Eine Verwendung dieser Funktionalität für das Testen der

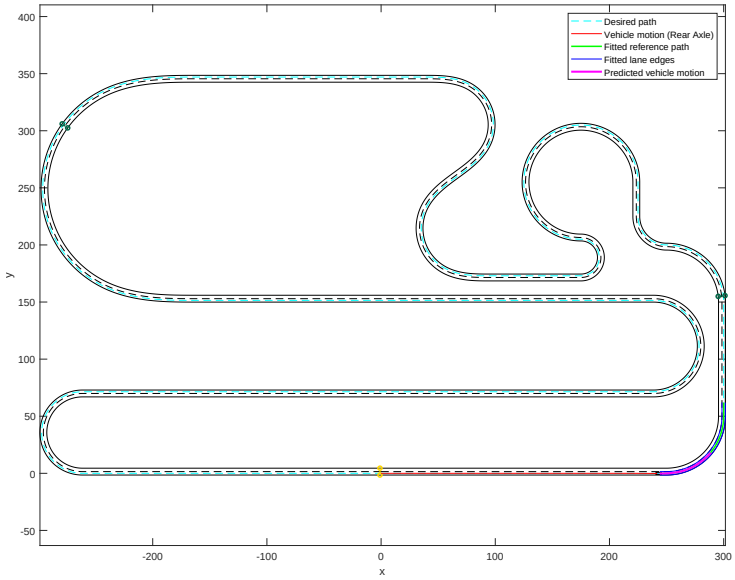


Abbildung 3.1: Beispiel einer komplexen Simulationsstrecke

MPFC liegt nahe, da diese vollständig in Matlab/Simulink implementiert ist. Da es firmenintern noch keine Auseinandersetzung mit diesem Tool gibt wurde sich in der ersten Phase des Praktikums darin eingearbeitet, um die Funktionen zu entdecken und einen möglichst eleganten Weg zu finden die weiteren Aufgaben darin umzusetzen.

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 Programm ist [9].

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* (hier: Testmethode). In dieser 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-Blöcke können Voraussetzungen für mehrere Klassen (shared fixtures), für alle Testmethoden einer Klasse (testclass setup) initial oder vor Ausführung jeder einzelnen Testmethode (testmethod setup) geschaffen werden.

Eine *test suite* ist eine Sammlung von Testmethoden, 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.

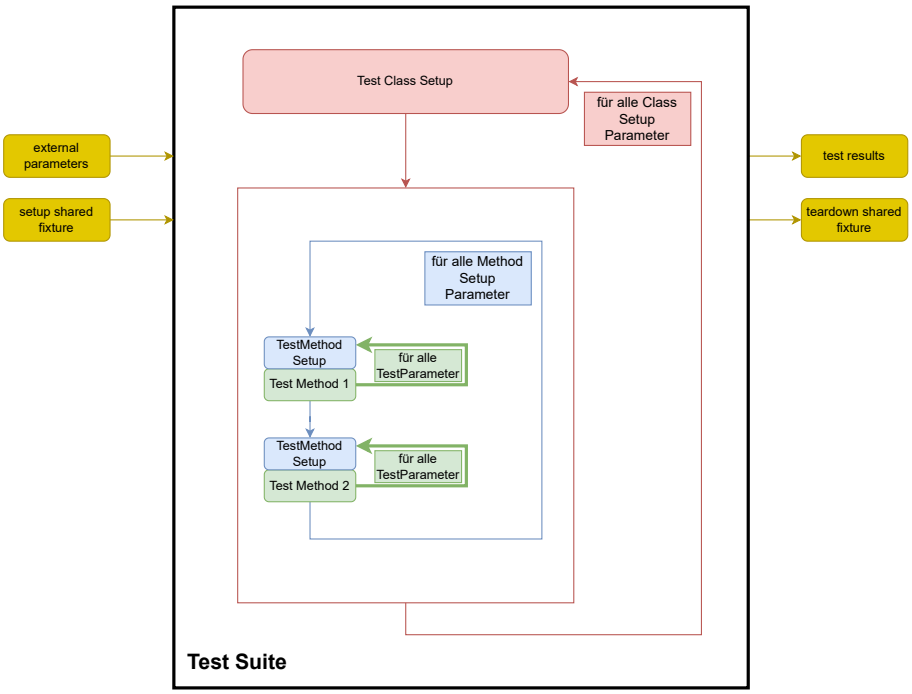


Abbildung 3.2: Ablauf von klassenbasierten Tests

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 Testmethoden und die verschiedenen Setup-Blöcke mehrmals ausgeführt. Eine eigene Darstellung des Ablaufs einer Testklasse ist in Abbildung 3.2 zu sehen.

Ein konfigurierbarer *test runner* führt die Tests aus. Das Framework sorgt selbstständig dafür, dass die notwendigen Setup-Blöce 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 fahrzeugspezifisch sind. 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 MPC selbst ist durch [17] bereits pareto-optimal parametriert. Es stehen allerdings für verschiedene Fahrweisen, von gemütlich bis sportlich, Parametersets zur Verfügung, welches so einen weiteren Simulationsparameter darstellt.

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. Ein Klothoid ist ein Kreisbogen, dessen Krümmung proportional zur Länge ist und somit keinen Sprung in der Krümmung aufweist [20].

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

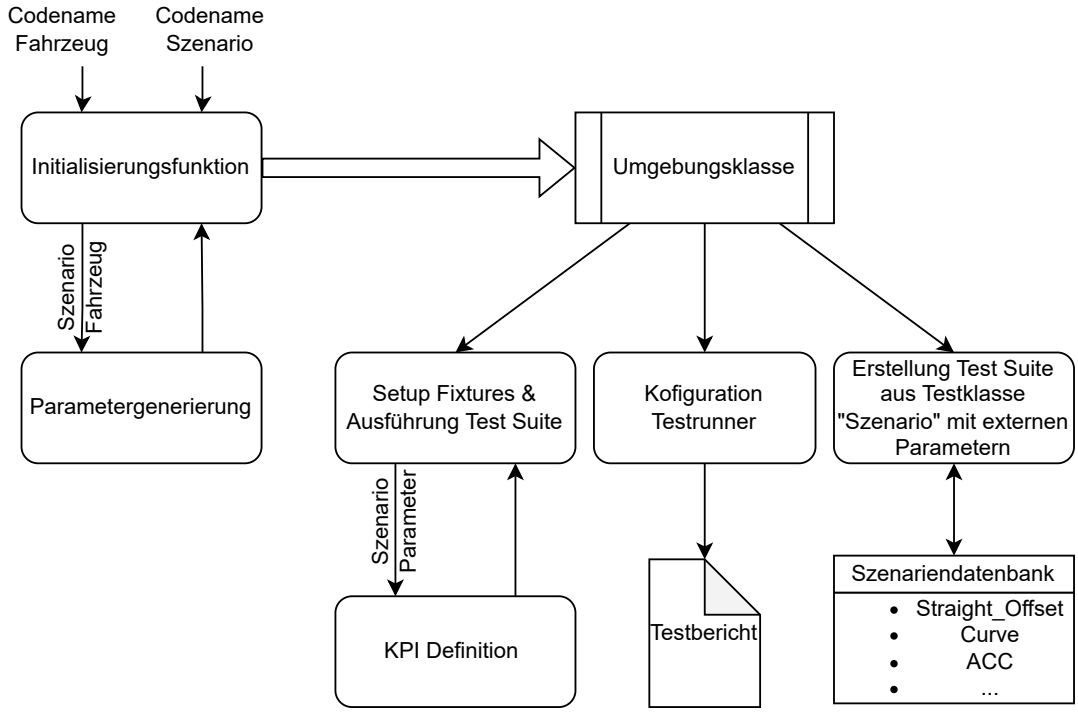


Abbildung 3.3: schematischer Ablauf eines Testskripts zur Ausführung eines Szenarios

lädt, die Simulation ausführt und die Simulationsergebnisse in ihren Testmethoden 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 Key Performance Indicators 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 3.6, generiert.

3.4.3 Parametergenerierung

In bisherigen Testkampagnen 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.

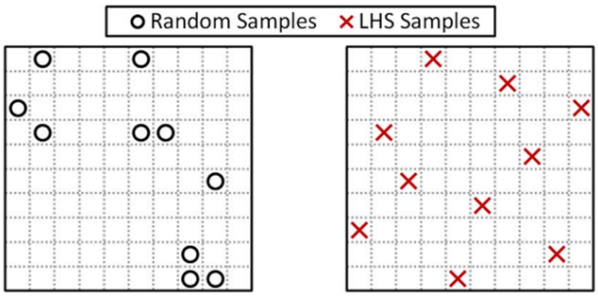


Abbildung 3.4: Vergleich der Verteilung von zufälligen Sampling mit Latin Hypercube Sampling [15]

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 [8] 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.

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. Unter-

Für Fahrzeuge der Klassen M_1 und N_1

Geschwindigkeitsbereich	10-60 km/h	> 60-100 km/h	> 100-130 km/h	> 130 km/h
Höchstwert für die angegebene maximale Querbeschleunigung	3 m/s²	3 m/s²	3 m/s²	3 m/s²
Mindestwert für die angegebene maximale Querbeschleunigung	0 m/s²	0,5 m/s²	0,8 m/s²	0,3 m/s²

Für Fahrzeuge der Klassen M_2, M_3, N_2 und N_3

Geschwindigkeitsbereich	10-30 km/h	> 30-60 km/h	> 60 km/h	
Höchstwert für die angegebene maximale Querbeschleunigung	2,5 m/s²	2,5 m/s²	2,5 m/s²	
Mindestwert für die angegebene maximale Querbeschleunigung	0 m/s²	0,3 m/s²	0,5 m/s²	

Abbildung 3.5: Höchstwerte für laterale Beschleunigung, aufgeschlüsselt nach Fahrzeugklassen und Geschwindigkeitszonen [18]

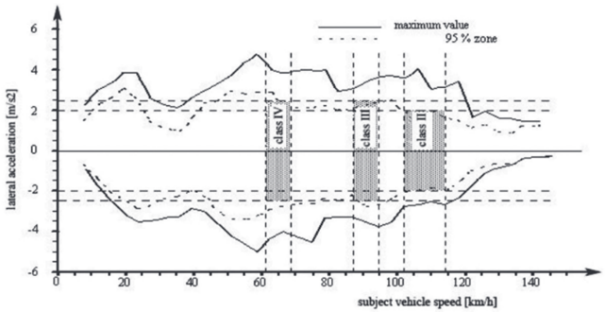


Abbildung 3.6: laterale Beschleunigung in Kurven eines typischen Fahrers [10]

schreiten dieser auf wird ein entsprechender Test als fehlgeschlagen definiert. Diese Kriterien werden als Key Performance Indicators (KPIs) bezeichnet.

Ein KPI kann beispielsweise die Zeit bis zum Abbauen eines initialen lateralen Versatzes sein. Um diese zu ermitteln werden die Pfaddaten und die tatsächlihce Position 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 [18] werden Bedingungen für die Zulassung von Fahrzeugen, welche in die Querführung des Fahrzeugs eingreifen definiert. [10] etabliert Standards für die Funktionen und Testkriteren für ein ACC-System.

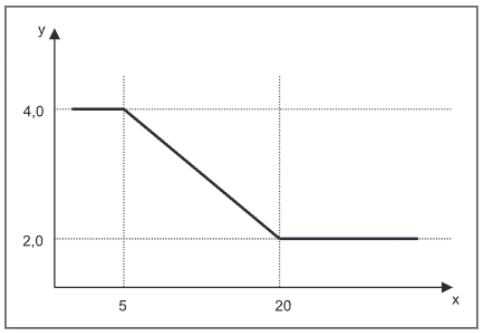
laterale Beschleunigung

Die laterale Beschleunigung hat großen Einfluss auf den Fahrkomfort. In [10] wurde das Verhalten eines durchschnittlichen Fahrers in Kurven bei verschiedenen Geschwindigkeiten analysiert, siehe 3.6. In [18] werden für verschiedene Fahrzeugklassen in verschiedenen Geschwindigkeitszonen maximale Querbeschleunigungen definiert, siehe 3.5.

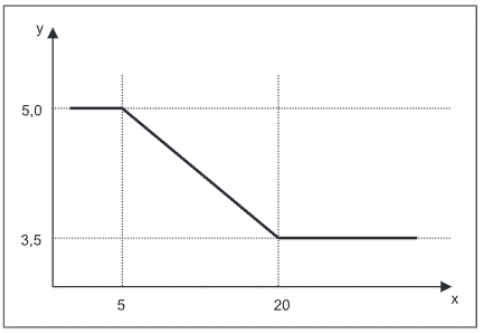
longitudinale Beschleunigung

Diese KPI ist vorallem im Kontext des ACC von großer Relevanz. In [10] sind dafür einige Grenzwerte definiert. Betrachtet werden hier keine Maximalwerte sondern der gleitende Durchschnitt über zwei Sekunden.

Ruck



(a) maximale Beschleunigung (y) $\frac{m}{s^3}$



(b) maximales Abbremsen (y) $\frac{m}{s^3}$

Abbildung 3.7: longitudinale Beschleunigungen bei verschiedenen Geschwindigkeiten (x) $\frac{m}{s}$ [10]

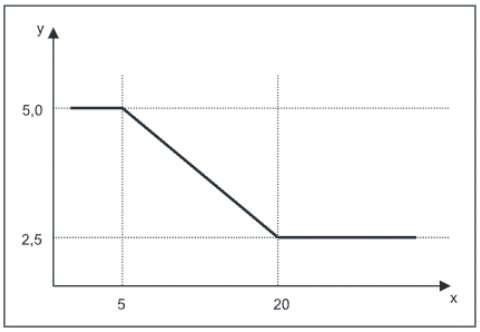


Abbildung 3.8: negativer Ruck (y) $[\frac{m}{s^3}]$ bei verschiedenen Geschwindigkeiten (x) $\frac{m}{s}$ [10]

Ruck ist ein unerwünschtes Verhalten bei Fahrzeugen, das durch ungleichmäßige Beschleunigung und Bremsen entsteht und maßgeblich für den Komfort mitverantwortlich ist. [10] 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 [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 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. In der Testkampagne von Euro NCAP bekommt ein System, welches die Geschwindigkeit automatisch anpassen kann, die maximale Punktzahl, wenn die Geschwnidigkeit auf $\pm 2\frac{km}{h}$ von einer niedrigeren Geschwindigkeitszone angepasst wurde, bevor die Frontachse diese Zone betritt [6]. 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

- **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 Vorstellung der implementierten Szenarien

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

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

In [10] und [6] werden verschiedene Szenarien für die Einhaltung gesetzlicher Vorschriften und ACC Systeme definiert. Aus einer intern zur Verfügung stehenden Szenariendatenbank wurden für die begrenzten Möglichkeiten der Simulationsumgebung (bspw. werden Kreuzungsszenarien zu Kurvenszenarien reduziert, da die Pfadplanung von einem übergeordneten Modul realisiert wird) weitere Szenarien entnommen. Um die Möglichkeiten und Schwierigkeiten der Simulationsumgebung zu analysieren wurden zusätzliche Szenarien implementiert. Dabei wurden auch sog. Negativtests berücksichtigt. 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.

3.5.1 Szenario: Gerade mit initialem Versatz

Beschreibung
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.

gewünschtes Verhalten
Das Ego sollte den initialen Versatz abbauen und mit der Pfadgeschwindigkeit auf dem gewünschten Pfad fahren.

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

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

- *laterale Beschleunigung* $[m \cdot s^{-2}]$: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $a_{Lat} < kpi_{a_{Lat}}$
- *longitudinale Beschleunigung* $[m \cdot s^{-2}]$: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $a_{Long} < kpi_{a_{Long}}$
- *lateraler Ruck* $[m \cdot s^{-3}]$: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $j_{Lat} < kpi_{j_{Lat}}$
- *longitudinaler Ruck* $[m \cdot s^{-3}]$: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $j_{Long} < kpi_{j_{Long}}$
- *Einschwingzeit* $[s]$: der laterale Versatz sollte innerhalb einer Zeit abgebaut werden: $t_{settle} < kpi_{t_{settle}}$
- *longitudinale Geschwindigkeit* $[m \cdot s^{-1}]$: die Geschwindigkeitsabweichung zum Pfad darf den Grenzwert nicht überschreiten: $|v_{Long} - v_{Path}| < kpi_{v_{diff}}$

3.5.2 Szenario: ACC mit konstanter Objektgeschwindigkeit

Beschreibung
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.

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

gewünschtes Verhalten
Das Ego sollte die Geschwindigkeitsdifferenz beseitigen und die gewünschte Zeitlücke herstellen.

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

- *longitudinale Beschleunigung* $[m \cdot s^{-2}]$: der Gleitende Mittelwert über eine halbe Sekunde darf einen definierten Grenzwert nicht überschreiten: $a_{Long} < kpi_{a_{Long}}$
- *longitudinaler Ruck* $[m \cdot s^{-3}]$: der Gleitende Mittelwert über eine halbe Sekunde darf den Grenzwert nicht überschreiten: $j_{Long} < kpi_{j_{Long}}$
- *Einschwingzeit* $[s]$: der gewünschte Abstand zum Target sollte innerhalb einer definierten Zeit eingestellt werden: $t_{settle} < kpi_{t_{settle}}$
- *Reaktionszeit* $[s]$: der Minimalabstand zum Target soll innerhalb einer definierten Zeit hergestellt werden: $t_{reaction} < kpi_{t_{reaction}}$

3.5.3 Szenario: Kurve negativ

Beschreibung
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.

Parametrierung
Die Parameter für dieses Szenario sind:

- Initialgeschwindigkeit $[km \cdot h^{-1}]$: $v_{Ego} \in [0.2 \cdot v_{vehicle,max}, 0.8 \cdot v_{vehicle,max}]$
- Kurvenradius $[m]$: $s_{radius} \in [3, r_{min,turn}]$
- verwendetes MPC Parameterset

gewünschtes Verhalten
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.

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

- *laterale Ablage* $[m]$: die Abweichung des Fahrzeugs vom gewünschten Pfad muss größer sein als ein definierter Grenzwert: $s_{error,lat} > kpi_{s_{error,lat}}$

3.6 Integration in die GitLab CI Pipeline

Für das automatisierte Testen in der Pipeline wird eine .yaml-Datei benötigt. 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 beschriebem. Es wird lediglich ein anderes Initialisierungsskript verwendet, um mit den geänderten Gegebenheiten in der Pipeline umzugehen.

3.6.1 Darstellung der Pipeline

Um die Wartung der 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 das *matrix* Kennwort die gewünschten Szenarien, welche für dieses Fahrzeug abgetestet werden sollen, hinterlegt. Die Pipeline generiert dann automatisch für jede Kombination aus Fahrzeug und Szenario einen eigenen Job an. Durch geeignete Wahl der Kennwörter der Testklasse ist es zusätzlich möglich in einem Job mehrere Testklassen/Szenarien nacheinander laufen zu lassen. Beispielsweise wird im Job "test_Curveßowohl ein normales Kurvenfahrtszenario als auch der in Abschnitt 3.5.3 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 3.9). Aus dieser ist direkt zu entnehmen, welche Szenarios für welches Fahrzeug (Alice, IAVShuttle) ausgeführt wurden und welche Szenarien zu Fehlern geführt haben.

3.6.2 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 3.10). Ist ein Test bestanden gibt es keine weiteren Informationen dazu (Abbildung 3.11). Schlägt ein Test fehl gibt es einen ausführlicheren Bericht (Abbildung 3.12). 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

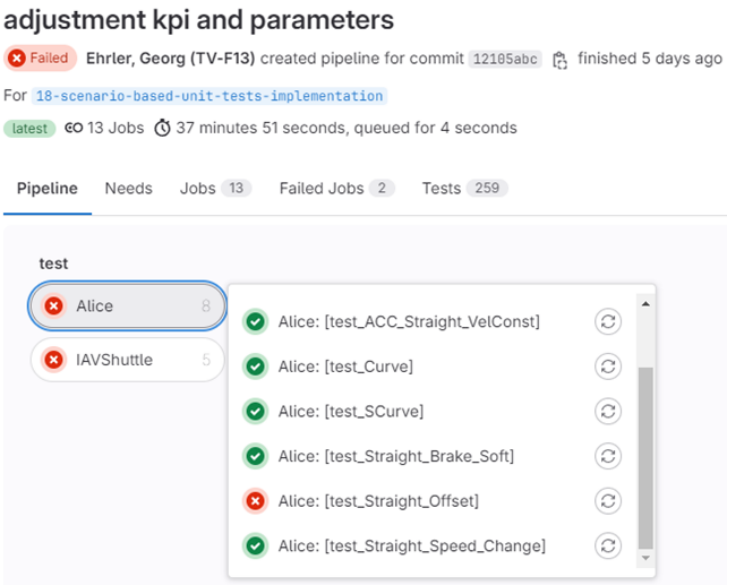


Abbildung 3.9: Darstelllung der Jobs in der Gitlab CI Pipeline im Browser

Parameter des aktuellen Szenarios werden ebenfalls angegeben, um den Test lokal reproduzieren zu können.

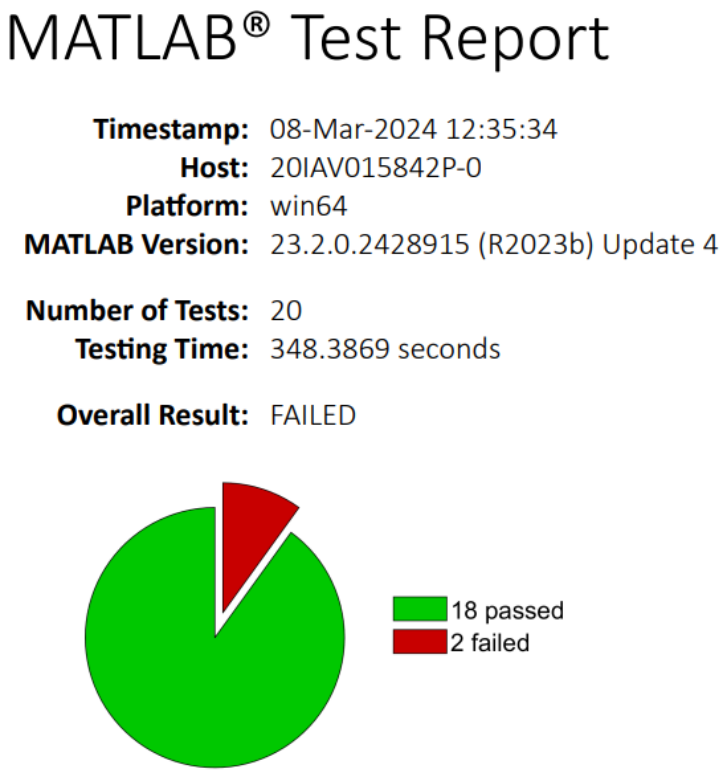


Abbildung 3.10: Übersichtliche Darstellung der Testergebnisse eines Szenarios auf der ersten Seite des Testberichts

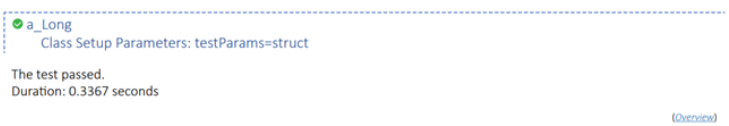


Abbildung 3.11: Darstellung eines erfolgreichen Test im Testbericht

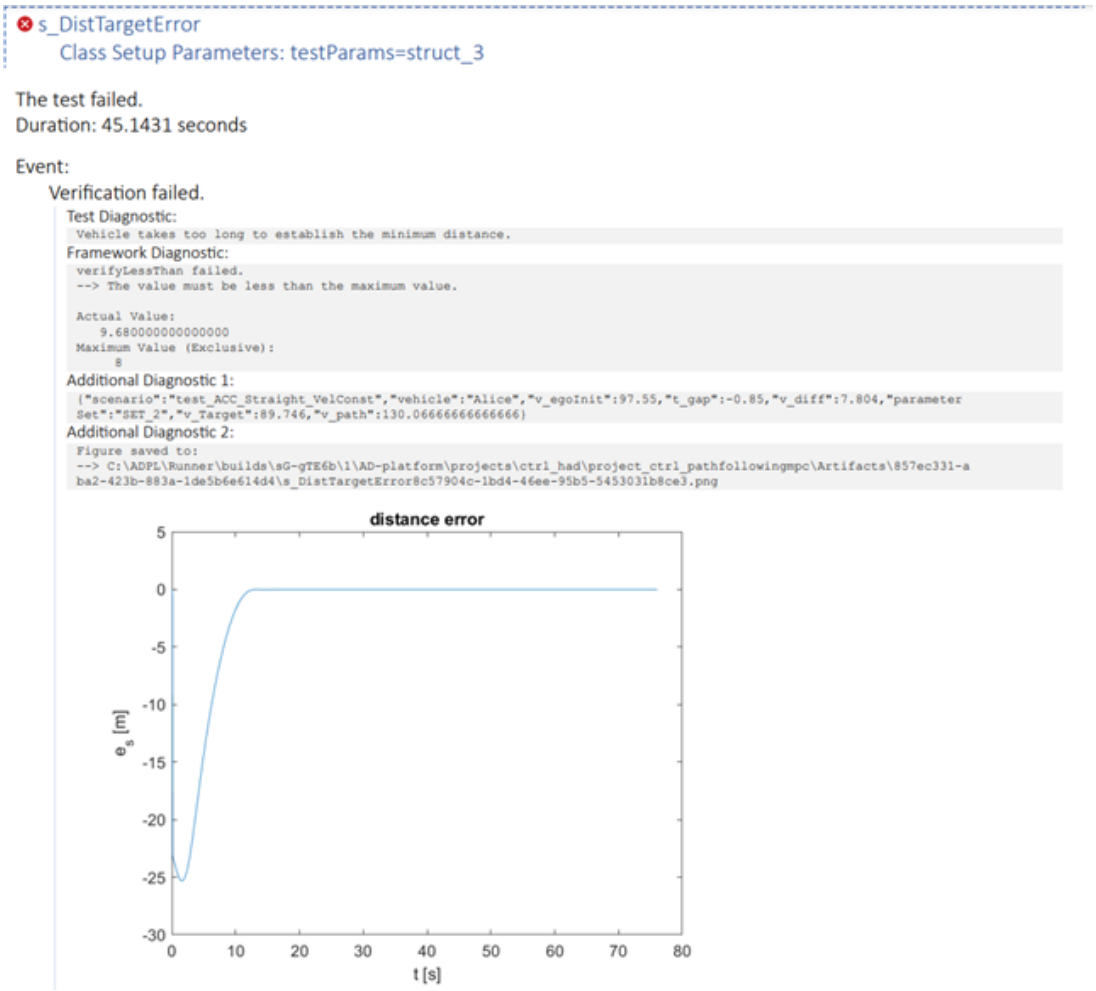


Abbildung 3.12: Darstellung eines fehlgeschlagenen Tests im Testbericht. Der Wert der KPI und der tatsächliche Wert werden verglichen. Ein kurzer Text gibt dem Entwickler wieder, was fehlgeschlagen ist. Wenn sinnvoll wird ein Plot mit den relevanten Daten erzeugt. Die Angabe der Simulationsparameter genutzt werden um diese eine Simulation lokal zu wiederholen.

4 Zusammenfassung und Ausblick

Nachdem in der Einleitung die Einordnung und Vorstellung des Themas erfolgt 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 einige 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. engere Kurven sind allerdings deutlich kritischer als weitere 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 der 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*. Springer Berlin Heidelberg, 2014. DOI: https://doi.org/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: https://doi.org/10.1016/j.trf.2018.02.036. URL: https://www.sciencedirect.com/science/article/pii/S1369847817301535.

[4] Statistisches Bundesamt. „Verkehrsunfälle“. In: *Fachserie 8 Reihe 7* (2021). URL: https://www.destatis.de/DE/Home/_inhalt.html.

[5] E.F. Camacho und C.B. Alba. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2013. ISBN: 9780857293985. URL: https://books.google.de/books?id=tXZDAAAAQBAJ.

[6] Euro NCAP. *Assisted Driving Highway and Interurban Assist Systems*. Euro NCAP. Feb. 2018.

[7] T. Faulwasser, B. Kern und R. Findeisen. „Model predictive path-following for constrained nonlinear systems“. In: *Proceedings of the 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.

[8] Bauformel Verlag GmbH. Jan. 2019. URL: https://www.bauformeln.de/strassenbau/grenz-und-richtwerte.

[9] The MathWorks Inc. *Matlab Unit Test Framework*. Natick, Massachusetts, United States, 2022. URL: https://de.mathworks.com/help/matlab/matlab-unit-test-framework.html.

[10] *Intelligent transport systems — Adaptive cruise control systems — Performance requirements and test procedures*. ISO 15622. Sep. 2018.

[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. URL: http://www.jstor.org/stable/1268522 (besucht am 12. 03. 2024).

[12] Till Menzel, Gerrit Bagschik und Markus Maurer. *Scenarios for Development, Test and Validation of Automated Vehicles*. 2018. arXiv: 1801.08598 [cs.SE].

[13] Gerard Meszaros. *xUnit Patterns*. 2011. URL: http://xunitpatterns.com/index.html.

[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: Okt. 2020. DOI: 10.46720/f2020-acm-096.

[15] 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.

[16] 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: https://doi.org/10.1016/j.ifacol.2019.08.112. URL: https://www.sciencedirect.com/science/article/pii/S240589631930446X.

[17] 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. URL: https://www.mdpi.com/2227-7390/11/2/465.

[18] *Einheitliche Bedingungen für die Genehmigung der Fahrzeuge hinsichtlich der Lenkanlage*. UNECE R 79. Okt. 2018.

[19] *Was ist CI/CD? Konzepte und CI/CD Tools im Überblick*. März 2024. URL: https://www.redhat.com/de/topics/devops/what-is-ci-cd.

[20] Wikimedia Foundation Wikipedia. *Klothoide*. URL: https://de.wikipedia.org/wiki/Klothoide.

[21] Wikimedia Foundation Wikipedia. *xUnit*. URL: https://en.wikipedia.org/wiki/XUnit.

[22] Frank Witte. *Testmanagement und Softwaretest: theoretische Grundlagen und praktische Umsetzung*. Springer, 2019.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Vorname: geb. am: Matr.-Nr.:	<u>Bitte beachten:</u> 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.