



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: 15. September 2024



IAV GmbH



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

INHALTSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS	III
ABBILDUNGSVERZEICHNIS	V
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	6
2.3 Softwareentwicklungsprozesse und Unit-Tests	7
3 IMPLEMENTIERUNG	9
3.1 Anforderungen	9
3.2 Simulationsumgebung	9
3.3 MATLAB® Unit-Test Framework	10
3.4 Aufbau und Ablauf eines Szenarios	11
3.4.1 Simulationsparameter	11
3.4.2 Ablauf eines Testskripts	12
3.4.3 Parametergenerierung	12
3.4.4 Definition von Key Performance Indicators und Testkriterien	14
3.5 Integration in die GitLab CI Pipeline	17
3.5.1 Darstellung der Pipeline	17
3.5.2 Darstellung der Testergebnisse	18
4 VORSTELLUNG AUSGEWÄHLTER SZENARIEN	21
4.1 Szenario: Gerade mit initialem Versatz	21
4.1.1 Parametrierung	21
4.1.2 Abgetestete KPIs	22
4.2 Szenario: Kurve negativ	24
4.2.1 Parametrierung	24
4.2.2 abgetestete KPIs	24
4.3 Szenario: ACC mit konstanter Objektgeschwindigkeit	25
4.3.1 Parametrierung	25
4.3.2 Abgetestete KPIs	26
5 ZUSAMMENFASSUNG UND AUSBLICK	29
LITERATUR	31

ABKÜRZUNGSVERZEICHNIS

MPC	Model Predictive Control, modellprädiktive Regelung
MPFC	Model Predictive Path Following Control, modellprädiktive Pfadfolgeregelung
ACC	Adaptive Cruise Control, Abstandsregeltempomat
KPI	Key Performance Indicator, Leistungskennzahl
OCP	Optimal Control Problem, Optimalsteuerungsproblem
CI	Continuous Integration, kontinuierliche Integration
CD	Continuous Deployment, kontinuierliche Bereitstellung
SUT	System Under Test, getestetes System
LHS	Latin Hypercube Sampling

ABBILDUNGSVERZEICHNIS

Abbildung 1.1	Bei Verkehrsunfällen Getötete pro Jahr in Tausend [4]	1
Abbildung 2.1	Ablauf einer modellprädiktiven Regelung [1]	4
Abbildung 2.2	schematischer Aufbau der MPFC [19]	4
Abbildung 2.3	Einteilung eines Szenarios nach Informationsgehalt [2]	7
Abbildung 2.4	Einteilung eines Szenarios nach Abstraktionsgrad [13]	7
Abbildung 2.5	Ablauf eines CI/CD Prozesses [8]	8
Abbildung 3.1	Beispiel einer komplexen Simulationsstrecke	10
Abbildung 3.2	schematischer Ablauf eines Testskripts zur Ausführung eines Szenarios	13
Abbildung 3.3	Vergleich von zufälligem Sampling mit Latin Hypercube Sampling [18]	14
Abbildung 3.4	laterale Beschleunigung in Kurven eines typischen Fahrers [20]	15
Abbildung 3.5	maximal zulässige longitudinale Beschleunigung, Durchschnitt über 2 s [20]	16
Abbildung 3.6	maximal zulässiger negativer Ruck, Durchschnitt über 1 s [20]	16
Abbildung 3.7	Darstellung der Jobs in der GitLab CI Pipeline	18
Abbildung 3.8	Übersichtliche Darstellung der Testergebnisse eines Szenarios	19
Abbildung 3.9	Darstellung eines fehlgeschlagenen Tests im Testbericht . . .	19
Abbildung 4.1	Darstellung des Szenarios: Gerade mit initialem Versatz	22
Abbildung 4.2	KPI Einschwingzeit abhängig der Simulationsparameter	22
Abbildung 4.3	KPI Geschwindigkeitsabweichung abhängig der Simulationsparameter	23
Abbildung 4.4	KPI laterale Beschleunigung abhängig der Simulationsparameter	23
Abbildung 4.5	KPI lateraler Ruck abhängig der Simulationsparameter	23
Abbildung 4.6	Darstellung des Szenarios: Kurvenfahrt negativ	24
Abbildung 4.7	KPI Pfadabweichung bei verschiedenen Kurvenradien	25
Abbildung 4.8	Darstellung des Szenarios: ACC mit konstanter Objektgeschwindigkeit	25
Abbildung 4.9	Simulationsdaten für longitudinale Beschleunigung und Ruck bei unterschiedlichen Parametrierungen	26
Abbildung 4.10	Simulationsdaten für Abstands- und Distanzfehler bei unterschiedlichen Parametrierungen	27

EINLEITUNG

Gesetzliche 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 Mortalitätsrate bei Verkehrsunfällen beigetragen.

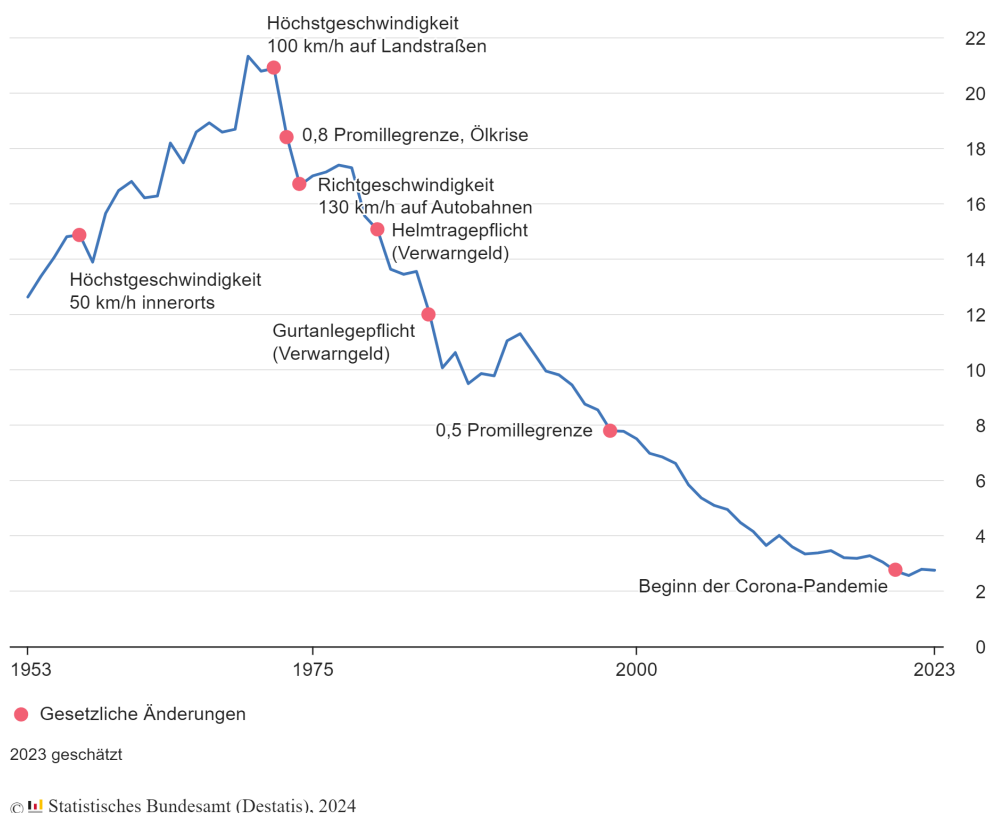


Abbildung 1.1: Bei Verkehrsunfällen Getötete pro Jahr in Tausend [4]

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 Assistenzsystemen unterstützt oder die menschliche Fehlerkomponente gänzlich beseitigt, indem ein Fahrzeug vollständig autonom fährt.

Um die Sicherheit der Fahrzeuge zu gewährleisten, muss sowohl das Gesamtsystem, als auch alle Teilsysteme intensiv getestet werden. Eines dieser Teilsysteme für das hochautomatisierte Fahren ist die Regelung der Längs- und Querverführung des Fahrzeugs. Bei der IAV¹ geschieht dies durch einen modellprädiktiven Pfadfolgeregler.

1 <https://www.iav.com/>

Aufgabenstellung

Im Rahmen des Praktikums soll eine automatisierte Teststrategie für einen modellprädiktiven Pfadfolgeregler (engl. Model Predictive Pathfollowing Control, MPFC) konzipiert und implementiert werden.

Modellprädiktive Regler (engl. Model Predictive Control, MPC) eignen sich hervorragend für die Vorhersage und Regelung dynamischer Systeme, da sie auf Störungen im Betrieb gut reagieren können, weshalb sie in vielen Industriezweigen, einschließlich der Automobilindustrie, Anwendung finden [1]. 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

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 Softwareentwicklungsabläufe gegeben.

2.1 Modellprädiktive Pfadfolgeregelung

2.1.1 Modellprädiktive Regelung

Die modellprädiktive Regelung ist ein Verfahren zur Regelung von dynamischen Systemen. Dabei wird ein mathematisches Modell des Systems erstellt. Durch das messen und beobachten des Ausgangszustandes können über die Gleichung 2.1 die zukünftigen Systemzustände x und Ausgänge y innerhalb des Prädiktionshorizonts n_p vorhergesagt werden. 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.

$$\begin{aligned}x(k+1) &= f(x(k), u(k)) \\ y(k) &= g(x(k), u(k))\end{aligned}\tag{2.1}$$

Der Ablauf einer modellprädiktiven Regelung wie in Abbildung 2.1 kann in drei Punkte unterteilt werden [5]:

1. Mithilfe eines Modells wird der Ausgang eines Systems $y(k+i)$ für die nächsten $i = 1, \dots, n_p$ Zeitschritte vorausgesagt. Grundlage für die Prädiktion sind die aktuellen Systemzustände und -eingänge und die zukünftigen Eingänge, welche im nächsten Schritt berechnet werden.
2. Innerhalb des Stellhorizonts n_c , wobei $n_c \leq n_p$, wird die Stellgrößensequenz u berechnet. Dabei wird eine Kostenfunktion, beispielsweise Gleichung 2.7, so gewählt, dass sich ein gewünschtes Systemverhalten einstellt. Häufig wird dabei ein quadratisches Gütemaß verwendet. 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.

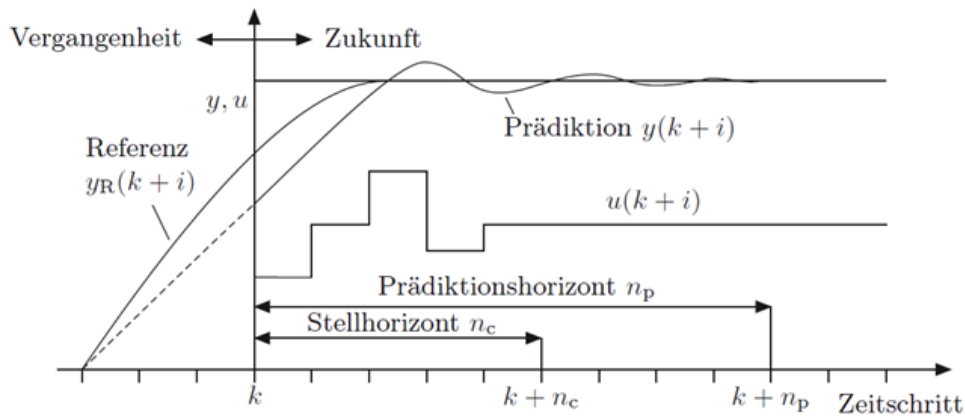


Abbildung 2.1: Ablauf einer modellprädiktiven Regelung [1]

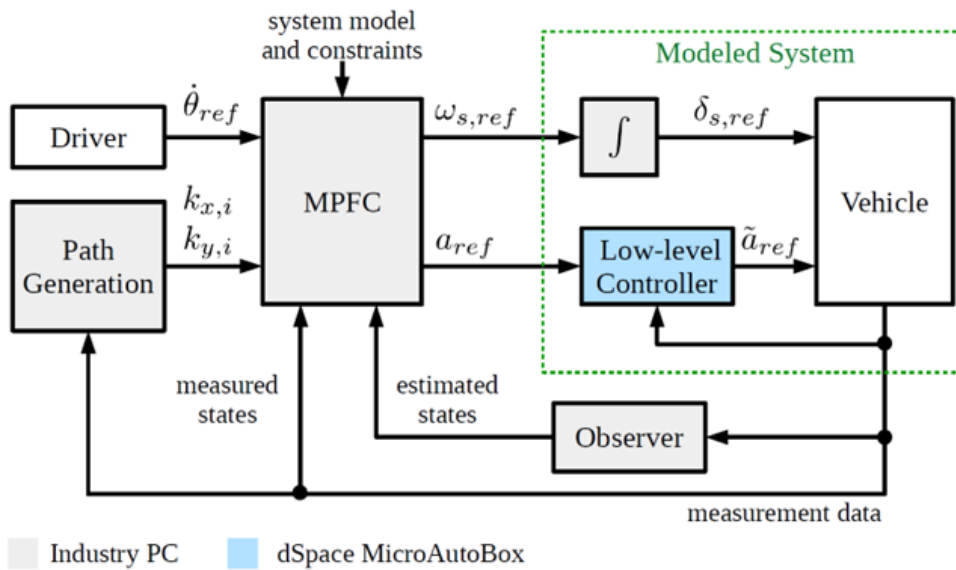


Abbildung 2.2: schematischer Aufbau der MPFC [19]

2.1.2 Pfadfolgeregelung

Die Struktur der modellprädiktiven Pfadfolgeregelung, für die in diesem Praktikum eine Teststrategie implementiert werden soll, ist in Abbildung 2.2 dargestellt und in [19] dokumentiert. Als Eingänge dienen der MPFC eine vorgegebene Sollgeschwindigkeit, Pfaddaten, Fahrzeugzustände, vorgegebene Beschränkungen und ein Fahrzeugmodell. Als Ausgang werden Sollbeschleunigung und Solllenkradwinkelgeschwindigkeit berechnet. Die Grundidee der Pfadfolgeregelung ist in [6] beschrieben. Der Regler soll einem vorgegebenen Pfad möglichst gut folgen. Im Gegensatz zu trajektorienbasierten Ansätzen ist dabei nicht vorgegeben, wann das System an welcher Stelle des Pfades sein soll. Die Regelung der Längsgeschwindigkeit ist daher zusätzlich erforderlich.

Betrachtet wird das zeitkontinuierliche, nichtlineare System der Form

$$\dot{x}(t) = f(x(t), u(t)), x(t_0) = x_0 \quad (2.2)$$

$$y(t) = h(x(t)) \quad (2.3)$$

mit den Zuständen $x \in \mathcal{X} \subseteq \mathbb{R}^{N_x}$, Eingängen $u \in \mathcal{U} \subseteq \mathbb{R}^{N_u}$ und dem Ausgang $y \in \mathcal{Y} \subseteq \mathbb{R}^{N_y}$. Die geometrische Kurve $\mathcal{P} : \{p(\theta) \in \mathbb{R}^{N_y} | \theta \in [\theta_0, \theta_1] \mapsto p(\theta)\}$ ist als Referenzpfad gegeben und wird durch $p(\theta)$ parametrisiert.

$$p(\theta) = \begin{bmatrix} x_{\text{ref}}(\theta) \\ y_{\text{ref}}(\theta) \\ \psi_{\text{ref}}(\theta) \end{bmatrix} \quad (2.4)$$

Der Pfadparameter θ gibt die Position auf dem Referenzpfad an. Dieser wird als ein virtueller Zustand von einem zusätzlichen Eingang, der Geschwindigkeit $\vartheta \in \mathcal{V} \subseteq \mathbb{R}$, beeinflusst. Dadurch ist eine Geschwindigkeitszuweisung möglich.

$$\dot{\theta}(t) = \vartheta(t) \quad (2.5)$$

Es kann nun ein Regler für das Lösen des Pfadfolgeproblems mit Geschwindigkeitszuweisung entworfen werden, der folgende Eigenschaften besitzt:

1. Der Ausgang des Systems y konvergiert auf den Pfad \mathcal{P} : $\lim_{t \rightarrow \infty} \|e(t)\| = 0$ mit der Pfadabweichung $e(t) := h(x(t)) - p(\theta(t))$.
2. Die Pfadgeschwindigkeit konvergiert auf die Referenzgeschwindigkeit: $\lim_{t \rightarrow \infty} \|\dot{\theta}(t) - \dot{\theta}_{\text{ref}}(t)\| = 0$
3. Die Zustandsbeschränkungen \mathcal{X} und Eingangsgrößenbeschränkungen \mathcal{U} werden eingehalten.

Das OCP, welches zu jedem diskreten Zeitpunkt $t_k = kT_s$ mit der Samplingzeit T_s gelöst werden soll, ist folgendes:

$$J(x(t_k), \theta(t_k), \bar{u}^*(\cdot), \bar{\vartheta}^*(\cdot)) = \min_{u(\cdot), \vartheta(\cdot)} J(x(t_k), \theta(t_k), \bar{u}(\cdot), \bar{\vartheta}(\cdot)) \quad (2.6a)$$

$$\text{s.t.} \quad \dot{\bar{x}}(\tau) = f(\bar{x}(\tau), \bar{u}(\tau)), \quad \bar{x}(t_k) = x(t_k) \quad (2.6b)$$

$$\dot{\bar{\theta}}(\tau) = \bar{\vartheta}(\tau), \quad \bar{\theta}(t_k) = \theta(t_k) \quad (2.6c)$$

$$\bar{e}(\tau) = h(\bar{x}(\tau)) - p(\bar{\theta}(\tau)) \quad (2.6d)$$

$$\bar{u}(\tau) \in \mathcal{U}, \quad \bar{x}(\tau) \in \mathcal{X} \quad (2.6e)$$

$$\bar{\theta}(\tau) \in [0, \theta_{\max}], \quad \bar{\vartheta}(\tau) \in \mathcal{V} \quad (2.6f)$$

$$h_c(\bar{x}(\tau), \bar{u}(\tau)) \leq 0 \quad (2.6g)$$

Die zu minimierende Kostenfunktion ist gegeben mit:

$$J(x(t_k), \theta(t_k), \bar{u}(\cdot), \bar{\vartheta}(\cdot)) = \int_{t_k}^{t_k + T_p} F(\bar{e}(\tau), \bar{x}(\tau), \bar{u}(\tau), \bar{\vartheta}(\tau)) d\tau + E(\bar{e}(t_k + T_p), \bar{x}(t_k + T_p)) \quad (2.7)$$

Prädizierte Systemzustände und Eingänge werden mit $\bar{x}(\cdot)$ und $\bar{u}(\cdot)$, optimale Eingangsgrößen mit $\bar{u}^*(\cdot)$, $\bar{\vartheta}^*(\cdot)$ bezeichnet. Die Systemdynamik wird durch die Gleichungen

2.6b und 2.6c repräsentiert. Die Abweichung des Systems vom Pfad geht durch Gleichung 2.6d in das Optimierungsproblem ein. Die Gleichungen 2.6e und 2.6f geben Beschränkungen für Zustände und Eingangsgrößen vor. Geschwindigkeitsabhängige Beschränkungen der Lenkaktorik werden mit Gleichung 2.6g berücksichtigt. Die Wegkosten $F(\cdot)$ und die Endkosten $E(\cdot)$ sind mit

$$F(e, x, u, \vartheta) = \left\| \begin{matrix} e \\ a_{lat}(x) \end{matrix} \right\|_Q^2 + \left\| \begin{matrix} u \\ \vartheta - \vartheta_{ref} \end{matrix} \right\|_R^2 \quad (2.8)$$

$$E(e, x) = \left\| \begin{matrix} e \\ a_{lat}(x) \end{matrix} \right\|_P^2 \quad (2.9)$$

gegeben. Die Kostenfunktionen werden durch die Diagonalmatrizen Q , R und P gewichtet:

$$\begin{aligned} Q &= \text{diag}(q_x, q_y, q_\psi, q_a), \\ P &= \text{diag}(p_x, p_y, p_\psi, p_a), \\ R &= \text{diag}(r_a, r_\omega, r_v), \end{aligned} \quad (2.10)$$

Die Qualität der Regelung hängt maßgeblich von einer geeigneten Wahl der Parameter in diesen Matrizen ab. Eine Methode zur Ermittlung geeigneter Parameterwerte ist in [21] dokumentiert. Durch eine bayes'sche Optimierung wurden Parameterwerte für eine möglichst optimale Verfolgung des Pfads und der Pfadgeschwindigkeit bei gleichzeitiger Minimierung der lateralen und longitudinalen Beschleunigungen ermittelt. Letzteres hat einen großen Einfluss auf die Wahrnehmung des Fahrkomforts [3].

2.2 Szenariobasiertes Testen im Automobilbereich

Um ein autonomes Fahrsystem zu realisieren und die 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 Fahrsituationen auftreten. Es bietet sich daher an, in einer Datenbank die kritischen Szenarien zu erfassen und für spätere Verifikationsprozesse wieder heranzuziehen [16].

Um Szenarien zu beschreiben und in eine Datenbank einordnen zu können, müssen diese klassifiziert werden. Eine Möglichkeit dafür ist die Einteilung nach Informationslevel (Abbildung 2.3). Die erste Ebene definiert dabei grundlegende Eigenschaften der Straße, wie etwa deren Beschaffenheit oder Grenzen. Die nächste Ebene fügt Infrastruktureobjekte 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, 16].

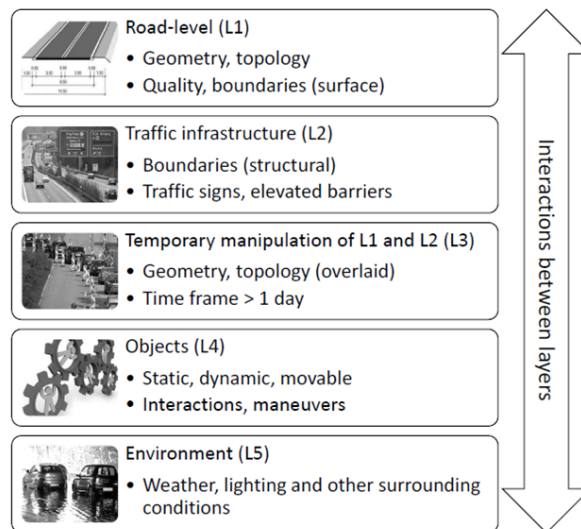


Abbildung 2.3: Einteilung eines Szenarios nach Informationsgehalt [2]

Eine weitere Klassifizierungsmöglichkeit unterscheidet die Szenarien nach dem Abstraktionsgrad (Abbildung 2.4). Zunächst werden funktionale Szenarien beschrieben. Diese können einfach mit Worten beschrieben werden und enthalten noch keine Information über die Parameter. In einer ersten Konkretisierung werden diese zu logischen Szenarien. 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 [13, 16].

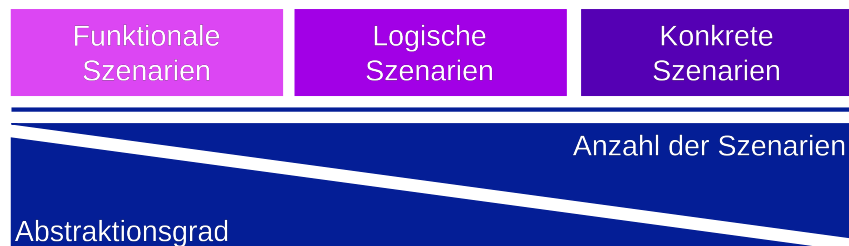


Abbildung 2.4: Einteilung eines Szenarios nach Abstraktionsgrad [13]

2.3 Softwareentwicklungsprozesse und Unit-Tests

Ein geeignetes Testmanagement erhöht die Qualität einer Software maßgeblich, indem Fehler bereits im Entwicklungsprozess entdeckt und behoben werden können. Eine Möglichkeit Testprozesse durchzuführen steckt hinter den Schlagwörtern Continuous Integration (CI) und Continuous Deployment (CD). Darunter versteht sich ein Prozess zum Entwickeln, Testen und Freigeben von Software unter Verwendung einer Versionsverwaltungssoftware, wie er in Abbildung 2.5 dargestellt ist. Wenn in einem Modul einer größeren Softwareplattform Änderungen vorgenommen werden, geschieht dies in der Regel auf einer lokalen Kopie der Gesamtsoftware. Bevor diese Änderungen in das

geteilte Repository für alle Entwickler übernommen werden, läuft ein automatisierter Testprozess ab (Continuous Integration). Dieser ist typischerweise in Stages unterteilt, beispielsweise build, test und merge. Dabei wird sowohl die Funktionalität eines Moduls und der Teilfunktionen (Unit-Tests), als auch das Zusammenspiel der Module untereinander (Integration-Test) überprüft. Es kann dadurch sichergestellt werden, dass die Software auch mit den Änderungen noch korrekt funktioniert. Treten in diesen Prozessschritten keine Fehler auf, werden die Änderungen übernommen und damit die Software auf den aktuellen Stand gebracht (Continuous Delivery). In einem weiteren Schritt kann die aktualisierte Software automatisiert an den Kunden ausgeliefert werden (Continuous Deployment). [8]. Die Implementierung und Ausführung dieser Schritte wird häufig als *Pipeline* bezeichnet [14].



Abbildung 2.5: Ablauf eines CI/CD Prozesses [8]

Der kleinste Baustein in diesem Testprozess ist dabei der *Unit-Test*, welcher im klassischen Sinne für die Überprüfung einer einzelnen Einheit (engl. unit) einer Software, beispielsweise einer Klasse oder einer einzelnen Funktion, zuständig ist. Dabei wird dieser Teil des Programms isoliert betrachtet und alle Abhängigkeiten durch Testcode ersetzt. Auf diese Weise kann sehr effizient der fehlerhafte Bereich des Programmcodes gefunden werden [11]. Von der klassischen Begriffsdefinition abweichend ist die betrachtete Einheit im Rahmen dieses Berichts die gesamte Implementierung des modellprädiktiven Pfadfolgereglers. Dieser ist ein Teilmodul in dem Softwarestack für das autonome Fahren und wird durch die Bereitstellung der Simulationsumgebung isoliert.

IMPLEMENTIERUNG

In diesem Kapitel werden die implementierten Funktionen und Klassen dokumentiert. Dabei wird zunächst der Aufbau und der 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 wird der Ablauf eines Testskriptes erläutert und auf die Einbindung in die GitLab CI-Pipeline eingegangen. Außerdem werden die KPIs zur Bewertung der Simulationsergebnisse eingeführt.

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 parametrisiert 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 bestimmt werden, die sowohl die Sicherheit als auch den Komfort der gefundenen Lösung bewerten. Diese Kriterien werden im weiteren Verlauf als KPIs bezeichnet.

Ein KPI ist ein Grenzwert für eine messbare Größe in den Simulationsdaten. Wird dieser Grenzwert durch ein Über- bzw. Unterschreiten verletzt, führt dies zu einem negativen Ergebnis für den Testlauf.

Aus den Testergebnissen soll abschließend 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 und straßenähnlich dargestellt. Darauf wird die tatsächliche und prädizierte Bewegung des Fahrzeugs abgebildet. In Abbildung 3.1 ist dieser generierte Live-Plot zu sehen.

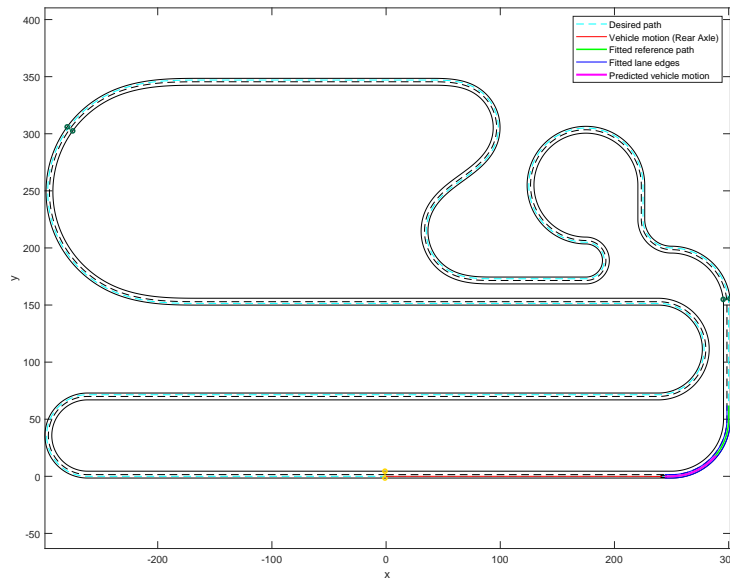


Abbildung 3.1: Beispiel einer komplexen Simulationsstrecke

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 [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. Diese unterscheiden sich vornehmlich im Funktionsumfang, wobei klassenbasierte Tests am umfangreichsten sind. Diese bieten als einzige die Möglichkeit, Tests durch externe Parameter zu parametrisieren. Da dies eine direkte Anforderung an das zu entwickelnde Konzept ist, wurde sich für die Verwendung der klassenbasierten Tests entschieden [10].

Das grundlegende Element dieses Frameworks ist der *test case*. In diesem wird über *assertions*, dies sind meist logische Operationen, festgestellt, ob sich ein System im Normbereich befindet. 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 [15]. 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 zudem möglich, externe Parameter in die Testklasse zu laden. Diese können als Testparameter, Testinitialisierungsparameter oder Klasseninitialisierungsparameter übergeben werden. Abhängig davon, welche Parameter-typen und wie viel Variationen definiert sind, werden Testfälle und die verschiedenen Setup-Methoden mehrmals ausgeführt.

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 oder XML, aufbereitet.

Die objektorientierte Natur von MATLAB® und dem Unit-Test Framework ermöglichen zusätzlich die Erweiterung der Funktionalität durch das Implementieren von eigenen Plugins 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 in den Workspace definiert. Beim Start der Simulation werden diese in das Simulink Modell geladen. Über die Protokollierungsfunktion 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 geeignete Methoden, beispielsweise wie in [21] vorgestellt, für die gestellten Anforderungen der Fahrzeugführung parametrisiert. 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 Klothoide - erzeugt werden kann. Eine Klothoide ist ein Kreisbogen, dessen Krümmung proportional zur Länge ist und somit keinen Sprung in der Krümmung aufweist [23].

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 3.2 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. Die Umgebungsklasse vereint alle Funktionen zur korrekten Initialisierung des Unit-Test Frameworks. Dabei wird zunächst 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 Parameterwerte in die Testklasse geladen.

Der konfigurierte Testrunner kann nun die Testsuite ausführen. Da die Größe der KPIs teilweise szenarien- und parameterabhängig ist, werden diese dynamisch für jede Simulation neu bestimmt und geladen. In 3.4.4 wird darauf näher eingegangen.

Aus den Ergebnissen 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.

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.

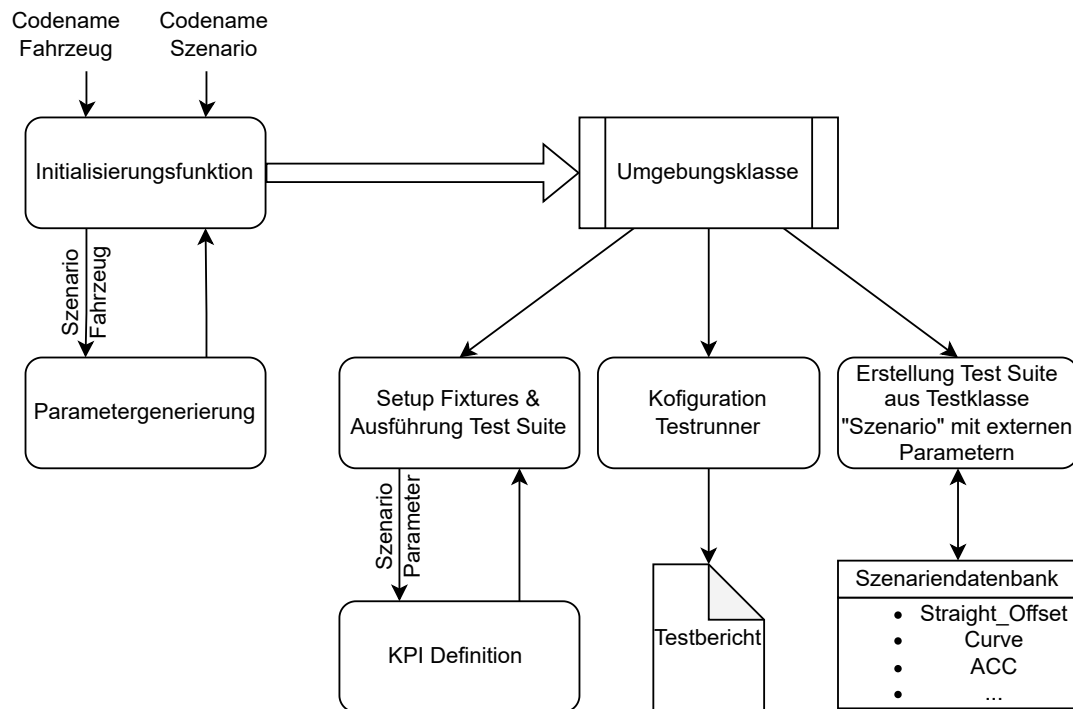


Abbildung 3.2: schematischer Ablauf eines Testskripts zur Ausführung eines Szenarios

Besitzt ein Szenario N Parameter und existieren 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. Vor allem 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 das 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 Samplingmethode ermöglicht eine Reduzierung der Anzahl der benötigten Simulationen. Für jeden Parameter wird aus dem jeweiligen Wertebereich ein zufälliger Wert gezogen. Dies wird für eine vorher definierte Anzahl an Samples wiederholt. Eine gute Abdeckung des gesamten Wertebereichs kann dabei allerdings nicht garantiert werden. Eine homogenere Verteilung wird durch die Verwendung des Latin Hypercube Samplings (LHS) erreicht [12]. Dabei wird der Wertebereich ei-

nes jeden Parameters in k äquivalent große Teilintervalle unterteilt. Aus jedem Teilintervall wird ein Wert zufällig ermittelt. Durch eine Permutation wird zusätzlich die Reihenfolge der ermittelten Werte geändert, da sonst beim Zusammenfügen zu Samplesets alle Parameterwerte in aufsteigender Reihenfolge vorliegen würden. Nach Abschluss dieser Methode stehen k -Samplesets zur Verfügung, die für alle Parameter in den jeweiligen Grenzen eine gute Abdeckung des Parameterraums bieten. Ein Vergleich der Verteilung von zufälligen Samples über den gesamten Parameterraum und dem LHS ist in Abbildung 3.3 zu sehen.

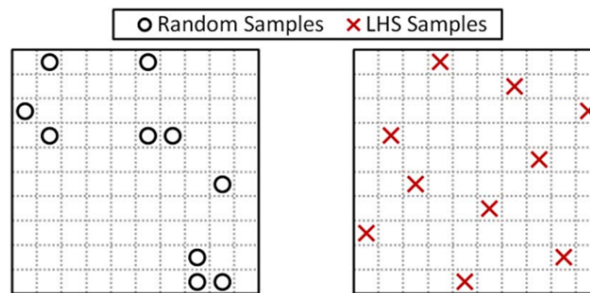


Abbildung 3.3: Vergleich von zufälligem Sampling mit Latin Hypercube Sampling [18]

3.4.4 Definition von Key Performance Indicators und Testkriterien

Um eine Bewertung der Simulationsergebnisse vornehmen zu können, werden geeignete Daten und Grenzwerte für diese benötigt. Wird in Tests ein Über- bzw. Unterschreiten dieser festgestellt, gilt der Test als nicht bestanden. Diese Testkriterien 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 des Fahrzeugs benötigt. Wird die Differenz der beiden Datensätze nicht innerhalb der in der KPI definierten Zeitspanne abgebaut, gilt der Test als fehlgeschlagen.

Es existieren Normen und Studien zur Bewertung von autonomen Fahrzeugen hinsichtlich der Sicherheit und des wahrgenommenen Komforts. In [22] werden Bedingungen für die Zulassung von Fahrzeugen, welche in die Querführung des Fahrzeugs eingreifen, definiert. [20] etabliert Standards für die Funktionen und Testkriterien für ein ACC-System. Weiterhin werden im Testkatalog von Euro NCAP (New Car Assessment Programme) [17], einer Organisation, die die Sicherheit von Fahrzeugen bewertet und strenge Prüfkriterien für das Verhalten von Fahrassistenzsystemen anwendet, ebenfalls Testszenarien und Grenzwerte festgelegt. Nachfolgend wird auf die verwendeten KPIs und die Ermittlung von möglichen Grenzwerten näher eingegangen.

Laterale Beschleunigung

Die laterale Beschleunigung hat großen Einfluss auf den Fahrkomfort [3]. In [20] wurde das Verhalten eines durchschnittlichen Fahrers in Kurven bei verschiedenen Geschwindigkeiten analysiert, siehe Abbildung 3.4. In [22] werden für verschiedene Fahrzeugklassen in verschiedenen Geschwindigkeitszonen maximale Querbeschleunigungen definiert, siehe Tabelle 3.1.

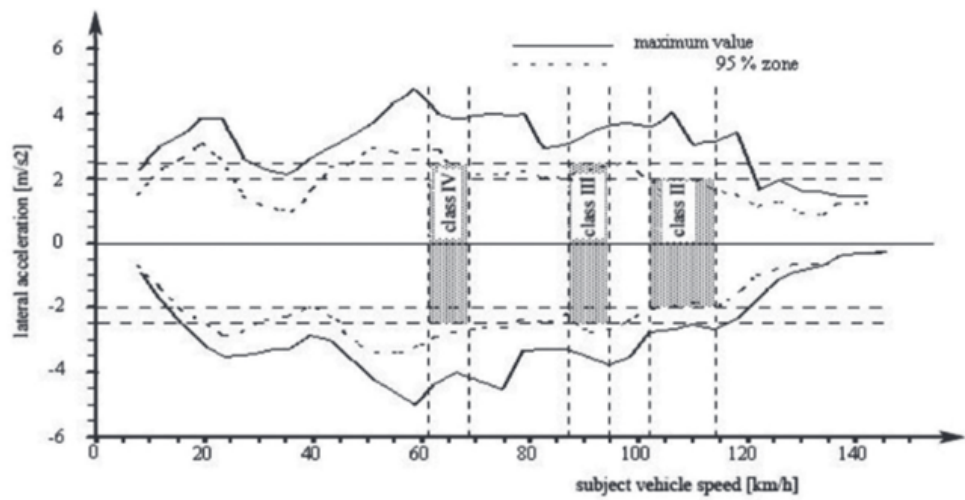


Abbildung 3.4: laterale Beschleunigung in Kurven eines typischen Fahrers [20]

Tabelle 3.1: Höchstwerte für laterale Beschleunigung, aufgeschlüsselt nach Fahrzeugklassen und Geschwindigkeitszonen [22]

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	

Longitudinale Beschleunigung

Diese KPI ist vor allem im Kontext des ACC von großer Relevanz, da auf ein vorausfahrendes Fahrzeug möglichst sanft reagiert werden sollte. In [20] sind dafür Grenzwerte definiert (Abbildung 3.5). Darin werden für diese Größe keine Maximalwerte, sondern der gleitende Durchschnitt über zwei Sekunden betrachtet, woran sich die Bewertung in den implementierten Tests ebenfalls orientiert.

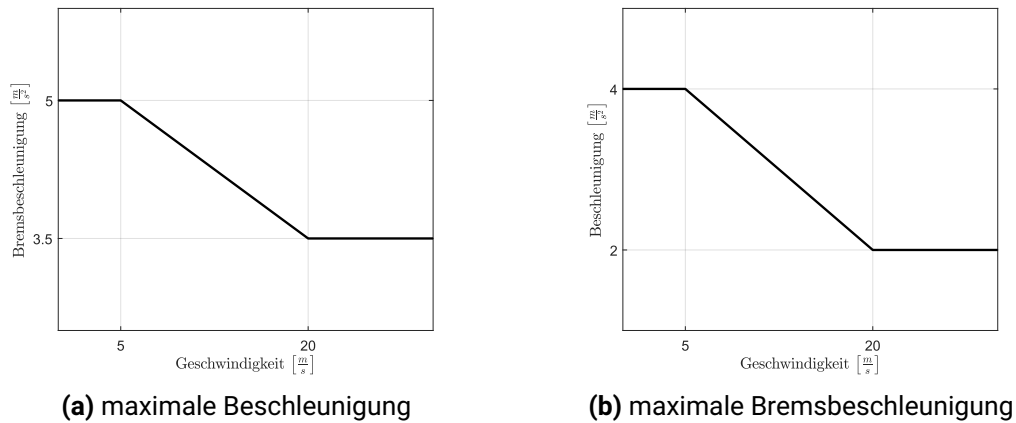


Abbildung 3.5: maximal zulässige longitudinale Beschleunigung, Durchschnitt über 2 s [20]

Ruck

Ruck ist ein unerwünschtes Verhalten bei Fahrzeugen, das durch ungleichmäßige Beschleunigungs- und Bremsvorgänge entsteht und maßgeblich für die Wahrnehmung von Komfort mitverantwortlich ist. [20] definiert im ACC Kontext dafür einen Grenzwert für einen negativen longitudinalen Ruck, wie er beim Abbremsen entsteht. Ausgehend davon wird der auftretende laterale Ruck in den Tests betragsmäßig gesehen und entsprechend bewertet. In [22] darf der gleitende Mittelwert des lateralen Rucks über eine halbe Sekunde nicht größer als $5 \frac{m}{s^3}$ sein.

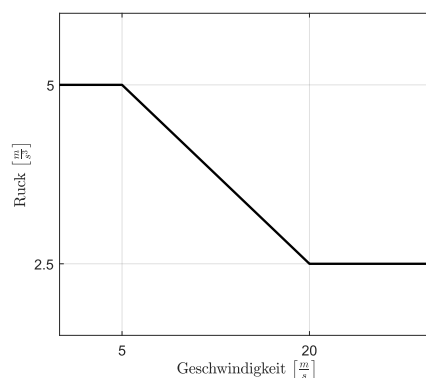


Abbildung 3.6: maximal zulässiger negativer Ruck, Durchschnitt über 1 s [20]

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 wird ein System mit automatischer Geschwindigkeitsanpassung als sehr gut bewertet, wenn es die Geschwindigkeit auf $\pm 2 \frac{km}{h}$ anpassen kann, bevor die Frontachse des Fahrzeugs eine Zone mit niedrigerer Pfadgeschwindigkeit betritt [17]. Da die Simulation die Hinterachse als Referenz nutzt und darauf regelt, wird dieses Kriterium 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. Diese Zeit ist beendet, wenn die betrachtete Größe weniger als $\pm 10\%$ vom Sollwert abweicht.
- **Reaktionszeit** - benötigte Zeit, bei der eine Sprungantwort zum ersten Mal die neue Sollgröße erreicht
- **Differenzgeschwindigkeit** - Geschwindigkeitsdifferenz zwischen Ego-Fahrzeug und dem ACC-Target zu einem bestimmten Zeitpunkt (abhängig vom Szenario)
- **Distanzfehler** - Abweichung von der gewünschten Distanz zwischen dem Ego und einem Zielobjekt zu einem bestimmten Zeitpunkt (abhängig vom Szenario)

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 3.2 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 zu vereinfachen 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 Fahrzeug 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 3.7). Aus dieser ist direkt zu entnehmen, welche Szenarios für welches Fahrzeug (Alice, IAVShuttle) ausgeführt wurden und welche Szenarien zu Fehlern geführt haben.

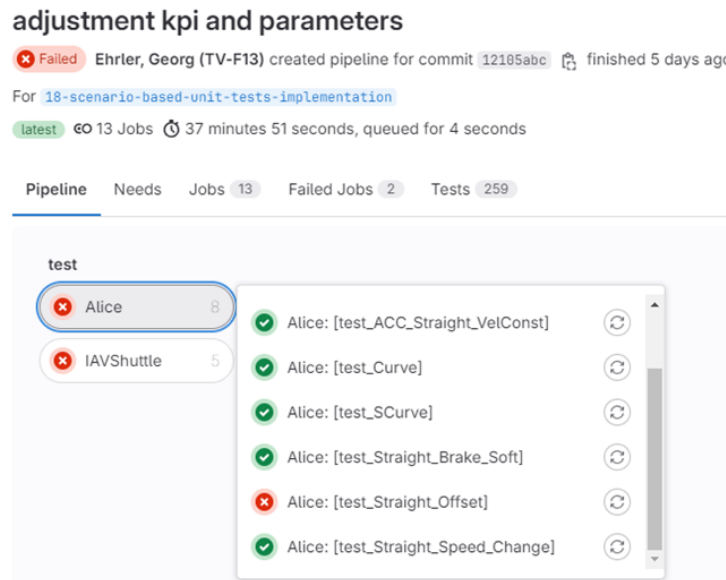


Abbildung 3.7: Darstellung der Jobs in der GitLab CI Pipeline im Browser. “Alice” und “IAVShuttle” sind Codenamen von Prototypenfahrzeugen

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 heruntergeladen werden.

Auf der Startseite befindet sich eine Übersicht (Abbildung 3.8). Schlägt ein Test fehl, gibt es einen ausführlicheren Bericht (Abbildung 3.9). 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.

MATLAB® Test Report

Timestamp: 08-Mar-2024 12:35:34
Host: 20IAV015842P-0
Platform: win64
MATLAB Version: 23.2.0.2428915 (R2023b) Update 4
Number of Tests: 20
Testing Time: 348.3869 seconds
Overall Result: FAILED

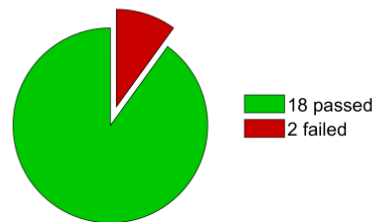


Abbildung 3.8: Übersichtliche Darstellung der Testergebnisse eines Szenarios auf der ersten Seite des Testberichts

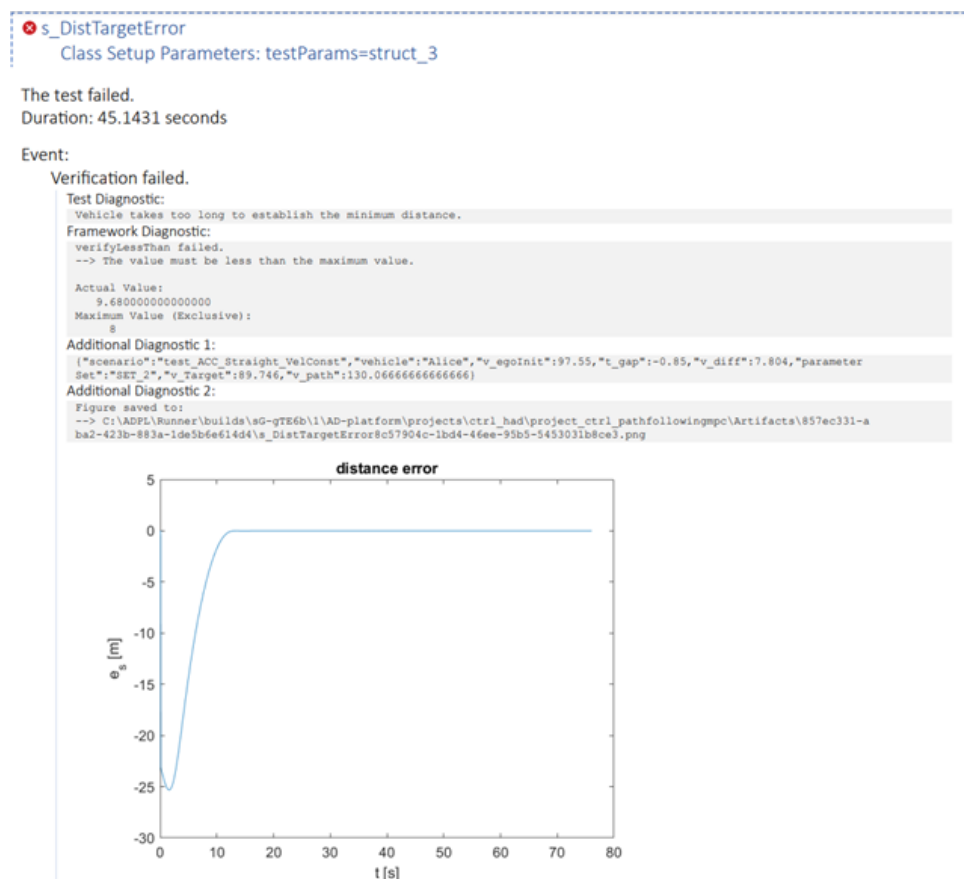


Abbildung 3.9: Darstellung eines fehlgeschlagenen Tests im Testbericht. Der Wert der KPI und der tatsächliche Wert werden verglichen. Ein kurzer Text gibt dem Entwickler wieder, was fehlgeschlagen ist. Wenn es sinnvoll ist, wird ein Plot mit den relevanten Daten erzeugt. Die Angabe der Simulationsparameter 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 diesen Anwendungsfall sind viele Szenarien allerdings nicht von Relevanz, da die Pfadplanung von einem übergeordneten Modul realisiert wird und bspw. Kreuzungsszenarien zu einfachen Kurvenszenarien reduziert werden. In [20] und [17] werden verschiedene Szenarien für die Einhaltung gesetzlicher Vorschriften und für ACC-Systeme definiert. Weiterhin wurden Negativtests implementiert. Dabei wird ein Szenario bewusst so parametrisiert, dass es der Simulation nicht möglich ist, die KPIs einzuhalten ohne Beschränkungen, beispielsweise dem maximalen Lenkeinschlag, zu verletzen. In diesen Fällen ist ein nicht einhalten der KPI Grenzwerte als positiv zu bewerten und der Test damit bestanden. Nachfolgend 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 geringe 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:

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

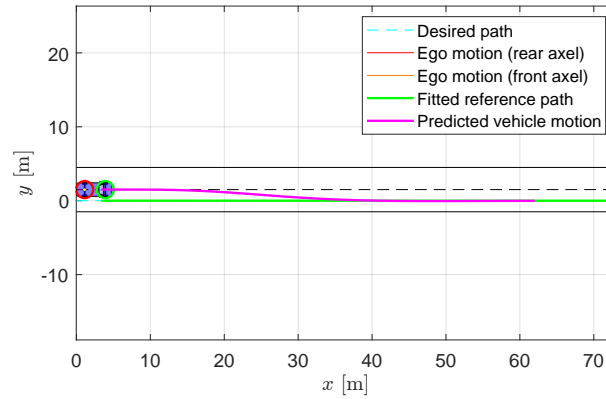


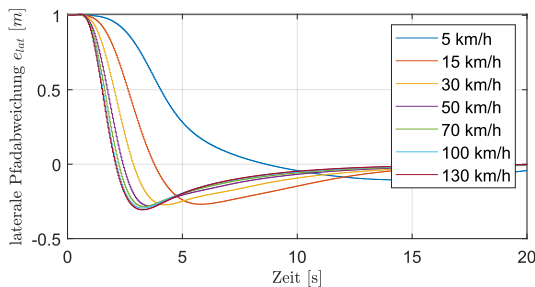
Abbildung 4.1: Darstellung des Szenarios: Gerade mit initialem Versatz

4.1.2 Abgetestete KPIs

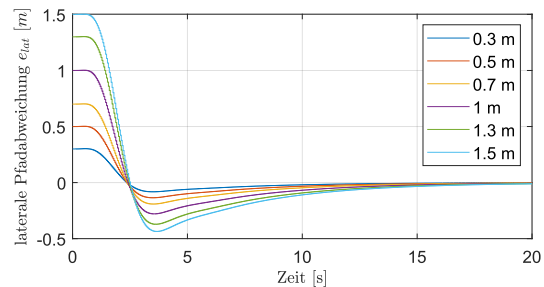
Es wurden Test für folgende KPIs mit den dazugehörigen Grenzwerten implementiert:

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 (Abbildung 4.2a) und konstantem Versatz von 1 m verschiebt sich der Zeitpunkt des maximalen Überschwingens nach hinten, je geringer die Geschwindigkeit wird.



(a) Variation der Initialgeschwindigkeit



(b) Variation des initialen Versatzes

Abbildung 4.2: KPI Einschwingzeit abhängig der Simulationsparameter

longitudinale Geschwindigkeitsabweichung: 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 vor allem bei niedrigeren Geschwindigkeiten und höheren Versätzen zu beobachten, siehe Abbildungen 4.2.

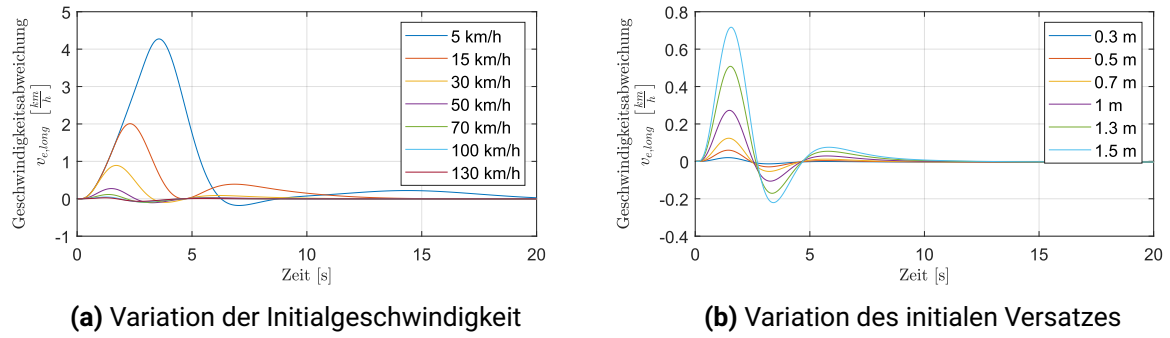


Abbildung 4.3: KPI Geschwindigkeitsabweichung abhängig der Simulationsparameter

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.

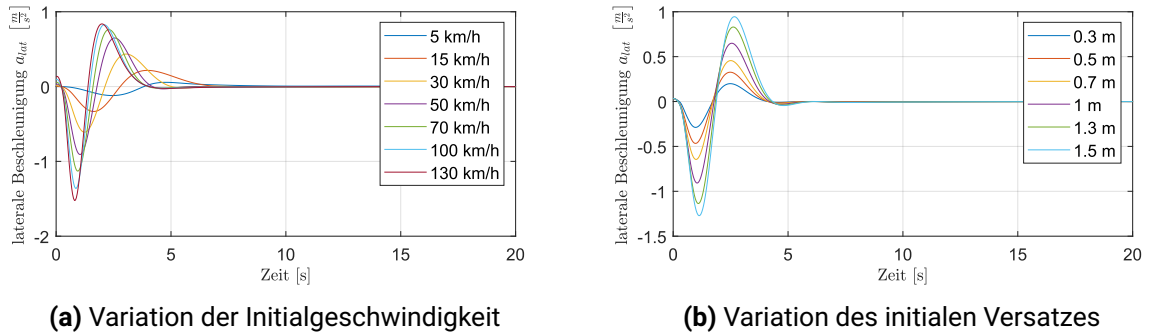


Abbildung 4.4: KPI laterale Beschleunigung abhängig der Simulationsparameter

lateralen 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.

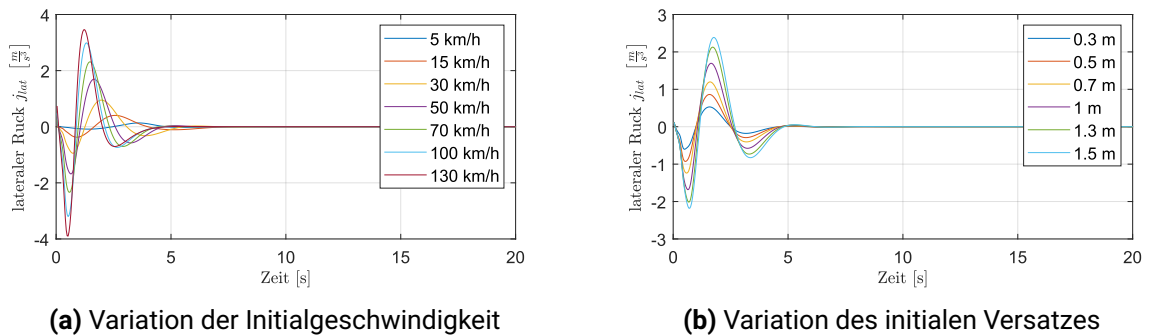


Abbildung 4.5: KPI lateraler Ruck abhängig der Simulationsparameter

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 einem Kreis mündet. 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.

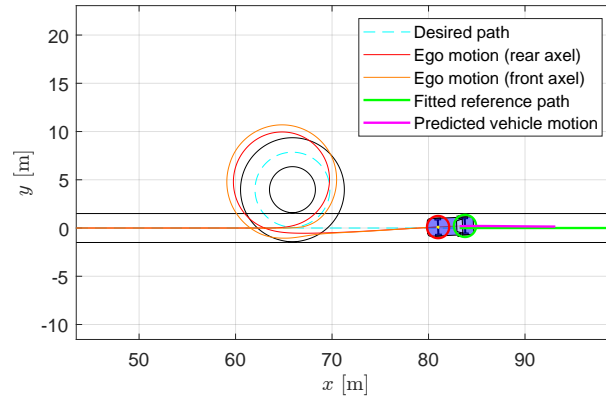


Abbildung 4.6: Darstellung des Szenarios: Kurvenfahrt negativ

4.2.1 Parametrierung

Die Parameter für dieses Szenario sind:

- maximale Querbewegung 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 Tests 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.

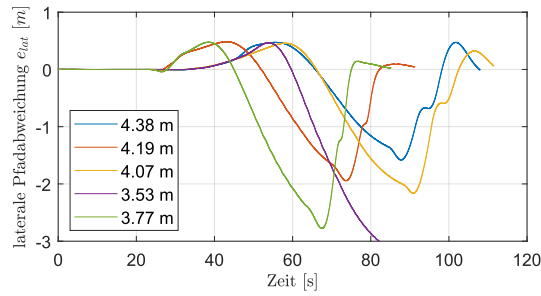


Abbildung 4.7: KPI Pfadabweichung bei verschiedenen Kurvenradien

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 existiert 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. Der rote Kegel stellt den Bereich dar, in welchem das Ego das Target als ACC-Objekt erkennt und darauf regelt.

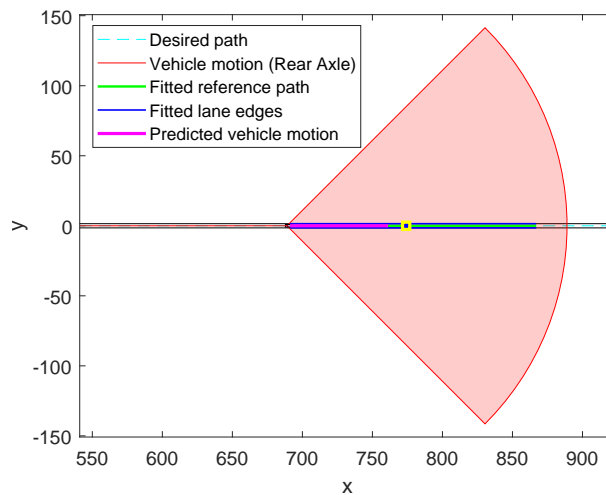


Abbildung 4.8: Darstellung des Szenarios: ACC mit konstanter Objektgeschwindigkeit

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 Rucke und Beschleunigungen sind stark abhängig von der gewählten Parametrierung. Eine Verletzung der Grenzwerte ist hier nicht zulässig. Vor allem darf die maximale Beschleunigung nicht größer sein als ein definierter Grenzwert. Sowohl Beschleunigung als auch Ruck befinden sich in einem akzeptablen Rahmen, siehe Abbildung 4.9.

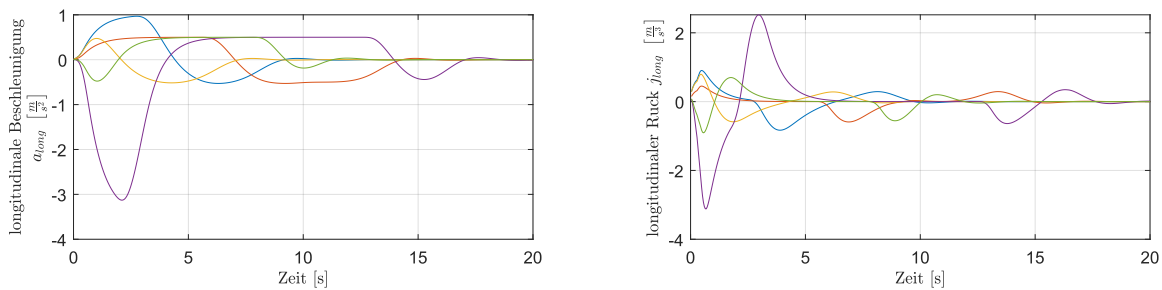


Abbildung 4.9: Simulationsdaten für longitudinale Beschleunigung und Ruck bei unterschiedlichen Parametrierungen

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 verringert werden. Allerdings ist die Definition eines festen Grenzwertes nicht möglich, da diese stark Parametrierungsabhängig ist (Abbildung 4.10). Die genaue Größe der KPI ergibt sich dynamisch aus der aktuellen Parametrierung der Simulation.

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.

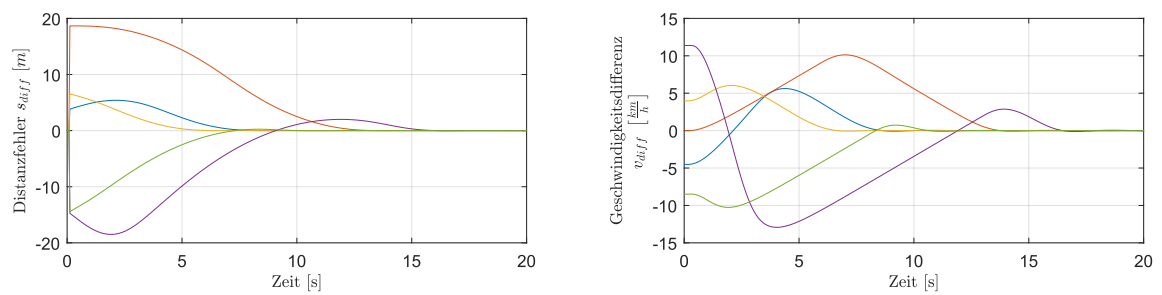


Abbildung 4.10: Simulationsdaten für Abstands- und Distanzfehler bei unterschiedlichen Parametrierungen

ZUSAMMENFASSUNG UND AUSBLICK

Nachdem in der Einleitung die Einordnung und Vorstellung des Themas erfolgt war, 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 das 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 das Abtesten von KPIs. Neue Szenarien können durch das Anlegen einer neuen Testklasse, Definition von KPIs und Parametern 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 Kurvenradien äquidistant 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 Äquivalenzklasse gleich groß ist.

Die automatisierte Ausführung der Testskripte und Testklassen in einer CI Pipeline wurde erreicht. Die Testergebnisse werden übersichtlich in eine 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](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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1007/978-0-85729-398-5).
- [6] T. Faulwasser, B. Kern und R. Findeisen. „Model predictive path-following for constrained nonlinear systems“. In: *48th 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](https://doi.org/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] Vladimir Khorikov. *Unit Testing Principles, Practices, and Patterns*. New York: Manning Publications Co. LLC, 2020. 1252 S. ISBN: 9781638350293.
- [12] 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](https://doi.org/10.2307/1268522).
- [13] 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](https://doi.org/10.1109/IVS.2018.8500406).

- [14] Henry van Merode. *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress, 2023. ISBN: 9781484292280. DOI: [10.1007/978-1-4842-9228-0](https://doi.org/10.1007/978-1-4842-9228-0).
- [15] Gerard Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.
- [16] 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](https://doi.org/10.46720/f2020-acm-096).
- [17] 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/>.
- [18] 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](https://doi.org/10.1109/TPWRS.2015.2417204).
- [19] 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](https://doi.org/10.1016/j.ifacol.2019.08.112).
- [20] International Organization for Standardization. *ISO 15622:2018. Intelligent transport systems – Adaptive cruise control systems – Performance requirements and test procedures*. Sep. 2018.
- [21] 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](https://doi.org/10.3390/math11020465).
- [22] Wirtschaftskommission der Vereinten Nationen für Europa. *UNECE-R79. Einheitliche Bedingungen für die Genehmigung der Fahrzeuge hinsichtlich der Lenkanlage*. 16. Okt. 2018.
- [23] Wikipedia. *Klothoide*. Wikimedia Foundation, Inc. 2024. URL: <https://de.wikipedia.org/wiki/Klothoide> (besucht am 18. 03. 2024).

Name: Ehrler	Bitte beachten: 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.
Vorname: Georg	
geb. am: 07.07.1999	
Matr.-Nr.: 581446	

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende **schriftliche Arbeit** ☒ 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: **15.09.2024**

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.

