Zeitlich präzises Steuergerätetestsystem auf Basis einer nicht echtzeitfähigen Plattform

Dissertation

der Mathematisch-Naturwissenschaftliche Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Ing. (FH) Daniel Ulmer, MEE
aus Sindelfingen

Tübingen
2012

Tag der mündlichen Qualifikation:	??.??.2012
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Wolfgang Rosenstiel
2. Berichterstatter:	Prof. Dr.-Ing. Ina Schieferdecker (Technische Universität Berlin)

Vorwort

Die vorliegende Arbeit entstand am Lehrstuhl für Technische Informatik des Wilhem-Schickard-Instituts für Informatik der Universität Tübingen im Rahmen eines Kooperationsprojektes zwischen dem Team „Softwareintegration und Test“ der Abteilung EP/ERV der Daimler AG in Sindelfingen und der IT-Designers GmbH in Esslingen.

Ich bedanke mich bei Herrn Prof. Dr. W. Rosenstiel für die Betreuung der Arbeit und die Übernahme des Hauptberichts. Für die Übernahme des Zweitberichts gilt mein Dank ebenso Frau Prof. Dr.-Ing. I. Schieferdecker. Mein besonderer Dank gilt Dr. O. Bühler und Prof. Dr. U. Bröckl für ihre fachlichen Diskussionen und ihre Korrekturen. **Bei Prof. Dr. J. Goll bedanke ich mich für das Umfeld, die anregenden Diskussionen und für die Unterstützung bei den Veröffentlichungen und Patenten.**

Bei meinen Kollegen und den ehemaligen Kollegen im Umfeld des Kooperationsprojektes möchte ich mich für die Hilfsbereitschaft bedanken. Insbesondere bedanke ich mich bei Dr. Helmut Keller und bei Joachim Missel, die diese Arbeit in der Abteilung EP/ERV ermöglicht haben.

Außerdem bedanke ich mich bei allen Herren der Firma Elektrobit, die den Weg zum Produkt für die in dieser Arbeit vorgestellte Technologie ermöglicht haben. Zum Zeitpunkt der Arbeit wurde ein Verkaufsstart für das Jahr 2012 angestrebt.

Herrenberg, im Juli 2011

Daniel Ulmer

Inhaltsverzeichnis

1. Einführung **1**
- 1.1. Motivation . . . 1
- 1.2. Zielsetzung der Arbeit . . . 2
- 1.3. Aufbau der Arbeit . . . 3

2. Grundlagen **5**
- 2.1. Controller Area Network . . . 6
- 2.2. Fahrerassistenzsysteme . . . 10
- 2.3. Funktionsorientierter Test von eingebetteten Systemen . . . 15

3. Stand der Technik **17**
- 3.1. Entwicklungsprozess von Fahrerassistenzsystemen . . . 17
  - 3.1.1. Entwurfsprozess . . . 20
  - 3.1.2. Der Integrations- und Testprozess . . . 22
- 3.2. Testen von Fahrerassistenzsystemen . . . 27
  - 3.2.1. Testen von Fahrerassistenzsystemen im Fahrzeug . . . 27
  - 3.2.2. Datenaufzeichnung bei interagierenden Fahrzeugen . . . 29
  - 3.2.3. Abspielen von im Fahrzeug aufgezeichneten Daten . . . 33
  - 3.2.4. Testen von Fahrerassistenzfunktionen an Testsystemen . . . 35
- 3.3. Grenzen derzeitiger Testsysteme . . . 41
  - 3.3.1. Flexible Beeinflussung des Zeitverhaltens und Wahren der verlangten Stabilität des Zeitverhaltens . . . 42
  - 3.3.2. Ubiquitäre Testsysteme . . . 45
- 3.4. Zusammenfassung vom Stand der Technik . . . 46

4. Konzept eines Zeitstempel-basierten Testsystems **49**
- 4.1. Methode des Zeitstempel-basierten Testsystems . . . 50
- 4.2. Architektur des Realzeitadapters (RTA) . . . 56
  - 4.2.1. Uhrregelung . . . 59

4.2.2. Zeitstempeleinheit 61
4.2.3. Empfangswarteschlange 62
4.2.4. Zeitgesteuerte Sendeeinheit 62
4.2.5. Sendewarteschlange 62
4.2.6. Sendeverzugseinheit 64
4.2.7. Fehlerwarteschlange 64
4.2.8. Realzeitadapter mit vier CAN-Bussen 65
4.3. Definition des Zeitverhaltens 68
4.4. Messungen des am Prototypen erreichten Zeitverhaltens 70
4.5. Zusammenfassung des Konzepts 76

5. Anwendung des Konzepts bei Fahrerassistenzsystemen 79
5.1. Datenlogger für die Integrationsstufe Fahrzeug 79
5.2. Testsystem für die Integrationsstufe Steuergerät 81
5.3. Einordnung in den Entwicklungsprozess 84
5.3.1. Bewertung von interagierenden Fahrzeugen 84
5.3.2. Bewertung von Fahrerassistenzfunktion im Labor 85
5.3.3. Ausblick für weitere Verbesserungen im Entwicklungsprozess 87

6. Experimentelle Untersuchungen 91
6.1. Steuergerätetestsystem für ein FAF-Steuergerät 91
6.1.1. Zeitverhalten im Vergleich zu existierenden Lösungen . . . 92
6.1.2. Test der Sensorplausibilisierung 100
6.2. Auswertung von Fahrzeugdaten eines Abstandsregeltempomaten . 103
6.3. Bewertung 109

7. Zusammenfassung und Ausblick 111
7.1. Zusammenfassung 111
7.2. Ausblick 115

A. Programmierschnittstelle des Prototypen 117
A.1. Initialisieren und Beenden 118
A.2. CAN-Nachrichten senden und empfangen 119
A.3. Ereignisse und Fehlermeldungen 120

B. Begriffsverzeichnis 123

Literaturverzeichnis 129

1. Einführung

1.1. Motivation

Laut Wirtschaftswoche [90] ist seit Jahren die Elektronik der wichtigste Innovationstreiber der Fahrzeugbranche. Bei europäischen Fahrzeugbauern entfallen 40% der Herstellungskosten auf Elektronikkomponenten, Kabel, Halbleiter und Software. Es wird ein Zusammenwachsen von Internet und Fahrzeug prognostiziert. Dabei sollen nicht nur Interenetanwendungen, wie beispielsweise Email oder Nachrichten im Fahrzeug verfügbar werden, sondern die breitbandige Kommunikation soll zukünftig auch helfen, Unfälle zu vermeiden. Hierzu kommunizieren die Fahrzeuge mit Ampeln, Verkehrsleitsystemen und mit anderen Fahrzeugen. Erste Fahrzeuge, die eine direkte Kommunikation zwischen zwei Fahrzeugen über eine Funkverbindung ermöglichen, sollen spätestens 2015 auf den europäischen Straßen sein.

Auch ohne die direkte Kommunikation über Funkverbindungen erfassen bereits heutige Fahrzeuge ihr Umfeld. Zunächst kamen hierfür Ultraschallsensoren zum Einsatz, welche dann schrittweise durch verschiedene Radar- und Videosysteme ergänzt wurden. Auf Basis der erfassten Informationen über das Fahrzeugumfeld bieten heutige Fahrzeuge bereits viele Funktionen, die Unfälle in ihrer Schwere mindern oder ganz vermeiden.

Die beschriebenen Funktionen sind zwischenzeitlich nicht nur in Oberklassefahrzeugen zu finden, sondern auch in den anderen Fahrzeugklassen beziehen sich oftmals die meisten Innovationen auf die Einführung von Fahrerassistenzsystemen auf Basis von Elektronik und Software [36].

Für die Umsetzung der gesamten Funktionalität eines Fahrzeugs sind zwischen-

zeitlich bis zu 100 Millionen Code-Zeilen nötig [15, 13]. Inkrementelle Verbesserungen der Software führen dazu, dass teilweise im Tages-Rhythmus neue Software gebaut wird. Nach Broy in [13] ist eine große Herausforderung für die Informatik im Automobilbereich, sowohl geeignete Software-Architekturen zu entwerfen als auch beispielsweise durch Testautomatisierung für einen umfassenden Test zu sorgen. Insbesondere sicherheitskritische Systeme stellen hohe Anforderungen an die Zuverlässigkeit von Steuergeräten und von deren Software. Eine Komplexitätsreduktion ist meist aus Kostengründen nicht optimal durchsetzbar, was zu vielen Varianten, knapp ausgelegten Mikrokontrollern und einfachen Bussen zur Vernetzung von Steuergeräten führt.

Ulrich Mellinghoff, Leiter der Direktion Safety, NVH[1] und Testing begründet in [74] den Bau eines neuen Prüfgeländes in der Nähe des Entwicklungszentrums von Mercedes damit, dass bei der Erprobung von immer mehr Varianten und Assistenzsystemen aus Effizienzgründen ein Prüfgelände möglichst nahe am Entwicklungszentrum nötig ist. Bevor die Fahrzeuge in der realen Welt erprobt werden, fordert er, dass die Fahrzeuge mit Hilfe von Simulationen – diese nennt er digitale Welt – entwickelt und getestet werden.

> „Unser Ziel ist es, einen optimalen Mix aus digitaler und hardwareorientierter Erprobung zu erreichen.“ [74]

1.2. Zielsetzung der Arbeit

In dieser Arbeit wird ein Testsystem für Fahrerassistenzfunktionen entwickelt, das sowohl die Erprobung von Fahrerassistenzfunktionen auf einem Prüfgelände als auch deren Test in einer virtuellen Welt ermöglicht. Weiterhin soll eine zusätzliche Effizienzsteigerung dadurch erreicht werden, dass die auf einem Prüfgelände gewonnen Daten mit Hilfe des vorgestellten Testsystems für Tests am Arbeitsplatz verwendet werden können. Das Testsystem soll sich so in den Entwicklungsprozess integrieren, dass Software, die im Tages-Rhythmus gebaut wird und eine hohe Variantenvielfalt aufweist, effizient und kostengünstig getestet

[1] NVH steht für Noise, Vibration, and Harshness und bezieht sich auf die Geräusche, Vibration und Rauheit von Fahrzeugen.

werden kann. Es soll insbesondere für den Test von sicherheitskritischen Fahrerassistenzfunktionen auf einem Steuergerät geeignet sein.

Derzeitige Testsysteme überlassen die Einhaltung des Zeitverhaltens dem Betriebssystem. Das Zeitverhalten wird daher unabhängig von der Logik der Simulation durch das verwendete Betriebssystem festgelegt und verantwortet. Basierend auf diesem Ansatz existieren zwei Arten von Testsystemen.

Zum einen Testsysteme, die auf spezieller, echtzeitfähiger Hard- und Software basieren und daher ein präzises Zeitverhalten erreichen. Nachteilig ist hierbei, dass das Zeitverhalten derzeit nicht aus der Simulationslogik heraus beeinflusst werden kann. Weiterhin sind diese Testsysteme nicht für den Test von Fahrerassistenzfunktionen im Fahrzeug konzipiert und eigenen sich auf Grund ihrer Größe und ihres Preises nicht für einen ubiquitären Einsatz, bei dem viele transportable Testsysteme nötig sind.

Zum anderen existieren Testsysteme auf Basis von Arbeitsplatzrechnern, die sich dadurch auszeichnen, dass sie preisgünstig und transportabel sind. Als Nachteil ist aber deren eingeschränkte Echtzeitfähigkeit zu nennen.

Ziel dieser Arbeit ist es, ein Testsystem zu entwickeln, das die Nachteile der bisherigen Testsysteme nicht aufweist. Dies bedeutet, dass das Testsystem echtzeitfähig sein soll und gleichzeitig preisgünstig und transportabel. Zusätzlich soll dessen Zeitverhalten feingranular aus der Simulationslogik heraus beeinflussbar sein.

1.3. Aufbau der Arbeit

Nach einem kurzen Überblick über den in dieser Arbeit verwendeten CAN-Bus, die Welt der Fahrerassistenzsysteme und die Testmethode des funktionsorientierten Tests von eingebetteten Systemen in Kapitel 2 beschreibt Kapitel 3 den Stand der Technik. Es widmet sich dem Entwicklungsprozess von Fahrerassistenzsystemen mit besonderem Augenmerk auf die Vorgehensweise bei deren Test. Außerdem werden derzeitige Testsysteme und deren Grenzen für den Test im Fahrzeug und von Steuergeräten mit Fahrerassistenzfunktionen beschrieben.

In Kapitel 4 wird dann das Konzept eines Zeitstempel-basierten Testsystems vorgestellt, das die zuvor aufgezeigten Schwachpunkte von Testsystemen adressiert. Dabei wird zunächst der methodische Ansatz eines Zeitstempel-basierten Testsystems beschrieben, um danach eine technische Lösungsmöglichkeit vorzustellen. Insbesondere wird gezeigt, wie bei dem Testsystem das Zeitverhalten beeinflusst werden kann und welche Genauigkeit hierbei der aufgebaute Prototyp erreicht.

Kapitel 5 beschreibt, wie das entwickelte Testsystem für die in Kapitel 3 aufgezeigten Problemstellungen eingesetzt werden kann. Kapitel 6 untermauert die Argumentation mit Experimenten. Das Zeitverhalten des entwickelten Testsystems wird hier mit dem von existierenden Lösungen verglichen. Außerdem wird gezeigt, wie die in einer Erprobung aufgezeichneten Daten eines Abstandsregeltempomaten über Fahrzeuggrenzen hinweg ausgewertet werden können. Kapitel 7 fasst die gewonnen Erkenntnisse zusammen und endet mit einem Ausblick auf zukünftige Arbeiten.

2. Grundlagen

Im Rahmen dieser Arbeit wird ein Steuergerätetestsystem betrachtet. In diesem Kapitel sollen zunächst grundlegende Begriffe hierzu wie *Steuergerät* und *Testsystem* erläutert werden. In dieser Arbeit erfolgt die Kommunikation zwischen Testsystem und Steuergerät über *CAN-Busse* (Controller Area Network), daher sollen die Grundlagen dieses Busses in Kapitel 2.1 erläutert werden. Da aus der Sicht von Steuergeräten an die Kommunikation Echtzeitanforderungen gestellt werden, soll hier auch der im Rahmen dieser Arbeit geltende Begriff von *Echtzeit* definiert werden und mit den Eigenschaften des CAN-Busses in Bezug gesetzt werden. Kapitel 2.2 führt in die Begriffswelt der *Fahrerassistenzsysteme* ein, deren Steuergeräte im weiteren Verlauf mit dem vorgestellten Testsystem getestet werden. Kapitel 2.3 positioniert den Begriff *funktionsorientierter Test* aus der Literatur des Testens und führt in diese *Testmethode* ein, die die Entwicklung des beschriebenen Testsystems motiviert hat.

In der Literatur wird der Begriff *System* definiert als „Gesamtheit, Gefüge von Teilen, die voneinander abhängig sind, ineinander greifen oder zusammenwirken“ [135]. Ein *eingebettetes System* ist nach Rosenstiel in [96] ein elektronisches System, das für eine spezielle Anwendung in ein größeres System integriert ist. Passend zur Anwendung kann das eingebettete System aus Hardware[1] und Software[2] bestehen [103]. Mit dem Gesamtsystem interagiert ein eingebettetes System unter verschieden strengen Zeitbedingungen [108]. Daher werden in [103] eingebettete Systeme, bei denen neben der funktionalen Korrektheit auch deren Reaktionszeiten von Belang sind, auch *Echtzeitsysteme* genannt.

Die „Experimentelle Untersuchung zur Feststellung bestimmter Eigenschaften,

[1] Unter Hardware versteht man alle physikalischen Komponenten eines informationsverarbeitenden Systems oder eine Teilmenge davon [1].

[2] Als Software werden Programme, Prozeduren, Regeln und zugehörige Dokumentation eines informationsverarbeitenden Systems bezeichnet [1].

Leistungen und Ähnlichem“ wird *Test* genannt [135]. Beim Testen eines eingebetteten Systems sollen somit die zugesicherten Eigenschaften – unter Anderem die Zeitbedingungen – untersucht werden. Dies muss in der Regel bei der Auswahl der Testmethoden und beim Aufbau eines Systems zum Test des eingebetteten Systems – im weiteren Verlauf *Testsystem* genannt – berücksichtigt werden. Weiterhin heißt dies, dass das Testsystem im Allgemeinen die Schnittstellen[3] des eingebetteten Systems zu dessen Umgebung im System bedienen muss.

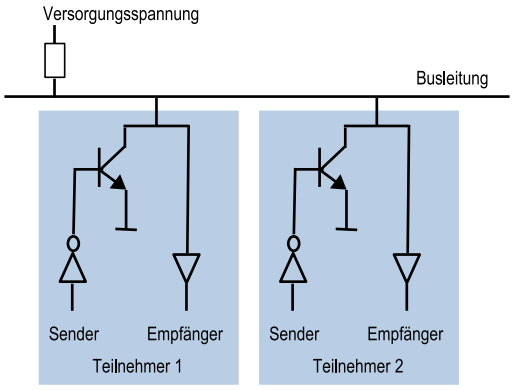
Ein eingebettetes System setzt sich in der Regel aus der Kombination von Sensorik, einer Kontrolleinheit und Aktorik zusammen [103]. Die Kontrolleinheit – wiederum ein eingebettetes System – wird als *Steuergerät* (engl.: Electronic Control Unit (ECU)) bezeichnet. Im Rahmen dieser Arbeit werden Steuergeräte betrachtet, die als Schnittstelle über CAN-Busse (siehe Kapitel 2.1) mit ihrer Umgebung und somit auch mit dem Testsystem kommunizieren. Die Software eines Steuergeräts in einem eingebetteten System wird nach Conrad [16] *eingebettete Software* genannt.

In dieser Arbeit soll insbesondere der *Funktionsorientierte Test* (siehe Kapitel 2.3) der eingebetteten Software von *Fahrerassistenzsystemen* (siehe Kapitel 2.2) betrachtet werden.

2.1. Controller Area Network

Der CAN-Bus (Controller Area Network) ist in der Kraftfahrzeugtechnik ein weit verbreitetes Bussystem zur Vernetzung von Steuergeräten. Marscholik und Subke bezeichnen diesen in [70] als „Standard für die Datenkommunikation im Fahrzeug“. Der CAN-Bus wurde zur ereignisgesteuerten Kommunikation [112] entworfen und arbeitet dabei nach dem CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance) Verfahren [26, 146]. Bei diesem Verfahren werden Kollisionen beim Zugriff auf den Bus durch das Buszugriffsverfahren erkannt und aufgelöst.

[3] Eine Schnittstelle ist die gemeinsame Grenze zwischen zwei funktionalen Einheiten. Die Schnittstelle wird dabei durch die Funktion, die physikalische Verbindung, den Signalaustausch und weitere Eigenschaften charakterisiert [1].



Sender		Pegel auf der Busleitung
Pegel Teilnehmer 1	Pegel Teilnehmer 2	
Dominant	Dominant	Dominant
Dominant	Rezessiv	Rezessiv
Rezessiv	Dominant	Rezessiv
Rezessiv	Rezessiv	Rezessiv

Abbildung 2.1.: CAN-Busanschaltung am Beispiel eines Eindraht-CAN [26]

Der physikalische Aufbau des CAN-Busses, wie er in Abbildung 2.1 beispielhaft für einen Eindraht-CAN dargestellt ist, sieht sogenannte *rezessive* und *dominante* Bits vor. Durch eine Offene Kollektorschaltung (engl.: Open Collector) mit Pullup-Widerstand wird eine sogenannte verdrahtete UND-Verknüfung (engl.: Wired-AND) realisiert. Dies bedeutet, dass sobald ein Teilnehmer einen dominanten Pegel auf der Busleitung anlegt, dieser einen gleichzeitig anliegenden rezessiven Pegel eines anderen Teilnehmers überschreibt. Letzterer muss den gleichzeitigen Zugriff auf den Bus erkennen, da ein von ihm gesendetes rezessives Bit durch ein dominantes Bit überschrieben wurde. Daraufhin stellt er seine Übertragung ein. Eine Übertragung darf erst wieder begonnen werden, wenn die vorherige Nachricht vollständig übertragen ist.

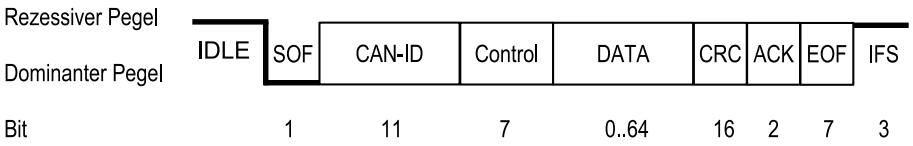


Abbildung 2.2.: Aufbau einer CAN-Nachricht [70]

Für CAN-Nachrichten mit Dateninhalt (siehe Abbildung 2.2) ist das Nachrichtenende so definiert, dass das zweite Bit des ACK Feldes, die End-of-Frame-Kennung (EOF) und der Inter-Frame-Space jeweils rezessiven Pegel führen. Dies bedeutet, dass mindestens 11 aufeinanderfolgende rezessive Bits anliegen. Ein darauf folgendes dominantes Bit – *Start-of-Frame (SOF)* genannt – repräsentiert dann

den Anfang einer neuen Nachricht. Mit dem Start-of-Frame signalisiert ein Teilnehmer, dass die Übertragung einer Nachricht beginnt.

Das für den CAN-Bus spezifizierte Protokoll [53, 54, 56] ist *nachrichtenorientiert*. Dies bedeutet, dass der Datenaustausch nicht durch die Adressierung des Empfängers, sondern durch eine Kennzeichnung der übertragenen Nachricht erfolgt. Der Inhalt einer Nachricht wird durch die *Nachrichtenkennung* (engl.: *Identifier*) gekennzeichnet, so dass jeder Empfänger prüfen kann, ob diese für ihn relevant ist [26].

In einer CAN-Nachricht ist die Nachrichtenkennung, die direkt nach dem Start-of-Frame verschickt wird, zugleich die Priorität für den Buszugriff der Nachricht. Da jede Nachrichtenkennung genau einem Teilnehmer zuzuordnen ist, kann bei gleichzeitigem Buszugriff mehrerer Teilnehmer eine Auflösung des Konflikts ohne Datenverlust erfolgen. Der Teilnehmer, der die Nachricht mit höchster Priorität versenden möchte, erhält exklusiven Buszugriff, da er die Nachrichtenkennung mit den meisten dominanten Bits auf dem Bus anlegt. Die anderen Teilnehmer stellen während der Übertragung fest, dass ein rezessives Bit ihrer Nachrichtenkennung durch ein dominantes überschrieben wurde und stellen daraufhin ihre Übertragung ein. Sobald wieder Ruhezustand auf dem Bus herrscht, kann ein neuer Versuch zur Übertragung der Nachricht erfolgen.

Das Buszugriffsverfahren bevorzugt somit die Übertragung von Nachrichten mit hoher Priorität. Daraus folgt, dass bei priorisierten Nachrichten mit einer geringen Wartezeit zwischen Sendewunsch und Buszugriff gerechnet werden kann. Diese Wartezeit soll im Folgenden *Latenz* genannt werden. Zur Lösung des Problems, dass die Latenz einer Nachricht von ihrer Priorität abhängt und somit nicht für alle Nachrichten garantiert werden kann, wird zwar im CAN-Standard [55] eine Zeitsteuerung beschrieben, diese hat sich aber für CAN-Busse in Fahrzeugen nicht durchgesetzt [70].

Um die Latenz auch für niederpriore Nachrichten im Rahmen zu halten, wird stattdessen in der Regel in vielen Fahrzeugarchitekturen zur *Vernetzung* von Steuergeräten eine zyklische Kommunikation auf dem CAN-Bus implementiert [70, 93]. Das heißt, es wird vorab festgelegt, welche Nachricht mit welcher Zykluszeit versendet wird. Wird eine hochpriore Nachricht beispielsweise alle 10ms verschickt, so bleibt zwischen den Nachrichten Zeit für niederpriore Nachrichten. Dennoch können keine konstanten Latenzen für alle Nachrichten erreicht wer-

den, da die Einhaltung der Zykluszeit Aufgabe des Senders ist und nicht durch den CAN-Bus vorgegeben wird.

Zusätzlich wird bei der Festlegung der Fahrzeugarchitektur darauf geachtet, dass der Bus nur zu einem Bruchteil ausgelastet wird. Durch diese Maßnahme sollen Verletzungen der gewünschten Latenz vermieden oder in engen Grenzen gehalten werden. Subke formuliert dies wie folgt:

> „Um Echtzeitfähigkeit sicher zu stellen, müssen in ereignisgesteuerten CAN-Netzwerken bis zu 80% der vorhandenen Bandbreite für mögliche Spitzen-Buslasten reserviert werden.“ [112]

Reichart fordert für Fahrerassistenzsysteme mit aktivem Eingriff in die Fahrdynamik in [91] eine „sorgfältige Betrachtung der möglichen Latenzzeiten“.

In der Regel überwachen Steuergeräte zur Laufzeit, ob die spezifizierten Zykluszeiten für Empfangsnachrichten eingehalten werden. Je nach Anwendungsfunktion, die auf dem Steuergerät implementiert ist, werden Verzögerungen toleriert. Außerhalb der Toleranz stellt das Steuergerät in der Regel seine Funktion ein.

Daraus ergibt sich aus der Sicht eines Testsystems, das eine Anwendungsfunktion auf einem Steuergerät über CAN stimulieren soll, dass die CAN-Nachrichten innerhalb der vom Steuergerät festgelegten Toleranz geschickt werden müssen. Im Rahmen dieser Arbeit soll der Begriff *Echtzeit* auf dieser Basis definiert werden:

Definition 1. *Echtzeitfähigkeit*
Im Rahmen dieser Arbeit soll der Begriff Echtzeitfähigkeit in Bezug auf Testsysteme so definiert werden, dass ein Testsystem dann echtzeitfähig ist, wenn es innerhalb der festgelegten Toleranz die Ausgangsdaten des Testobjekts einlesen und das Testobjekt mit neuen Eingangsdaten versorgen kann.

Die Ansprüche an das Zeitverhalten eines Steuergeräts für Fahrerassistenzsysteme hängen in der Regel von den zeitlichen Rahmenbedingungen der umgesetzten Fahrerassistenzfunktionen ab. Es ist davon auszugehen, dass je schneller eine Fahrerassistenzfunktion reagieren muss, desto höher die Ansprüche an das Zeitverhalten werden. Bei den in dieser Arbeit aufgeführten Beispielen für Testsysteme bedeutet dies, dass das Fahrerassistenzsteuergerät einen zyklischen Empfang

verschiedener CAN-Nachrichten mit einer Abweichung von $\pm 2\%$ zur spezifizierten Zykluszeit einer Nachicht erwartet. Im Falle einer spezifizierten Zykluszeit von $10ms$ muss die Zykluszeit somit auf $\pm 200\mu s$ eingehalten werden.

2.2. Fahrerassistenzsysteme

Im Fahrzeug eingebettete Systeme, die den Fahrer entlasten und die Fahrsicherheit erhöhen sollen, werden nach Maurer in [73] *Fahrerassistenzsysteme* genannt. Oft wird zwischen „fortschrittlichen Fahrerassistenzsystemen“ (engl.: Advanced Driver Assistance Systems, ADAS) und konventionellen Fahrerassistenzsystemen unterschieden [72, 25]. Erstere zeichnen sich durch eine sensorische Erfassung der Fahrzeugumgebung und durch eine komplexe Signalverarbeitung aus. Im Folgenden werden die fortschrittlichen Fahrerassistenzsysteme betrachtet. Dabei soll die für den Fahrer erlebbare Funktion, die ihn in seiner Fahraufgabe unterstützt, *Fahrerassistenzfunktion* genannt werden. Der Begriff *Fahrerassistenzsystem* wird hingegen für das im Fahrzeug eingebettete System, das eine oder mehrere Fahrerassistenzfunktionen realisiert, verwendet.

Die Arbeit betrachtet Fahrerassistenzsysteme anhand der in Abbildung 2.3 dargestellten Fahrzeugarchitektur. Die skizzierte Architektur entstammt einem anonymisierten Beispiel aus der Industrie. Recht ähnliche Architekturen werden jedoch auch in [139] beschrieben. Das Fahrerassistenzsystem besteht aus mehreren Steuergeräten mit unterschiedlichen Teilaufgaben. Die Steuergeräte aus der Kategorie Sensorik, im weiteren Verlauf *Sensorsteuergerät* genannt, erfassen mittels unterschiedlicher Sensoren die Umgebung des Fahrzeugs. In Abbildung 2.3 sind die Sensorsteuergeräte Radar Fernbereich, Radar Nahbereich und Kamera, wie sie in [91, 4, 136] vorgestellt werden, dargestellt. Außerdem sind beispielhaft Steuergeräte der Kategorie Fahrerassistenzfunktion-Verarbeitung und Aktorik abgebildet.

In Abbildung 2.4 ist in einem Beispiel dargestellt, wie die Steuergeräte der einzelnen Kategorien zusammen wirken. Jedes Sensorsteuergerät (Sensorik) stellt die gewonnenen Informationen über die Fahrzeugumgebung in aufbereiteter Form über die Vernetzung zur Verfügung [40]. Die Informationen werden vom Fahrerassistenzsteuergerät eingelesen und für die Algorithmen der verschiedenen

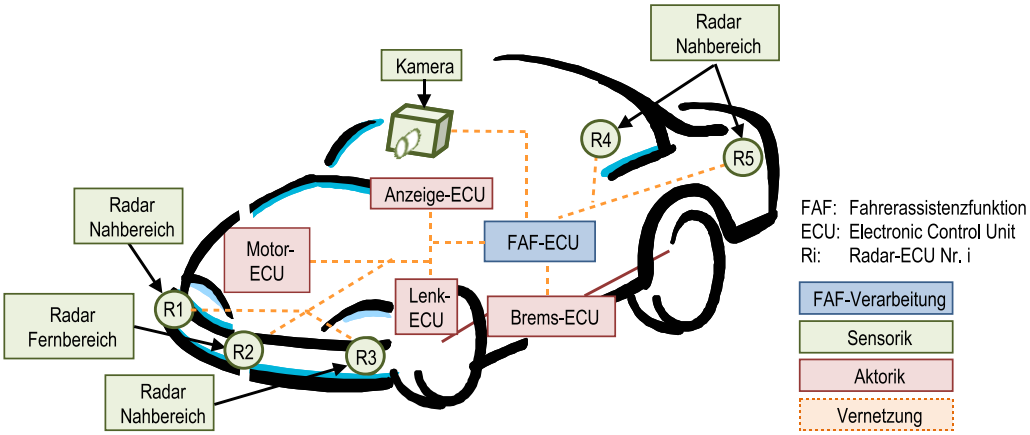


Abbildung 2.3.: Mögliche Steuergerätearchitektur für Fahrerassistenzsysteme

Fahrerassistenzfunktionen verwendet [91]. Als Ergebnis fordert das verantwortliche Fahrerassistenzsteuergerät über die Vernetzung den berechneten Eingriff in das Fahrverhalten des Fahrzeugs von der Aktorik. Dieser Eingriff führt zu einem veränderten Verhalten des Fahrzeugs auf der Fahrbahn und somit zu einer veränderten Situation in der Fahrzeugumgebung (räumlicher Bezug zu Umgebungsobjekten). Diese Veränderungen werden über die Sensorik erneut erfasst.

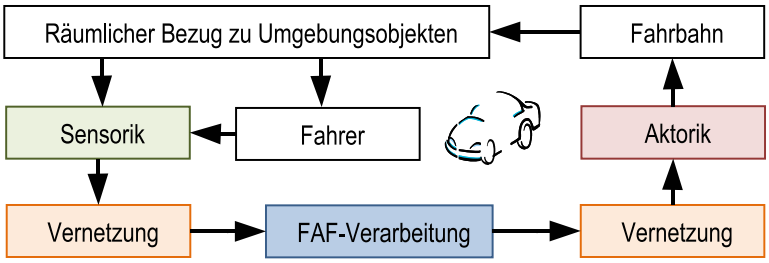


Abbildung 2.4.: Umgebungserfassendes Fahrerassistenzsystem im Fahrzeug

In dieser beispielhaften Architektur ist das Fahrerassistenzsteuergerät für alle auf Basis der Sensorsteuergeräte realisierten Fahrerassistenzfunktionen verantwortlich. Diese Architektur hat zum Vorteil, dass die Sensorik und die Aktorik von einem zentralen Steuergerät getrennt werden [138]. Nach [133] steigt die Anzahl von Fahrerassistenzfunktionen exponentiell über die Zeit und es existiert der Trend, die Fahrerassistenzfunktionen zu bündeln, so dass das Fahrerassistenzsteuergerät optimal genutzt werden kann. Die eingebettete Software dieses Fah-

rerassistenzsteuergeräts soll im Folgenden *Fahrerassistenzsoftware* genannt werden.

An Hand von Beispielen für Fahrerassistenzfunktionen soll gezeigt werden, wie deren Software-Anteile in einem Steuergerät realisiert werden können. Relevant für diese Arbeit sind insbesondere der Abstandsregeltempomat sowie der Notbrems- und der Totwinkelassistent, da diese für die experimentellen Untersuchungen in Kapitel 6 verwendet werden.

Derzeit sind unter anderen folgende Fahrerassistenzsysteme am Markt verfügbar:

- *Antiblockiersystem* (ABS): Dauerhaftes Blockieren der Räder soll verhindert werden [94].

- *Elektronisches Stabilitätsprogramm* (ESP): Ein Schleudern oder Umkippen des Fahrzeugs soll durch das ESP verhindert werden [86].

- *Abstandsregeltempomat* (engl.: Adaptive Cruise Control (ACC)): Der Abstand zum vorausfahrenden Fahrzeug wird durch das System nahezu konstant gehalten [46]. Als herstellerübergreifende Referenzen existieren die Normen ISO 15622 [57] für die Standardfunktionalität mit eingeschränktem Geschwindigkeitsbereich und ISO 22179 [51] für die Erweiterung auf eine Regelung bis zum Stillstand.

- *Notbremsassistent*: Bei der Gefahr eines Auffahrunfalls bremst das Fahrzeug autonom [98, 10].

- *Einparkassistent*: Das Fahrzeug parkt teil- oder vollautomatisch in eine Parklücke ein [60].

- *Spurhalteassistent* (engl.: Lane Departure Prevention System (LDP)): Durch aktive Lenkeingriffe wird das Fahrzeug innerhalb seiner Spur gehalten [33, 107].

- *Totwinkelassistent* (engl.: Blindspot Monitoring (BSM)): Wenn die Gefahr einer Kollision mit einem Fahrzeug im Toten Winkel des Rückspiegels be-

steht, warnt der Totwinkelassistent den Fahrer. In zukünftigen Ausprägungen soll es möglich sein, dass das System einen Unfall aktiv vermeidet [5].

Das Fahrzeug, in dem ein Fahrerassistenzsystem verbaut ist, soll im weiteren Verlauf *Systemfahrzeug* genannt werden. Die Fahrzeuge in der Umgebung des Systemfahrzeugs, auf die sich eine Fahrerassistenzfunktion bezieht, sollen im Rahmen dieser Arbeit *Objektfahrzeuge* genannt werden.

Die Drei-Ebenen-Hierarchie der Fahraufgaben besteht nach Donges [23] aus den menschlichen Fahraufgaben „Navigation“, „Führung“ und „Stabilisierung“. Diese können nach Rasmussen in [88] unterschiedlich stark durch die drei verschiedenen Verhaltensweisen „wissensbasiert“, „regelbasiert“ und „fertigkeitsbasiert“ durchgeführt werden [24, 139]. Für die Stabilisierung eines Fahrzeugs ist primär ein „fertigkeitsbasiertes“ Verhalten gefordert. Die Führung eines Fahrzeugs verlangt alle drei Verhaltensweisen, während die Navigation in der Regel rein „wissensbasiert“ stattfindet. Donges verknüpft die Fahraufgaben zusätzlich mit Zeithorizonten innerhalb derer der Fahrer auf Veränderungen in einer der drei Ebenen reagieren muss. Für die Navigation wird ein Bereich von mehreren Stunden bis Minuten genannt. Der Zeithorizont für eine Führungsaufgabe liegt bei wenigen Minuten bis Sekunden, während die Stabilisierung in der Regel innerhalb von weniger als einer Sekunde zu geschehen hat. Analog zur Drei-Ebenen-Hierarchie können die aufgelisteten Fahrerassistenzsysteme den drei Fahraufgaben zugeordnet werden. Die Systeme ABS und ESP helfen bei der Stabilisierung des Fahrzeugs, während die anderen den Fahrer bei seiner Führungsaufgabe unterstützen.

Insbesondere die zeitlichen Anforderungen an Fahrerassistenzsysteme zur Unterstützung der Führungsaufgabe stellen eine wichtige Anforderung des später gezeigten Tests einer Fahrerassistenzfunktion dar.

In Abbildung 2.5 ist eine mögliche Architektur von Fahrerassistenzsoftware in einem Fahrerassistenzsteuergerät dargestellt. Die drei dargestellten Fahrerassistenzfunktion zur Unterstützung bei der Fahrzeugführung (ACC, BSM und LDP) haben jeweils eine eigene *Sensorfusion* [19]. Nach Gern in [34] ist es Aufgabe der Sensorfusion, die Daten der Sensorsteuergeräte möglichst intelligent zu kombinieren. Die verschiedenen Fahrerassistenzfunktionen nutzen dabei unterschiedliche Teilmengen der verfügbaren Sensorinformationen, so dass Informationen möglichst komplementär je Fahrerassistenzfunktion vorliegen [58]. Der Abstands-

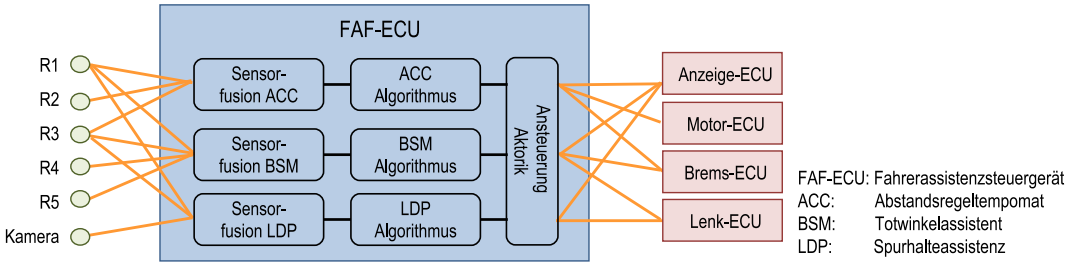


Abbildung 2.5.: Software für verschiedene Fahrerassistenzfunktionen zur Fahrzeugführung [23]

regeltempomat nutzt die vorderen Radarsensoren für den Nahbereich (R1, R3) und den Fernbereichsradar (R2). Der Totwinkelassistent verwendet die Daten der vorderen und hinteren Nahbereichradarsensoren (R1, R3, R4, R5), während der Spurhalteassistent auf den Radarinformationen von R1 und R3 sowie auf den Daten der Kamera basiert. Die jeweiligen Algorithmen der Fahrerassistenzfunktionen verwenden danach die so individuell fusionierten Eingangsdaten und melden ihre Ergebnisse an das Softwaremodul zur Ansteuerung der Aktorik. Hier werden die Anforderungen von Fahrdynamikeingriffen priorisiert und an die entsprechenden Aktorsteuergeräte übermittelt [91, 13].

Für eine korrekte Funktion der Fahrerassistenzfunktion wird nicht nur ein korrektes Übertragen der Daten – und somit der Stärke eines Eingriffs in die Fahrdynamik – gefordert, sondern insbesondere auch, dass der Eingriff zum richtigen Zeitpunkt erfolgt.

2.3. Funktionsorientierter Test von eingebetteten Systemen

Aufgabe des Testens ist es zu überprüfen, ob ein Produkt den gestellten Anforderungen entspricht. Testen weist dabei nicht die Abwesenheit von Fehlern sondern lediglich deren Anwesenheit nach. Aufgabe des Testens ist es also, Fehler zu finden [22].

Zur systematischen Vorgehensweise werden in der Regel *Testmethoden* verwendet. Nach Spillner et al. in [110] ist eine Testmethode ein planmäßiges und regelbasiertes Vorgehen zur systematischen Ermittlung von Testfällen. Gängige Testmethoden können nach [66] zunächst in *statische* und *dynamische Testmethoden* unterteilt werden. Bei ersteren kommt die zu testende Software nicht zur Ausführung sondern wird über analysierende Methoden auf Fehler untersucht [130]. Beispiele hierzu sind Reviews oder statische Codeanalysen. Für diese Arbeit haben die statischen Testmethoden keine Relevanz, da bei dem vorgestellten Testsystem die eingebettete Software im Steuergerät zur Ausführung kommt. Auf diesen Fall beziehen sich die dynamischen Testmethoden, die zum Ziel haben, während der Ausführung Fehler zu finden [81]. Wegener bezeichnet das dynamische Testen in [137] als „die wichtigste analytische Methode, um die Qualität von Echtzeitsystemen sicherzustellen“.

Eine für den weiteren Verlauf der Arbeit wichtige dynamische Testmethode ist der *funktionsorientierte Test*. Nach Veenedaal in [130] basiert der funktionsorientierte Test auf einer Analyse der funktionalen Spezifikation eines Systems oder eines Teilsystems. Dabei werden die von der Spezifikation geforderten Funktionen des Systems zur Ausführung gebracht und deren Ergebnisse gegen die Spezifikation geprüft. Dabei wird das Objekt, das die geforderte Funktion im System erfüllen soll, oft als *Testobjekt* bezeichnet. In der Regel findet der funktionsorientierte Test von Software auf der Zielhardware im *Blackbox-Verfahren* statt. Der Test erfolgt dabei ohne Kenntnis der internen Eigenschaften des Testobjekts.

In dieser Arbeit bezieht sich der funktionsorientierte Test eines Fahrerassistenzsystems auf den Test der eingebetteten Software des Fahrerassistenzsteuergeräts, das die zu testende Funktion erbringen soll. Das Testobjekt ist somit eine Fahrerassistenzfunktion, die im Fahrerassistenzsteuergerät ausgeführt wird.

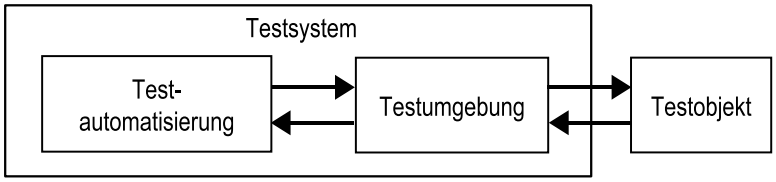


Abbildung 2.6.: Testsystem, bestehend aus Testumgebung und Testautomatisierung

Die zur Durchführung von Tests benötigte Umgebung für das Testobjekt wird *Testumgebung* genannt [130]. Sie besteht unter anderem aus Hardware, Simulationen, Softwarewerkzeugen und weiteren Elementen, die für den Test benötigt werden [48]. Ein Testsystem besteht in dieser Arbeit aus der Testumgebung und aus Softwareelementen, die für den Testbetrieb notwendig sind. Dies ist insbesondere Software zur automatischen Abarbeitung von Testdaten und zur Auswertung der Reaktionen des Testobjekts. In Abbildung 2.6 wird diese Software als *Testautomatisierung* bezeichnet. Für diese Arbeit ist es insbesondere wichtig, dass über die Testautomatisierung die Stimulation des Testobjekts mit unterschiedlichem Zeitverhalten möglich ist.

3. Stand der Technik

In diesem Kapitel wird der Stand der Technik zum Test von Fahrerassistenzsystemen beschrieben. Die eingesetzte Technik und deren Defizite werden zusammengefasst und motivieren die Entwicklung eines neuartigen Steuergerätetestsystems.

Das Kapitel 3.1 analysiert den Entwicklungsprozess von Fahrerassistenzsystemen in der Fahrzeugindustrie mit dem Fokus auf der Untersuchung des Verhaltens der eingebetteten Software bei unterschiedlichen Anforderungen an das Zeitverhalten in ihrer jeweiligen Umgebung. In Kapitel 3.2 werden die Besonderheiten beim Test von im Fahrzeug eingebetteten Systemen für Fahrerassistenzfunktionen erläutert. Insbesondere wird gezeigt, wie Fahrerassistenzsysteme im Fahrzeug getestet werden und wie ein Steuergerät, das die Fahrerassistenzsoftware beinhaltet, an einem Steuergerätetestsystem getestet wird. Im Speziellen wird die Umsetzung des Zeitverhaltens in den bestehenden Testsystemen betrachtet. In Kapitel 3.3 werden die Grenzen derzeitiger Testsysteme dargestellt und in Kapitel 3.4 wird der Stand der Technik zusammengefasst.

3.1. Entwicklungsprozess von Fahrerassistenzsystemen

Dieses Kapitel beschreibt den derzeitig meist anzutreffenden Prozess für die Entwicklung von eingebetteten Systemen – insbesondere Fahrerassistenzsysteme – in der Fahrzeugindustrie. Der Fokus bei der Beschreibung liegt auf der Vorgehensweise beim funktionsorientierten Test der eingebetteten Software und der Untersuchung ihres zeitlichen Verhaltens. Für sicherheitskritische Fahrerassistenzsysteme sind Vorgaben an den Entwicklungsprozess beispielsweise in der Norm DIN EN 61508-4 [50] und in Zukunft in der ISO 26262 [52] gegeben.

Nach DIN EN 61508-4 [50] ist ein E/E/PE-System ein System zur Steuerung, zum Schutz oder zur Überwachung, basierend auf einem oder mehreren elektrischen/elektronischen/programmierbaren elektronischen Geräten. Eingeschlossen sind dabei alle Elemente des Systems wie z. B. Energieversorgung, Sensoren und andere Eingabegeräte, Datenverbindungen und andere Kommunikationswege sowie Aktoren und andere Ausgabeeinrichtungen. Weiterhin muss nach [50] ein solches System so entwickelt werden, dass mit technischen Mitteln und durch deren Entwicklungsprozess das Risiko, Menschenleben durch eine Fehlfunktion des Gesamtsystems zu gefährden, auf ein gesellschaftlich vertretbares Minimum reduziert wird.

Ein möglicher Entwicklungsprozess für eine Anwendungsfunktion eines E/E/PE-Systems in der Fahrzeugindustrie wird von Bühler in [9] beschrieben. Eine Anwendungsfunktion beschreibt hier eine Leistung eines Systems. Bei Fahrerassistenzsystemen ist diese Leistung des Systems durch den Kunden erlebbar und wird im Rahmen dieser Arbeit als *Fahrerassistenzfunktion* bezeichnet (siehe Kapitel 2.2).

Bis zur Trennung der Entwicklungspfade für Hardware und Software werden von Bühler in [9] die Zerlegungsebenen

- Fahrzeug,
- Steuergerät,
- Steuergeräte-Hardware und Steuergeräte-Software

im Rahmen einer hierarchischen Dekomposition des Systems nach dem V-Modell [128, 31, 41] vorgeschlagen. Danach können verschiedene Methoden für die Umsetzung der Fahrerassistenzfunktion gewählt werden. Zum einen kann die Software klassisch als C-Code [20] entwickelt werden oder werkzeuggestützt mit blockorientierten Modellierungswerkzeugen [92, 89, 27]. Der programmierte Algorithmus kann danach für eine Zielhardware compiliert werden und beispielsweise im Rahmen eines Betriebssystems ausgeführt werden. Ein Trend ist, die Basisfunktionalität eines Steuergerätes und die Anwendungsfunktion zu trennen. Ein gängiger Ansatz ist hierzu das Autosar-Framework [146]. Unabhängig von der gewählten Methode für die Softwareentwicklung ist eine Aussage über das Zeitverhalten der eingebetteten Software nur eingeschränkt möglich, wenn diese

nicht auf der Zielhardware ausgeführt wird. Die Zielhardware umfasst hierbei den eingesetzten Prozessor inklusive angeschlossener Schaltkreise, wie Speicher oder CAN-Controller in der später verbauten Zusammensetzung. Erst wenn bei der Hardware-/Sofware-Integration die entwickelte Software auf der Zielhardware läuft, kann das Zeitverhalten der eingebetteten Software auf dem Steuergerät überprüft werden.

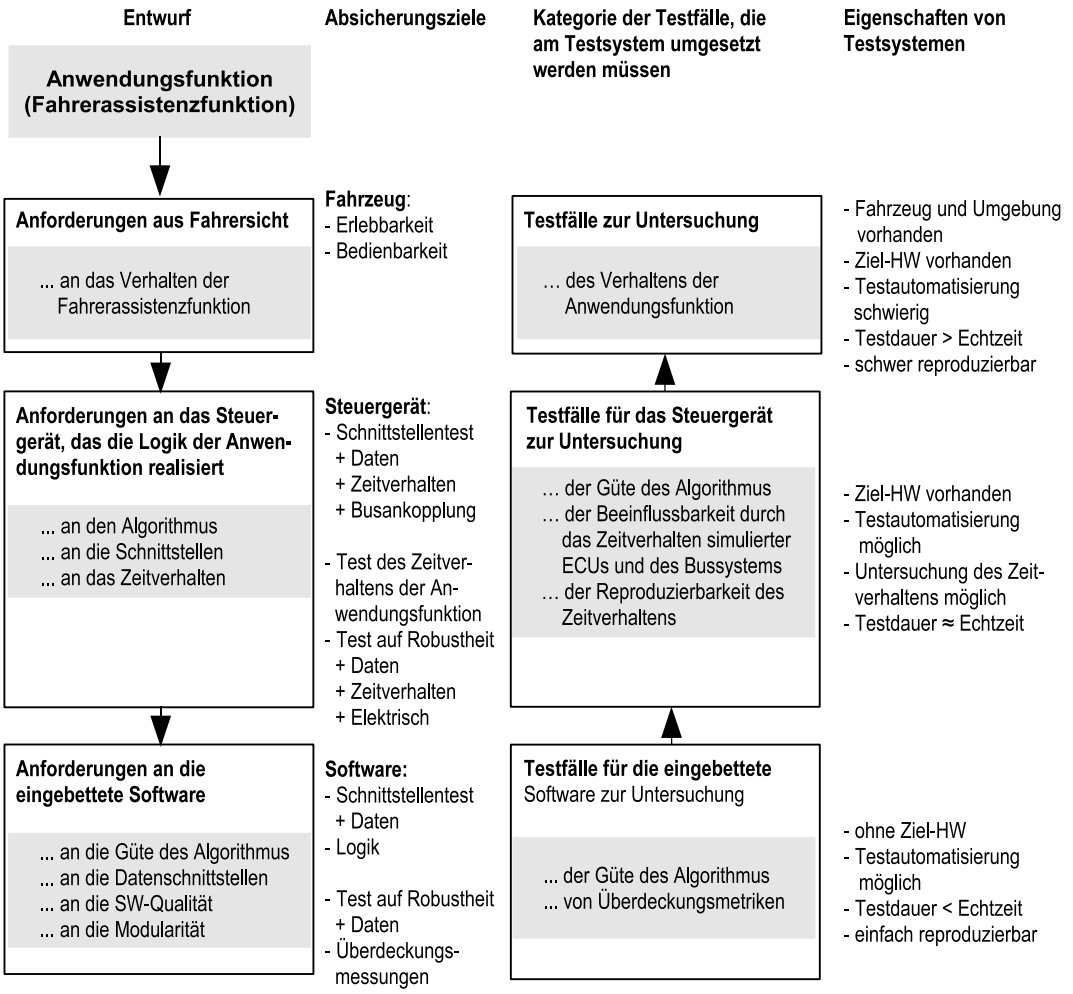


Abbildung 3.1.: Entwicklungsprozess aus der Testsicht

Da ein Fokus von Steuergerätetestsystemen – und somit dieser Arbeit – auf der Überprüfung des Zeitverhaltens von eingebetteten Systemen liegt, ist in Abbil-

dung 3.1 als Ausschnitt aus dem gesamten Entwicklungsprozess die Zerlegungsebene Steuergerät zwischen den Ebenen Fahrzeug und Steuergeräte-Software dargestellt. Die Abbildung erläutert auf Basis von eigenen Industrieerfahrungen den Entwicklungsprozess aus Entwurfs- und Testsicht und versucht, die verschiedenen Aspekte des Tests von eingebetteten Systemen zusammen zu fassen.

Die Darstellung in Abbildung 3.1 ist als iteratives Modell zu sehen [93]. Es kann jederzeit wieder „zurück“ gesprungen werden. In der Regel werden die dargestellten Schritte in mehreren Iterationsschleifen durchlaufen. Im von Simmes in [109] vorgeschlagenen Entwicklungsprozess für im Fahrzeug eingebettete Systeme ist es das Ziel, in einem Rapid Prototyping-Prozess möglichst frühzeitig zu einer Demonstration der geforderten Fahrerassistenzfunktion im Fahrzeug zu kommen [79, 80, 83, 12, 61]. Nach jeder Iteration wird der entstandene Prototyp zur Verbesserung der Spezifikation und als Referenz für die nächste Iteration verwendet. Außerdem können Funktionalität und Fehlerbehandlung hinzugefügt werden. Somit gelangt man schrittweise zu einem serienreifen System. Jeder entstandene Prototyp muss verifiziert[1] werden. Aus Effizienzgründen schlägt Simmes [109] vor, jedes Entwicklungsergebnis so früh wie möglich zu testen. Nach Frühauf et al. in [30] ist eine frühest mögliche Verifizierung sinnvoll, denn die Fehlerbehebungskosten steigen exponentiell mit der Zeit, über die sich ein Fehler im System befindet.

3.1.1. Entwurfsprozess

Die Entwicklung – in Abbildung 3.1 in der linken Spalte dargestellt – beginnt mit der Festlegung der gewünschten Fahrerassistenzfunktion. Hieraus werden Anforderungen an das Fahrzeug aus Sicht des Fahrers abgeleitet. Diese Anforderungen auf Fahrzeugebene spezifizieren das gewünschte Verhalten eines Fahrzeugs, welches die zu entwickelnde Fahrerassistenzfunktion enthält. Dabei steht das durch den Fahrer erlebbare Verhalten des Fahrzeugs im Vordergrund. Die Anforderungen sollen nach [95] so spezifiziert sein, dass sie nachprüfbar sind. D. h. bei der Anforderungsdefinition müssen sofort auch die Testfälle – auf dieser

[1] Brockhaus erklärt diesen Begriff mit seiner lateinischen Herkunft: „verificare, zu lat. verus = wahr, richtig u. facere = machen (...) durch Überprüfen die Richtigkeit einer Sache bestätigen“ [104].

Ebene als *Fahrmanöver*[2] – spezifiziert werden.

Im Rahmen der Systemanalyse auf der Zerlegungsebene Fahrzeug wird festgelegt, aus welchen logischen Systemfunktionen sich eine Fahrerassistenzfunktion des Fahrzeugs zusammensetzt. Im nächsten Entwurfsschritt wird entschieden, wie sich die geforderten Funktionsbeiträge auf die Hardware- und Softwaremodule aufteilen. Dieser Schritt setzt sich aus der Hardware- und Softwarearchitektur des Fahrzeugs zusammen und liefert als Ergebnis unter anderem die Anforderungen an ein Steuergerät. Über die Hardwarearchitektur des Fahrzeugs wird festgelegt, durch welche Hardwarekomponenten, wie z. B. Sensor-ECUs, Aktor-ECUs oder reine Rechen-Steuergeräte (z. B. FAF-ECU in Abbildung 2.5) die Fahrerassistenzfunktion realisiert wird.

Außerdem wird die Topologie der Vernetzung der Hardwarekomponenten festgelegt und die zugehörige Kommunikationsspezifikation erstellt. Über die Softwarearchitektur wird festgelegt, welcher in Software umgesetzte Funktionsbeitrag durch welche Hardwarekomponente erbracht wird. Nach dieser Festlegung ergibt sich die benötigte Kommunikation zwischen den einzelnen Komponenten und somit die Kommunikationsinhalte des Netzwerks im Fahrzeug.

Wird aus der entstandenen Architektur ein Steuergerät freigeschnitten, so können dessen bereitzustellende Funktionalität und dessen Schnittstellen sowie das jeweils geforderte Zeitverhalten in einem Anforderungsdokument festgehalten werden. Auf Basis der Anforderungsdokumentation für das Steuergerät und dessen Schnittstellen zu anderen Komponenten können die Testfälle für das Steuergerät abgeleitet werden.

Als nächster Entwicklungsschritt erfolgt die Systemanalyse und der Systementwurf des Steuergeräts auf Basis der übergebenen Anforderungen. Im Systementwurf des Steuergeräts wird festgelegt, wie die Anforderungen durch das Zusammenspiel von Steuergeräte-Hardware und Software umgesetzt werden. Danach können Hardware und Software getrennt entwickelt werden. Über die Anforderungen an die Software und die Systemanalyse werden die logischen Systemfunktionen und deren Kommunikation festgelegt. Die Architektur der Software

[2] Unter dem in der Praxis verwendeten Begriff *Fahrmanöver* ist das gezielte Fahren eines Manövers mit einem Fahrzeug gemeint. Nach Brockhaus ist ein Manöver die taktische (planvolle) Bewegung eines Autos [104].

legt fest, wie die einzelnen Softwaremodule durch Betriebssystemprozesse und Interrupt-Service-Routinen auf dem Steuergerät zusammenwirken. Ein Vorteil der Zerlegung der Software in unabhängige Teilkomponenten ist, dass diese danach parallel von mehreren Entwicklungsteams implementiert werden können. Für den Test der einzelnen Softwaremodule werden parallel Testfälle formuliert, die das gewünschte Verhalten eines Softwaremoduls überprüfen.

3.1.2. Der Integrations- und Testprozess

In Abbildung 3.1 sind neben den Entwurfsschritten in den gewählten Zerlegungsebenen auch mögliche Absicherungsziele und Kategorien von Testfällen sowie die Eigenschaften der korrespondierenden Testsysteme skizziert.

Nachdem über den Systementwurf festgelegt wurde, wie die Anforderungen an das Fahrzeug durch dessen Teilkomponenten (Sensorik, Verarbeitung, Aktorik) und der darin enthaltenen Software umgesetzt werden, kann mit der Implementierung der Software begonnen werden. Bereits parallel zu den Anforderungen werden – wie in Abbildung 3.1 gezeigt und vom V-Modell [31] gefordert – die Absicherungsziele (Beispiele in Spalte 2) und die dazugehörigen Testfälle für die einzelnen Zerlegungsebenen festgelegt. Absicherungsziele in Kombination mit den Kategorien der ermittelten Testfälle (dargestellt in Spalte 3) können als Anforderungen an die bei der schrittweisen Integration benötigten Testsysteme gesehen werden. Die Eigenschaften der jeweiligen Zerlegungs- bzw. Integrationsebene beeinflussen daher die geforderten Eigenschaften des Testsystems (siehe Spalte 4).

Basierend auf den Anforderungen an die Testsysteme kann parallel zur Implementierung der Funktionssoftware mit der Entwicklung und Bereitstellung von geeigneten Testsystemen begonnen werden. Oft werden diese von separaten Testverantwortlichen angefordert und bedient. Die Entwicklung und Bereitstellung erfolgt meist durch unabhängige Teams, Abteilungen oder durch externe Firmen. Ziel ist es, dass mit dem Beginn von Integration und Test im Entwicklungsprozess genügend Testsystemressourcen mit den geforderten Eigenschaften zur Verfügung stehen.

Während der Integrations- und Testphase werden nach der Implementierung beginnend von der niedrigsten Zerlegungsebene die einzelnen Softwaremodule zur Steuergeräte-Software integriert. Die jeweiligen Produkte werden gegen die Anforderungsdefinitionen geprüft. Vor allem die Einhaltung der Schnittstellen aus Datensicht kann durch statische und dynamische Prüfverfahren untersucht werden.

Des Weiteren kann die geforderte Funktion der Software durch funktionsorientierte Tests untersucht werden. Da die Software noch nicht auf der späteren Hardwareplattform ausgeführt wird, ist eine Aussage zum Zeitverhalten der untersuchten Funktion nur bedingt möglich. Wenn die Software im Sourcecode vorliegt, können weiterhin Überdeckungsmessungen durchgeführt werden und die Einhaltung von geforderten Codierungsrichtlinien untersucht werden. Automatisch durchgeführte Software-Tests sind von großem Vorteil, weil dadurch nach einer Änderung ohne manuellen Aufwand erneut getestet werden kann. Da die zu testende Software nicht auf der Steuergeräte-Hardware läuft, ist die Ausführungsdauer eines Testfalls meist kürzer als in Echtzeit und vor allem durch die Leistungsfähigkeit des Testsystems bestimmt.

Sobald die Softwareanteile auf das Steuergerät appliziert sind, kann das Zeitverhalten der Applikation und der Busankopplung getestet werden. Wird das Datenverhalten bevorzugt auf der Softwareebene untersucht, so muss auf der Ebene „Steuergerät“ zumindest sichergestellt werden, dass die Daten von der Software auch korrekt und zum richtigen Zeitpunkt über die Busschnittstelle übertragen werden. Die Buskommunikation zwischen Testsystem und Steuergerät muss daher das in den Anforderungen an das Steuergerät festgelegte Zeitverhalten aufweisen. An das Testsystem ergibt sich somit Anforderung 1.

Anforderung 1. *Echtzeitfähigkeit*
Das Testsystem muss in Echtzeit nach Definition 1 mit dem Steuergerät kommunizieren.

Im Sinne von Robustheitstests wird das Testobjekt neben dem Test auf inhaltliche Abweichungen der Eingangsdaten meist auch auf die Reaktion bezüglich zeitlicher Abweichungen der Eingangsdaten untersucht. Zur Stimulation von verschiedenen zeitlichen Abweichungen muss das Zeitverhalten des Testsystems flexibel gestaltbar sein, wie es Anforderung 2 fordert.

Anforderung 2. *Flexible Beeinflussung des Zeitverhaltens*
Das Zeitverhalten des Testsystems auf dem CAN-Bus muss flexibel beeinflussbar sein. Im Rahmen dieser Arbeit bedeutet dies, dass CAN-Nachrichten gezielt innerhalb des 10ms Zeitfensters der zyklischen Kommunikation platziert werden können und dass die Zeitbasis des Testsystems an die Zeitbasis des Testobjekts angepasst werden kann.

Auch auf dieser Integrationsstufe ist vollautomatisiertes Testen möglich. Da die Tests zusammen mit der Zielhardware ausgeführt werden, muss die Stimulation des Testobjekts in Echtzeit erfolgen und die Testdauer ist somit abhängig von der Ausführungszeit der zu testenden Funktion. Hinzu kommen noch die Zeiten, die benötigt werden, um das Testobjekt vor und nach einem Testlauf in einen definierten Zustand zu bringen und eventuell die für die Testauswertung benötigten internen Informationen auszulesen. Die Gesamtdauer eines Testfalls auf der Zielhardware ist somit etwas größer als die Ausführungszeit der im Fahrzeug geforderten Funktion.

Im Rahmen des Fahrzeugtests wird das Steuergerät mit der zu testenden Funktion in ein Fahrzeug mit realer Sensorik und Aktorik integriert. Die entwickelte Fahrerassistenzfunktion ist in dieser Integrationsstufe vom Fahrer erlebbar. Jetzt werden meist Kundenerlebbarkeit und Bedienbarkeit der Fahrerassistenzfunktion untersucht [72, 11, 44]. Die zu testende Funktion wird auch hier in Realzeit ausgeführt. Der zeitliche Aufwand, die Tests auf einer Teststrecke vorzubereiten und durchzuführen, ist deutlich höher als an einem Testsystem [101].

Der Ablauf im beschriebenen Integrations- und Testprozess für im Fahrzeug eingebettete Systeme mit Realzeitanforderungen zeigt, dass es ein unerlässlicher Schritt ist, deren Funktionalität in einem Echtzeit-Kontext zu prüfen. Aus Effizienzgründen sollte möglichst frühzeitig und damit in den niedrigen Integrationsstufen mit dem Test der Echtzeiteigenschaften begonnen werden können. Für die iterative Entwicklung im Rapid Prototyping-Verfahren ist es hilfreich, wenn das Testsystem möglichst arbeitsplatznah zur Verfügung steht, so dass ein Softwareentwickler Codeänderungen und Neuentwicklungen mit geringem zusätzlichen Zeitaufwand testen kann. In den Vorgehensmodellen Scrum [105] und Extreme Programming [8, 38] wird eine *fortlaufende Integration* (engl.: Continuous Integration) verlangt, so dass geänderte Software-Stände über Nacht oder direkt nach einer Änderung automatisiert integriert und getestet werden. Damit ist jederzeit eine Aussage über den aktuellen Entwicklungsstand möglich.

Bei einem Fahrzeughersteller werden zahlreiche Fahrerassistenzfunktionen arbeitsteilig entwickelt [13]. Bei einem großen Fahrzeughersteller führt dies zu über einhundert einzelnen Softwaremodulen, die innerhalb einer Entwicklungsabteilung von durchaus sechzig bis achtzig Entwicklern parallel bearbeitet werden. In Abbildung 3.2 ist skizziert, wie im Laufe des Entwurfs die Anforderungen auf Fahrzeugebene bis auf die Ebene einzelner Softwaremodule heruntergebrochen werden. Diese Softwaremodule werden von einzelnen Entwicklern oder von Teams implementiert. Jedes Modul wird über Software-Tests von den Entwicklern getestet. Um auch das Zeitverhalten einzelner Module untersuchen zu können, kann im Rahmen einer Vorabintegration das Softwaremodul auf der Zielhardware getestet werden. Da die Entwicklung der Softwaremodule in der Regel auf eine bestimmte Fahrzeugbaureihe ausgerichtet ist, haben sie meist ähnliche Zieltermine für die Fertigstellung wie diese Baureihe. Daraus resultiert, dass vor diesem Termin die entsprechenden Testsysteme für die einzelnen Softwaremodule zur Verfügung stehen müssen, so dass die parallel entwickelten Module auch parallel getestet werden können.

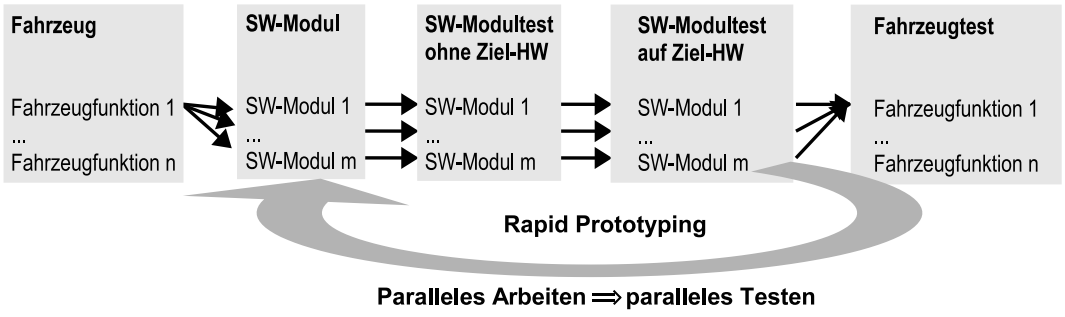


Abbildung 3.2.: Begründung für die Notwendigkeit zur Bereitstellung von vielen Testsystemen

Idealer Weise stehen nicht wenige Testsysteme in einem entfernten Labor zur Verfügung, sondern jeder Entwickler hat ein Testsystem an seinem Arbeitsplatz, so dass er jede Softwareänderung zeitnah auf der Zielhardware untersuchen kann. Unvollständig durchgeführte Tests oder Testergebnisse, die zu spät zur Verfügung stehen, führen bei der nachfolgenden Integration in ein Fahrzeug und bei der Fehlersuche zu einem erhöhten Aufwand [13]. Zur optimalen Anpassung der entwickelten Software an ein Fahrzeug und um für den nächsten Entwicklungszyklus die Anforderungen an das zu entwickelnde System zu optimieren, wird die Software am Ende eines Entwicklungszyklus im Fahrzeug erprobt [109, 35, 58, 10]. Hierzu verlegen die Entwickler oftmals ihren Arbeitsplatz für mehrere Tage

an spezielle Testgelände [10]. Wenn eine neue Softwareversion auf dem Testgelände erstellt wird, kann idealer Weise auch hier zunächst ein Testlauf an einem Testsystem erfolgen, bevor die Software im Fahrzeug getestet wird. Hierfür ist es nötig, dass das Testsystem auch an diesem Ort verfügbar ist. Für ein an mehreren Orten verfügbares System kann nach Brockhaus der Begriff *ubiquitär* verwendet werden [104].

Aus den aufgeführten Ansätzen, das zu entwickelnde System möglichst frühzeitig, kontinuierlich und an verschiedenen Lokationen zu testen, ergeben sich folgende Anforderungen an ein geeignetes Testsystem:

Anforderung 3. *Preisgünstig*
Das Testsystem muss so preisgünstig sein, dass viele Instanzen betrieben werden können und jeder Entwickler jederzeit Zugriff auf ein Testsystem hat. Die für das Testsystem zusätzlich zu einem Arbeitsplatzrechner benötigten Teile dürfen für die praktische Umsetzbarkeit daher nicht mehr kosten als der zehnfache Preis eines Arbeitsplatzrechners.[3]

Anforderung 4. *Ubiquitär*
Das Testsystem muss auf Basis eines Arbeitsplatzrechners – hierbei kann es sich auch um einen Laptop handeln – aufgebaut sein, so dass der Entwickler das Testsystem zusammen mit seinem Arbeitsplatzrechner überall hin mitnehmen kann und somit immer vor Ort testen kann.

Erfüllt ein Testsystem diese Anforderungen, dann hat jeder Entwickler jederzeit Zugriff auf ein Testsystem. Dies spart wertvolle Zeit im Entwicklungsprozess, da zeitliche Engpässe an den Testsystemen nicht mehr auftreten. Außerdem können durch den parallelen Einsatz von mehreren automatisierten Testsystemen kontinuierliche Tests im Rahmen einer fortlaufenden Integration durchgeführt werden.

[3] Da im Jahr 2011 zum Zeitpunkt der Arbeit leistungsstarke Rechner in großen Unternehmen für ungefähr 500 Euro inklusive Wartung erhältlich sind, sollen sich die zusätzlichen Hardware-Kosten für das Testsystem auf maximal 5000 Euro belaufen.

3.2. Testen von Fahrerassistenzsystemen

In Kapitel 3.1 wurde der Entwicklungsprozess von Fahrerassistenzsystemen und dessen Auswirkung auf die Eigenschaften von Testsystemen erörtert. Im Folgenden sollen Testsysteme für die Integrationsstufen „Fahrzeug“ und „Steuergerät“ in Bezug auf den Test von Fahrerassistenzfunktionen erläutert werden.

Bei der Entwicklung von eingebetteten Systemen ist es nicht nur wichtig, auf eine korrekte Implementierung der eingebetteten Software Wert zu legen, sondern auch Anforderungen hinsichtlich der Sicherheit, der Benutzbarkeit, der Zuverlässigkeit, des Energieverbrauchs und der Kosten zu erfüllen. Diese Kriterien müssen im Entwicklungsprozess sichergestellt und im Rahmen der Integrationsphase getestet werden. Eine häufig angewandte Testmethode ist der funktionsorientierte Test des Produktes einzelner Integrationsschritte. Wird das eingebettete System im Fahrzeug getestet (Kapitel 3.2.1), befindet es sich in seiner Zielumgebung, für die die Testergebnisse meist aussagekräftig sind. Die Testfälle für Fahrerassistenzsysteme sind aber oft sehr komplex, da mehrere Fahrzeuge beteiligt sind, die gesteuert werden müssen und deren Daten in einer zur Testauswertung geeigneten Form vorliegen müssen (Kapitel 3.2.2).

Bei einem isolierten Test der eingebetteten Software an einem Steuergerätetestsystem können im Fahrzeug aufgezeichnete Daten zur Stimulation der eingebetteten Software verwendet werden (Kapitel 3.2.3), sofern deren Eingangsdaten keine Abhängigkeit über die Umgebung zu den Ausgangsdaten haben. Ist letzteres der Fall, muss der fehlende, für den Test relevante Teil der Umgebung durch eine Umgebungssimulation ersetzt werden (Kapitel 3.2.4). Die Aussagekraft eines solchen Tests ist abhängig von der Realitätsnähe der verwendeten Umgebungssimulation und muss bei der Interpretation der Testergebnisse berücksichtigt werden.

3.2.1. Testen von Fahrerassistenzsystemen im Fahrzeug

Soll ein Fahrerassistenzsystem im Fahrzeug getestet werden, so ist es erforderlich, dass dessen komplette Umgebung dargestellt wird. In Kapitel 2.2 ist aufgeführt, dass diese Umgebung aus den Teilkomponenten des Fahrerassistenzsys-

tems sowie dem Fahrzeug selbst besteht. Zusätzlich sind ein Fahrer und eine geeignete Fahrzeugumgebung (Fahrbahn, umgebende Fahrzeuge mit Fahrern) notwendig.

Dieser Aufwand macht den Test eines Fahrerassistenzsystems im Fahrzeug zeitintensiv und teuer. Mit den sich derzeit in der Entwicklung befindenden Fahrerassistenzsystemen steigt die Komplexität der geforderten Fahrzeugumgebung und der Testfälle stark an, so dass selbst für geübte Testfahrer bei der Durchführung der Manöver teilweise eine enorme Gefahr für Leib und Leben entsteht. Als Beispiel sei ein Überholassistent genannt [45, 58, 59]. Dieses Fahrerassistenzsystem soll das Systemfahrzeug, das zum Überholen auf die Fahrbahn des Gegenverkehrs gewechselt ist, bei einer drohenden Kollision mit dem Gegenverkehr abbremsen und wieder zurück auf die richtige Fahrspur lenken, sofern dies auf Grund der Umgebungssituation noch möglich ist.

Einen Ansatz, **zusätzlichen Nutzen aus dem in dieser Arbeit vorgestellten Konzept für ein Testsystem und der zugehörigen Hardware zu ziehen und den Fahrer bei der Durchführung komplexer Fahrmanöver im Fahrzeug bestmöglich zu unterstützen, ist zusammen mit K. Hünlich in [42] entstanden und in [123, 122] patentiert**. Da ein menschlicher Fahrer ein gefordertes Fahrmanöver in der Regel ungenauer ausführt als ein simulierter Fahrer, schlagen die Autoren vor, durch zusätzliche Überwachungsbedingungen im Testfall dem Fahrer sofort mitzuteilen, wenn er eine Überwachungsbedingung verletzt. Beispielsweise kann die Vorgabe „auf 60km/h beschleunigen“ durch eine Genauigkeitsangabe „maximal 5km/h Abweichung“ ergänzt werden. Verstöße fallen durch diese Zusatzfunktionalität sofort auf und nicht erst bei der Testauswertung.

Sowohl die Präzision der Durchführung von Fahrmanövern als auch die Minimierung der Gefährdung von Menschen adressieren Luther und Schaal in [67]. In den von ihnen beschriebenen Versuchen werden die Fahrzeuge von Robotern gesteuert. Es werden somit keine Menschen direkt gefährdet und ein gewisser Grad an Automatisierung kann erreicht werden.

Trotz der geschilderten Maßnahmen zur Verbesserung von Tests im Fahrzeug bleiben diese ein komplexes und zeitintensives Unterfangen.

3.2.2. Datenaufzeichnung bei interagierenden Fahrzeugen

Die Entwicklung moderner Fahrassistenzsysteme, wie eines Abstandsregeltempomaten oder von Unfall vermeidenden Fahrerassistenzfunktionen erfordert die Integration von umgebungserfassender Sensorik in das Fahrzeug. Die zusätzliche Sensorik verschiebt dabei die Systemgrenze über das Fahrzeug hinaus. Nach Ammon [2] wird durch Nutzung der umgebungserfassenden Sensorik die Systemgrenze deutlich über das Fahrzeug hinaus auf die Fahrzeugumgebung erweitert. Das Systemfahrzeug und die Objektfahrzeuge interagieren dabei mit Hilfe der umgebungserfassenden Sensorik. Interagieren bedeutet hierbei „aufeinander bezogen handeln“ [135]. Durch die umgebungserfassende Sensorik können Fahrerassistenzsysteme ohne direktes Zutun des Fahrers auf umgebende Objektfahrzeuge bezogen handeln.

Die Erweiterung der Systemgrenze hat zur Konsequenz, dass das System innerhalb der neu gesetzten Grenzen getestet wird. Die resultierenden Tests beziehen sich somit auf das Systemfahrzeug und dessen Konstellation zu umgebenden Objekten, mit denen es interagiert. Für die Auswertung der Tests müssen somit sowohl Informationen über das Testobjekt selbst als auch über dessen aktuelle Situation in seiner Umgebung vorhanden sein. In [124] wird ein Ansatz vorgestellt, diese Informationen durch eine vom Fahrerassistenzsystem unabhängige Datenaufzeichnung in den Objektfahrzeugen zu erhalten. Für den Test im Fahrzeug wird die Verwendung von geeigneten Datenloggern[4] vorgeschlagen. Außerdem wird der Test von interagierenden Fahrzeugen in einem Entwicklungsmodell berücksichtigt. Innerhalb einer zusätzlichen Zerlegungsebene „Fahrzeug-Fahrzeug“ im auf die Fahrzeugindustrie angewandten V-Modell [128, 31] werden Objekte betrachtet, die sich innerhalb des Erfassungsbereichs der umgebungserfassenden Sensorik und damit innerhalb der neu gezogenen Systemgrenze befinden. Auf der Entwicklungsseite können hier die Anforderungen an interagierende Fahrzeuge eingeordnet werden, wie z. B. Anforderungen für eine Abstandsregelfunktion oder eine Kollisionsvermeidungsfunktion.

Beim Test von Fahrerassistenzsystemen im Fahrzeug wird die Fahrerassistenz-

[4] Ein Datenlogger nimmt nach Lerch in [65] vor Ort Prozessdaten – bei eingebetteten Systemen im Fahrzeug sind dies z. B. CAN-Nachrichten – auf und speichert diese. Die gespeicherten Daten können danach über eine Schnittstelle an einen Rechner zur Auswertung übertragen werden.

funktion im Zusammenspiel mit der realen Sensorik untersucht. Um bei der Auswertung des Tests auch deren korrekte Funktionsweise zu prüfen, müssen die von den Sensoren ermittelten Informationen über die Fahrzeugumgebung durch eine unabhängigen Datenerfassung bestätigt werden. Eine Möglichkeit besteht darin, Fahrdynamikdaten von den CAN-Bussen der Objektfahrzeuge als unabhängige Datenquelle aufzuzeichnen. Da sich die Fahrdynamikzustände der beteiligten Fahrzeuge sowohl über die Zeit als auch über die Bewegung auf der Fahrbahn ändern, können die aufgezeichneten Daten zeitlich oder ortsabhängig korreliert werden. In dieser Arbeit soll eine Korrelation über die Zeit betrachtet werden.

Zum Beispiel soll die Sensorinformation „Relativgeschwindigkeit zum vorausfahrenden Fahrzeug“ überprüft werden. Mögliche Lösungsansätze sind die Übertragung von Geschwindigkeitsinformationen zwischen den Fahrzeugen oder eine Referenzmessung durch einen zusätzlichen Sensor. Einfacher scheint der Ansatz, dass die Daten der unabhängigen Fahrzeuge durch ein übergeordnetes System einen Bezug zueinander bekommen. Als Basis für den Vergleich von Ereignissen innerhalb der zwei Systeme bietet sich die gemeinsame Zeitbasis des übergeordneten Systems an.

Aktuelle Datenlogger [37] zeichnen die Nachrichten mit Bezug zur eigenen Systemzeit auf. Aufzeichnungen von verschiedenen Datenloggern in verschiedenen Fahrzeugen lassen sich daher über deren Auftrittszeitpunkt nur im Rahmen der Genauigkeit der Uhrensynchronität vergleichen.

Zunächst könnte man durch einen Uhrenvergleich zu Beginn der Aufzeichnung sicherstellen, dass die aufgezeichneten Daten überhaupt über die Zeit vergleichbar sind. Die lokale Zeit innerhalb eines Datenloggers hängt von der Genauigkeit des verwendeten Oszillators ab und ist mit einer individuellen Drift beaufschlagt [132].

Ein einfacher Uhrenvergleich wird über einen begrenzten Zeitraum die Synchronität zweier Datenlogger gewährleisten. Abhängig von den Oszillatorgenauigkeiten werden die Systeme dann auseinander driften. Die individuelle Drift eines Datenloggers lässt sich durch Abbildung der Datenlogger-Systemzeit auf die Zeit eines übergeordneten Systems darstellen. Abbildung 3.3 zeigt den Bezug der Datenlogger-Systemzeit t_{DL} zu einer solchen dem System übergeordneten Zeit $t_{absolut}$.

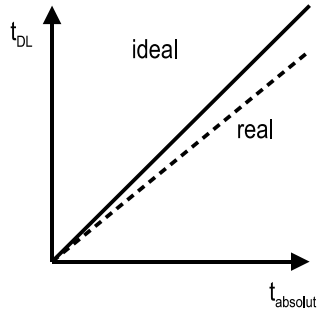


Abbildung 3.3.: Systemzeit in einem Datenlogger

In Abbildung 3.3 ist als einfaches Beispiel eine lineare Drift der Zeit innerhalb eines Datenloggers bezogen auf eine weltweit gültige Referenzzeit $t_{absolut}$ dargestellt. Die Referenzzeit kann beispielsweise von einer Atomuhr der physikalisch-technischen Bundesanstalt[5] bezogen werden. Die Kurve ideal zeigt den Fall, dass t_{DL} der $t_{absolut}$ entspricht. Die Datenloggeruhr läuft also synchron zur Absolutzeit. Die Kurve real zeigt eine lineare Drift der Datenloggeruhr relativ zur Absolutzeit. Die Uhr im Datenlogger läuft zu langsam.

Die lineare Drift hat ihre Ursache in der fertigungsbedingten Toleranz der verbauten Uhrquarze [132], daher ist sie in jedem System anders. In der Praxis sind dieser linearen Drift noch andere Ungenauigkeiten wie z. B. ein temperaturabhängiger Verlauf überlagert. Eine detaillierte Betrachtung des Driftverhaltens von Quarzen soll nicht der Fokus dieser Arbeit sein. In der Literatur wird beschrieben, wie sich die Präzision und Temperaturabhängigkeit von Quarzen fertigungstechnisch beeinflussen lässt. Marrison beschreibt in [69] verschiedene Varianten der Quarzfertigung. Hochpräzise Quarze benötigen einen komplexen Fertigungsprozess und sind daher sehr teuer. Barz und Drews [6] beschreiben Datenlogger auf Basis hochpräziser Quarze. Doch auch hier können Aufzeichnungen von unabhängigen Datenloggern in verschiedenen Fahrzeugen nicht in ausreichender Genauigkeit in deren Zeitverhalten verglichen werden. Als Hilfsmittel schlagen die Autoren eine *Synchronisation* der Datenlogger vor.

Es existieren verschiedene Möglichkeiten, unabhängige Uhren zu synchronisieren. Die Uhren der Systeme können über ein Netzwerk verbunden und dann wie in [67] vorgeschlagen über ein geeignetes Protokoll wie z. B. IEEE1588 [47] syn-

[5] Physikalisch-Technische Bundesanstalt, http://www.ptb.de.

chronisiert werden. Für fahrende Systeme muss die Vernetzung drahtlos erfolgen. Luther gibt für dieses Verfahren in [67] eine erreichte Synchronität von unter fünf Millisekunden an. Es erfolgt dann zwar eine Synchronisation der fahrenden Systeme untereinander, aber keine Synchronisation auf ein globales System. Eine Synchronisation auf eine global gültige Zeit kann beispielsweise durch den Einsatz des deutschlandweit verfügbaren DCF77 [7] Signals oder wie Barz und Drews in [6] vorschlagen durch die weltweit gültige und hochpräzise Zeit des *Global Positioning System (GPS)*[6] [143] erfolgen.

Die amerikanische National Highway Traffic Safety Administration (NHTSA) fordert in ihrem Bestätigungstest von Kollisionswarnsystemen [82, 29], dass über ein GPS-System die Positionsdaten der beteiligten Fahrzeuge erfasst werden. Weiterhin soll der Zeitpunkt der im Systemfahrzeug provozierten Warnung möglichst direkt auf dem CAN-Bus aufgezeichnet werden. Um eine Aussage über die Ausgabe der Warnung in Bezug zur Positionen der Fahrzeuge machen zu können, müssen beide Messungen beispielsweise über die GPS-Zeit zusammengeführt werden.

Für einen Datenlogger, der zur Aufzeichnung von Daten für die Analyse von interagierenden Fahrzeugen und somit von Fahrerassistenzsystemen basierend auf umgebungserfassender Sensorik verwendet werden soll, ergibt sich folgende Anforderung:

Anforderung 5. *Aufzeichnung von CAN-Nachrichten mit Bezug zu einer globalen Uhrzeit*
Die Aufzeichnung von CAN-Nachrichten in verschiedenen Fahrzeugen muss mit Bezug zu einer globalen Uhrzeit, wie zum Beispiel der GPS-Zeit, erfolgen. Eine Uhrgenauigkeit zwischen zwei Fahrzeugen von kleiner 5ms genügt nach [67] für die Messung von Fahrerassistenzsystemen bei einer Geschwindigkeit von bis zu 20m/s (72km/h). Da davon auszugehen ist, dass Fahrerassistenzsysteme in Zukunft auch in höheren Geschwindigkeitsbereichen untersucht werden müssen, soll die Uhrgenauigkeit zwischen zwei Fahrzeugen mindestens um Faktor fünf größer sein. Zusätzlich ist zu berücksichtigen, dass sich die Einzelfehler mehrerer Datenlogger addieren.[7] *Jede Uhr im System muss sich daher auf mindestens 500µs genau zur globalen Zeit synchronisieren.*

[6] Nach Brockhaus auch Navigation System with Time and Ranging (NAVSTAR/GPS) ganannt. Der GPS-Empfänger besteht unter anderem aus einer Präzisionsuhr, die auf Basis der Satellitensignale korrigiert wird [87].
[7] Berechnung der geforderten Genauigkeit zur globalen Zeit: $(5ms/5)/2 = 500\mu s$

Durch Anforderung 5 wird erreicht, dass Daten zeitlich synchron zueinander aufgezeichnet werden. Damit hängt das Ergebnis nicht von der Instanz des verwendeten Datenloggers ab. Darüber hinaus können Daten von verschiedenen Datenloggern zueinander in Bezug gesetzt werden.

3.2.3. Abspielen von im Fahrzeug aufgezeichneten Daten

Ein wichtiger Aspekt bei der Entwicklung von Fahrerassistenzfunktionen ist das Aufzeichnen und spätere Abspielen von im Fahrzeug aufgezeichneten Daten an einem Testsystem [3, 17]. Da es sich bei den zu testenden Systemen um Echtzeitsysteme handelt, muss beim Abspielen nicht nur der Dateninhalt identisch sein, sondern auch der Zeitpunkt, an dem ein Datum das Testobjekt erreicht, sollte eine vom Testobjekt abhängige Toleranz nicht überschreiten. Es ist somit zu differenzieren, wie kritisch das Testobjekt auf zeitliche Unterschiede bei seinen Eingangssignalen reagiert. Es gibt Systeme im Fahrzeug, die sehr tolerant sind, was das zeitliche Auftreten von Signalen betrifft. Ausschlaggebend ist hier lediglich der richtige Signalinhalt. In Abbildung 3.4 sind zwei Signalverläufe dargestellt, die für ein solches System identisch sind.

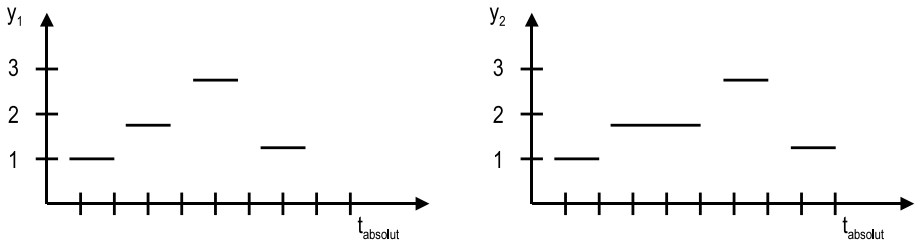


Abbildung 3.4.: Mögliche Signalverläufe beim Abspielen von aufgezeichneten Daten

Es gibt allerdings auch Systeme, deren Verhalten vom genauen zeitlichen Verlauf der Signale abhängt. Bei diesen führen die in Abbildung 3.4 gezeigten Verläufe der Eingangssignale zu unterschiedlichen Reaktionen. Wird beispielsweise derselbe Datensatz an einem Testsystem mehrmals hintereinander abgespielt, können diese kleinen Timing-Unterschiede zu einer anderen Reaktion des Testobjekts führen. Daher wird bei derzeitigen Testsystemen vor allem Wert darauf gelegt, dass die vom Testsystem generierten Nachrichten zeitlich möglichst präzise an das Testobjekt versandt werden. Es wird aber nicht berücksichtigt, dass bereits

beim Aufzeichnen der Signale eine zeitliche Ungenauigkeit durch den Datenlogger entstehen kann. In Abbildung 3.5 wird dargestellt, wie sich der zeitliche Fehler vom Aufzeichnen mit einem Datenlogger bis zum Abspielen mit einem Testsystem fortpflanzen kann. Das Testobjekt ist hier ein Steuergerät des Fahrzeugs.

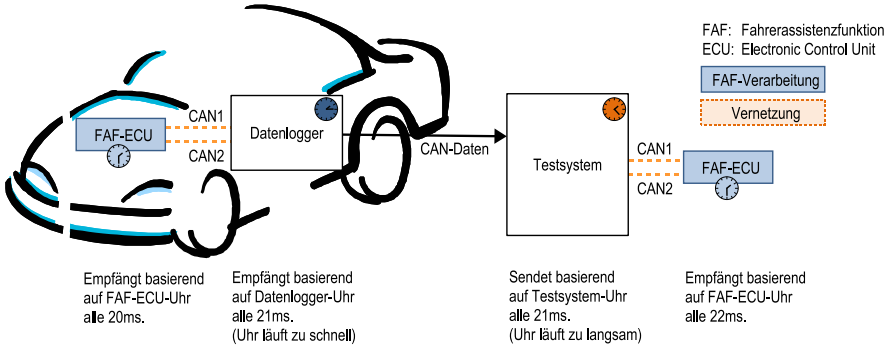


Abbildung 3.5.: Fehlerfortpflanzung von der Aufzeichnung zum Abspielen

Innerhalb des Fahrzeuges empfängt das Steuergerät relevante Signale beispielsweise im Abstand von 20ms. Der Datenlogger zeichnet diese parallel zum Steuergerät auf, allerdings basieren die Zeitinformationen auf dessen eigener Uhr. Nach dieser Uhr erscheinen die Signale alle 21ms. Werden die gesammelten Daten am Testsystem abgespielt, so versucht dieses nach seiner Uhr, die Daten hochgenau alle 21ms an das Steuergerät weiter zu geben. Allerdings basiert die Genauigkeit des Testsystems auf dessen lokalen Uhr, die wiederum eine Ungenauigkeit zur Absolutzeit aufweist. Im Gesamten entsteht ein Fehler, der dazu führt, dass die Signale aus der Sicht des Steuergeräts alle 22ms das Steuergerät erreichen. Wenn diese zeitlichen Ungenauigkeiten den Algorithmus im Steuergerät tangieren, dann reagiert dasselbe Steuergerät im Fahrzeug anders als am Testsystem.

Die zeitlichen Ungenauigkeiten der involvierten Systeme können sich teilweise kompensieren. In einer Worst-Case-Betrachtung jedoch addieren sich die Fehler. In einem einfachen Beispiel in Abbildung 3.6 sind verschiedene Fälle von Uhrgenauigkeiten relativ zur absoluten Zeit dargestellt. Die linke Grafik zeigt schematisch die Genauigkeit der Datenloggeruhr. Diese läuft zu langsam, daher vergeht die t_{DL} langsamer als die t_{abs}. Die Kurve *real* liegt daher unter der Kurve *ideal*. Das mittlere Diagramm verdeutlicht die Genauigkeit der Uhr im Testsystem. Auch diese Uhr läuft zu langsam. Werden an diesem Testsystem die vom

Datenlogger aufgezeichneten Daten abgespielt, so kann dies höchstens mit der im dritten Graphen durch die Kurve *real* gezeigten resultierenden Genauigkeit geschehen, da sich die Ungenauigkeiten addieren können.

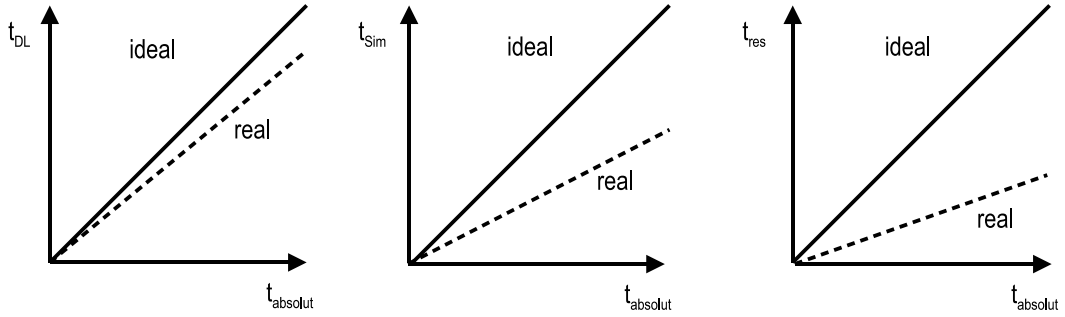


Abbildung 3.6.: Resultierende Systemzeit an einem Testsystem

Das auf die globale Zeit bezogene Zeitverhalten, das das Testsystem dem angeschlossenen Testobjekt bieten kann, ergibt sich aus der Überlagerung der Drifts der verschiedenen involvierten Systeme. Untersuchungen während der Arbeit haben ergeben, dass bei manchen Fahrerassistenzsystemen mit Bezug zur Systemzeit mit diesem Testsystem keine zeitlich exakt reproduzierbaren Ergebnisse mehr erzielt werden können.

Ein geeignetes Testsystem muss daher den Bezug zu einer globalen Zeit herstellen können, so dass Unterschiede, die durch die Umgebungstemperatur oder durch die Verwendung einer anderen Testsystem-Hardware auftreten, kompensiert werden.

Anforderung 6. *Abspielen von CAN-Nachrichten mit Bezug zu einer globalen Uhrzeit Das Abspielen von in einem Fahrzeug aufgezeichneten CAN-Nachrichten an einem Testsystem muss mit Bezug zu einer globalen Zeitbasis wie zum Beispiel der GPS-Zeit erfolgen. Dabei gilt analog zum Aufzeichnen in Anforderung 5, dass sich die lokale Uhr mit einer Genauigkeit von 500µs zur GPS-Zeit synchronisieren muss.*

3.2.4. Testen von Fahrerassistenzfunktionen an Testsystemen

Auf Grund der zunehmenden Komplexität der Tests im Fahrzeug und den damit verbundenen Kosten und Gefahren für die Testfahrer erfolgt der Test der durch

Software realisierten Funktionalität oft an einem Testsystem. Des Weiteren wird in der Literatur empfohlen, möglichst frühzeitig im Entwicklungsprozess zu testen [103, 78, 14]. Es ist daher sinnvoll, vor dem Test im Fahrzeug, Steuergeräte an einem Testsystem zu untersuchen. Abbildung 3.7 zeigt den prinzipiellen Aufbau eines möglichen Testsystems für ein Steuergerät mit Fahrerassistenzfunktionen. Je nach Testziel können einzelne Softwaremodule, die gesamte Funktionssoftware, das gesamte Steuergerät oder sogar das Steuergerät zusammen mit Sensorik und Aktorik mit dem Testsystem verbunden und getestet werden. Generell muss aus Sicht des Testobjekts der Kontext des umgebenden Systems simuliert werden. In Abbildung 3.7 besteht der Kontext beispielhaft aus den simulierten Modulen Fahrzeugumgebung, Fahrer, Fahrzeug, Sensorik und Aktorik. Wenn es sich beim Testobjekt um ein Steuergerät handelt, das über seine Schnittstellen an das Testsystem angeschlossen wird, wird von einem *Hardware-in-the-Loop (HiL)* Testsystem gesprochen [64, 75, 99].

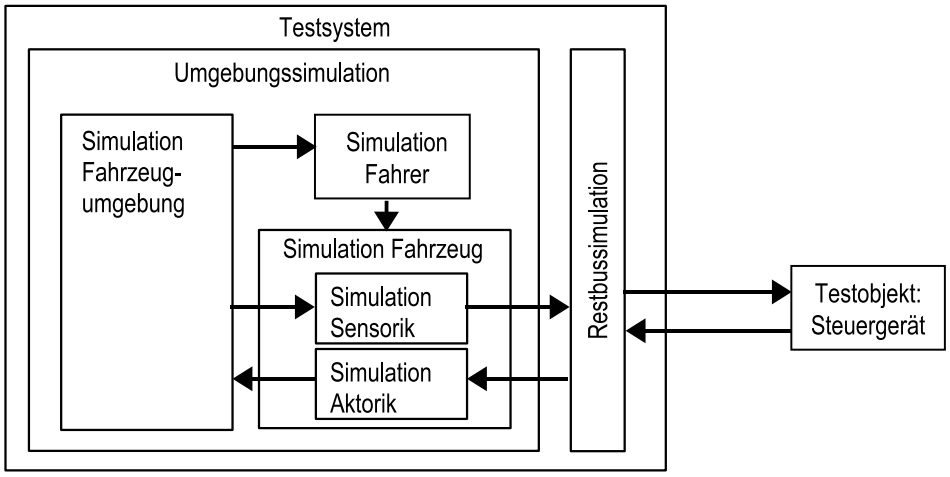


Abbildung 3.7.: Testsystem für eine Steuergerät mit Fahrerassistenzfunktionen

Entscheidend ist, dass alle aus Sicht des Testfalls und des Testobjekts relevanten Teile des Fahrzeugs und dessen Umgebung, die nicht real vorhanden sind, durch eine geeignete Simulation ersetzt werden. Dies kann durch eine Simulation oder durch die Wiedergabe vorher aufgezeichneter Daten erfolgen. Zusätzlich müssen die Zeitanforderungen der Schnittstelle zum Testobjekt eingehalten werden [99]. Eine passende Simulation für einzelne Teile, wie z. B. Aktorik, Sensorik, Fahrer und Fahrzeugumgebung zu erstellen, ist nach [3] die größte Herausforderung beim Bau eines Testsystems.

Die Anbindung des Testobjekts an das Testsystem erfolgt beim Harware-in-the-Loop Test über die Schnittstellen, die das Steuergerät im realen Fahrzeug zu den umgebenden realen Komponenten hat. Da Fahrerassistenzsteuergeräte zum Ziel haben, möglichst viele Funktionen zu bündeln [62], sind sie meist über Bussysteme mit ihrer Umgebung vernetzt. Im Rahmen dieser Arbeit handelt es sich hierbei um CAN-Busse. Daher wird in Anforderung 7 eine CAN-Verbindung zwischen Testsystem und Testobjekt gefordert.

Anforderung 7. *Anbindung des Testobjekts über CAN*
Das Testsystem kommuniziert über vier CAN-Busse mit dem Testobjekt.

Bei einem Hardware-in-the-Loop Testsystem für Fahrerassistenzsteuergeräte können in der Simulationssoftware zwei große Teilkomponenten unterschieden werden. Zum einen sorgt die Umgebungssimulation mit den darin enthaltenen Modellen für die Generierung eines semantisch korrekten Eingangsdatensatzes für das Testobjekt. Zum anderen ist es Aufgabe der *Restbussimulation*, die Informationen der Umgebungssimulation in einer für das Steuergerät akzeptablen Form zur Verfügung zu stellen und umgekehrt, Informationen vom Steuergerät in eine für die Umgebungssimulation verarbeitbare Form zu bringen [93, 134]. Die Restbussimulation simuliert somit das Kommunikationsverhalten nicht vorhandener Teilsysteme [68, 32]. Als Eingangsgröße erhält die Restbussimulation beispielsweise die Informationen aus der Umgebungssimulation. Letztere soll die reale Welt möglichst präzise widerspiegeln. Die gelieferten Eingangsgrößen werden oft auch *physikalische Werte* genannt.

Im Falle einer Anbindung des Testobjekts über einen CAN-Bus überführt die Restbussimulation die physikalischen Werte, die beispielsweise als hochauflösende 64-Bit Float-Werte [49] zur Verfügung stehen, in die in der Kommunikationsspezifikation festgelegte *Bit-Repräsentation*. Die Bit-Repräsentation deckt nur den tatsächlich benötigten Wertebereich und die geforderte Schrittweite der zu übertragenden Größe ab. Für die Umrechnung von hochauflösenden physikalischen Größen in eine verkürzte Bit-Repräsentation wird nach Zimmermann et al. [146] der Begriff *Skalierung* verwendet. Die Bit-Repräsentation wird in der Regel als Hexadezimalwert notiert. Dieser kann nach Gleichung 3.1 in den physikalischen Wert überführt werden. Die Restbussimulation muss zusätzlich dafür sorgen, dass die Daten innerhalb der vom Testobjekt geforderten zeitlichen Anordnung übertragen werden.

$$\mathit{Physikalischer\ Wert} = \mathit{Hexadezimalwert} \cdot \mathit{Steigung} + \mathit{Offset}. \tag{3.1}$$

Im Falle von eingebetteten Systemen im Fahrzeug wird für die Übertragung zeitkritischer Daten oftmals eine zyklische Kommunikation über den CAN-Bus implementiert [93]. Die Restbussimulation muss in diesem Fall dafür sorgen, dass die geforderte Zykluszeit eingehalten wird und dass zu jedem Zyklus der richtige Datensatz aus der Umgebungssimulation verschickt wird. Es lässt sich also festhalten, dass es sowohl aus der Sicht der Fahrerassistenzfunktion ein definiertes Zeitverhalten gibt, als auch aus Sicht der Kommunikation einzelner Steuergeräte. Je wichtiger der zeitlich korrekte Datenfluss aus Sicht der Anwendungsfunktion ist, desto empfindlicher reagiert die Qualität der Anwendungsfunktion in der Regel auf Verstöße gegen das spezifizierte Zeitverhalten und desto genauer wird dieses in der Regel zur Laufzeit überwacht.

Soll die Fahrerassistenzfunktion sowohl auf ihr korrektes Sollverhalten hin als auch auf ein korrektes Verhalten außerhalb der Spezifikation hin getestet werden, bedeutet dies, dass es das Testsystem ermöglichen muss, das Testobjekt sowohl innerhalb des spezifizierten Bereichs als auch außerhalb zu stimulieren. Auf das geforderte Zeitverhalten übertragen heißt das, dass das Zeitverhalten des Testsystems bei Positivtests reproduzierbar innerhalb des spezifizierten Bereichs liegen muss. Bei Negativtests muss es hingegen in weiten Bereichen flexibel beeinflussbar sein (siehe Anforderungen 1 und 2).

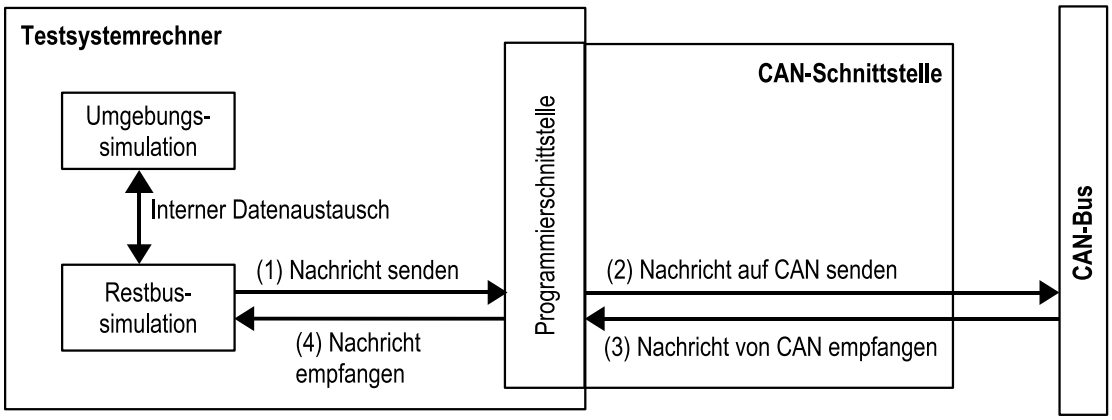


Abbildung 3.8.: Prinzipieller Aufbau eines Testsystems

In Abbildung 3.8 ist das Zusammenwirken der Komponenten eines typischen Testsystems dargestellt. Die Restbussimulation kommuniziert über eine Datenschnittstelle mit der Umgebungssimulation. Über den Mechanismus Nachricht

senden (1) überträgt die Restbussimulation die Daten an die zur CAN-Hardware gehörige Programmierschnittstelle[8] und beauftragt diese, die Daten auf den CAN-Bus zu legen (2). Die Ausgangsdaten des Testobjekts werden von der CAN-Hardware am Bus mitgelesen (3) und dann über die Programmierschnittstelle der Restbussimulation zur Verfügung gestellt (4). Die Restbussimulation stellt die Informationen dann der Umgebungssimulation zur Verfügung, nachdem sie die physikalischen Werte nach 3.1 errechnet hat. Das Zeitverhalten der CAN-Nachrichten hängt bei dieser Architektur von der Restbussimulation ab und somit von der zeitlichen Präzision der Hardware-Plattform, auf der die Restbussimulation realisiert ist.

Es existieren verschiedene Ansätze, das Zeitverhalten der Restbussimulation an den Anforderungen des Testobjekts auszurichten. Zum einen kann das gesamte Testsystem auf Basis einer Echtzeitplattform aufgebaut werden. Zum anderen kann versucht werden, mit dem Zeitverhalten einer nicht echtzeitfähigen Rechnerplattform zurecht zu kommen, wenn das Testsystem möglichst günstig und arbeitsplatznah sein soll. Im Folgenden sollen beide Ansätze kurz diskutiert werden.

Bei den Testsystemen für Echtzeitplattformen existiert der Trend zum Einsatz von Standard PC-Technologie. Nach Wittler [140] rücken auch hier Effizienz- und Kostengesichtspunkte verstärkt ins Blickfeld. PC-basierte Software- und Hardware-Lösungen eignen sich besonders, um diese Herausforderungen anzunehmen. Das von Wittler vorgestellte Testsystem setzt die Benutzerschnittstelle auf PC-Basis um. Vorgänge, die in Echtzeit auszuführen sind, werden vom Bedien-PC auf einen Echtzeit-PC geladen. Auch der Echtzeit-PC basiert auf Standard PC-Hardware, als Betriebssystem wird hier ein Echtzeit-Linux verwendet. Der Vorteil des Konzeptes ist, dass durchweg auf preisgünstige PC-Hardware aufgesetzt wird. Allerdings ist es immer noch nötig, spezielle Rechner mit speziellem Betriebssystem und passender Software für das Testsystem bereit zu stellen. In der Regel sind diese Systeme weder arbeitsplatznah noch preisgünstig realisierbar.

In der Praxis hat sich herausgestellt, dass Steuergeräte, vor allem, wenn sie lediglich über CAN-Schnittstellen mit der Umgebung kommunizieren, auch mit

[8] Eine Programmierschnittstelle (engl.: Application Programming Hardware, API) verbindet Anwendungssoftware – in diesem Fall die Restbussimulation – mit der zur Hardware gehörigen Betriebssoftware – hier die Treiber der CAN-Hardware [77].

Hilfe eines Arbeitsplatzrechners[9] mit CAN-Schnittstelle im In-the-Loop Betrieb getestet werden können. In Abbildung 3.9 ist ein solcher Aufbau dargestellt.

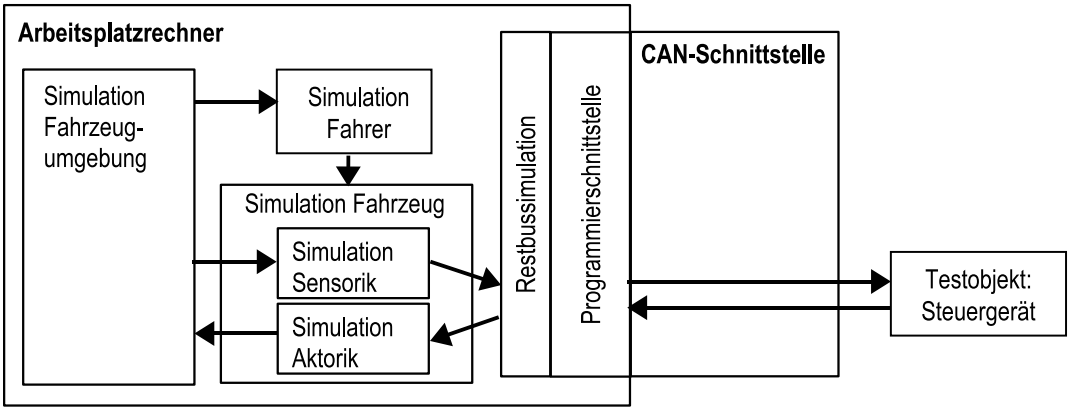


Abbildung 3.9.: Testsystem auf Basis eines Arbeitsplatzrechners

Es hat sich gezeigt, dass auch von einem Arbeitsplatzrechner Signale innerhalb der vom Steuergerät geforderten Echtzeitanforderungen über den CAN-Bus übertragen werden können. Als Beispiel erwartet das Steuergerät bestimmte Nachricht auf dem CAN-Bus mit einer Zykluszeit von 20ms. Es ist möglich, dass diese Zykluszeit von einem Arbeitsplatzrechner so eingehalten werden kann, dass das Steuergerät vorhandene Abweichungen toleriert. Das heißt die Vorgaben, die das Steuergerät an seine Umgebung stellt, werden vom Arbeitsplatzrechner eingehalten. Aus Sicht des Steuergerätes ist eine Kommunikation in Echtzeit gegeben.

Die Erfahrung zeigt, dass die Möglichkeit des Testens von Steuergeräten mit einem Arbeitsplatzrechner für die Funktionsentwickler und den Testsystembetreiber sehr attraktiv ist. Für den Testsystembetreiber ist es möglich, zügig auf Änderungswünsche zu reagieren und auch mehrere Testsysteme wirtschaftlich zu

[9] In dieser Arbeit ist mit dem Begriff Arbeitsplatzrechner ein Computer gemeint, wie er in großen Firmen standardmäßig von einer zentralen Stelle installiert, ausgeliefert und gewartet wird. In der Regel besteht ein solcher Rechner aus einer PC-Hardware der mittleren bis gehobenen Leistungsklasse, einem nicht echtzeitfähigen Betriebssystem (in der Regel Microsoft Windows [76]), einer Verbindung zum Hausnetz und Standardanwendungen wie Mailprogramm, Textverarbeitung, Virenscanner und Entwicklungswerkzeuge. In der Regel verfügen diese Geräte über genügend Rechenleistung und Hauptspeicher für komplexe Simulationen, so dass Software-in-the-Loop-Tests durchgeführt werden können. Die Installation einer echtzeitfähigen Betriebssystemlösung ist meist nur in Ausnahmefällen und zu deutlich höheren Kosten möglich.

verantworten. Für den Funktionsentwickler ergibt sich die Chance, direkt vor Ort am Arbeitsplatz ein solches preisgünstiges Testsystem zu haben, so dass er häufig und ohne Reservierungsaufwand am Steuergerät vorgenommene Änderungen testen kann. Insbesondere in Kombination mit einer an den Bedürfnissen des Funktionsentwicklers angepassten Sprache zur Formulierung von Testfällen für funktionsorientierte Tests, kann eine solches Testsystem den Funktionsentwickler effizient unterstützen. Eine für Fahrerassistenzsysteme sehr gut geeignete Sprache wird in [97, 18] vorgestellt.

Ein Nachteil des in Abbildung 3.9 dargestellten Konzeptes ist, dass nicht sicher erkannt werden kann, wenn durch den Arbeitsplatzrechner ein Verletzung der Echtzeitanforderungen vorliegt. Hieraus resultiert Anforderung 8. Das Zeitverhalten des Testsystems ist zwar ausreichend, das Testobjekt zu bedienen, aber eine beliebige Beeinflussung des Zeitverhaltens wie in Anforderung 2 gefordert, ist auf einem Arbeitsplatzrechner ohne die Unterstützung eines Echtzeitbetriebssystems nicht möglich. Außerdem scheint es, dass mit der Zykluszeit von $20ms \pm 7ms$ die Grenze dieser Architektur erreicht ist. Für zukünftige Generationen von Steuergeräten soll eine Zykluszeit von $10ms \pm 1ms$ erreicht werden, ohne die kommerziellen Vorzüge zu verlieren.

Anforderung 8. *Erkennen einer Verletzung von Echtzeitanforderungen*
Das Testsystem muss Verletzungen des auf den CAN-Bussen geforderten Zeitverhaltens erkennen können, wenn sie eine einstellbare Toleranz überschreiten.

3.3. Grenzen derzeitiger Testsysteme

Im Rahmen dieser Arbeit werden unter derzeitigen Testsystemen zum einen Testsysteme auf Basis von Realzeitbetriebssystemen (engl.: Real-Time Operating System (RTOS)) verstanden, wie sie beispielsweise von den Firmen Berner&Mattner [64], dSPACE [142], ETAS [141] und weiteren angeboten werden oder in der Forschung eingesetzt werden [101]. Diese sollen im weiteren Verlauf *RTOS-Testsysteme* genannt werden. Zum anderen existieren projektspezifische Lösungen auf Basis nicht echtzeifähiger Betriebssysteme auf Arbeitsplatzrechnern (engl.: Personal Computer (PC)) und einfacher CAN-Schnittstellen, wie z.B. Vector CANcaseXL [129]. Diese Testsysteme erweitern oftmals ein Testsystem für Software-Tests um

eine CAN-Schnittstelle, so dass ein Steuergerät als Testobjekt angeschlossen werden kann [101]. Sie werden im Rahmen dieser Arbeit *PC-Testsystem* genannt. In Kapitel 6 wird je ein Testsystem der Kategorie RTOS-Testsystem und PC-Testsystem hinsichtlich deren Zeitverhalten auf dem CAN-Bus mit der im Rahmen dieser Arbeit vorgeschlagenen Lösung verglichen. In Kapitel 3.1 und 3.2 wurde gezeigt, dass an ein Testsystem sowohl funktionale als auch nicht funktionale Anforderungen gestellt werden. Idealerweise sollte das Testsystem sowohl das Abspielen von Fahrzeugdaten als auch eine in-the-Loop Simulation unterstützen. Es wird argumentiert, dass derzeitige Testsysteme diesen Anforderungen nur teilweise gerecht werden oder sie nur mit hohem Aufwand umsetzbar sind. Im Folgenden sollen die Argumente zusammengefasst werden.

3.3.1. Flexible Beeinflussung des Zeitverhaltens und Wahren der verlangten Stabilität des Zeitverhaltens

In der Designphase eines Fahrerassistenzsystems wird das zeitliche Verhalten aus Kundensicht festgelegt. Oft bezieht es sich auf das Verhalten eines Fahrzeugs relativ zu anderen Fahrzeugen. Für die Analyse dieser geforderten Eigenschaften ist ein Vergleich von aufgezeichneten Daten aus allen beteiligten Fahrzeugen nötig. Dies ist dann möglich, wenn diese Daten mit derselben Zeitbasis aufgezeichnet werden.

Durch die architektonische Zerlegung der Fahrerassistenzsysteme auf verschiedene Steuergeräte werden auch an die entstandenen Teilsysteme Anforderungen bezüglich des Zeitverhaltens gestellt. Das Zeitverhalten der Umgebung des Teilsystems soll somit von der Umgebungssimulation möglichst realistisch abgebildet werden. Weiterhin muss das zeitliche Verhalten des CAN-Busses, mit dem das Testsystem und das Testobjekt verbunden sind, realitätsnah umgesetzt werden. Insbesondere beim Test auf Robustheit ist es nötig, das geforderte Zeitverhalten sowohl innerhalb der spezifizierten Grenzen als auch außerhalb derer nachzustellen. Dies kann bedeuten, dass einzelne Nachrichten verspätet zum Testobjekt gelangen oder ganz ausfallen. Bei heutigen Testsystemen ist es möglich, einzelne Nachrichten ausfallen zu lassen. Einzelne Nachrichten gezielt zeitlich zu verschieben, ist jedoch nicht vorgesehen. Zurzeit ist ein flexibel gestaltbares Zeitverhalten zum Beispiel zur Synchronisation des Testsystems auf das

Testobjekt nicht oder nur mit sehr hohen Aufwänden implementierbar.

Ein interessanter Ansatz, um den zeitlichen Anforderungen von Echtzeitsystemen aus der Sicht eines Testsystems gerecht zu werden, wird auch im Ptolemy Projekt der Universität Berkeley verfolgt [116]. Als Beitrag des Center for Hybrid and Embedded Software Systems wird vorgeschlagen, das gewünschte Zeitverhalten von der (Test-) Applikation mit Hilfe einer geeigneten Programmiersprache zu modellieren. Die Einhaltung des hierdurch geforderten Zeitverhaltens wird durch eine erweiterte CPU-Architektur sichergestellt [115]. Im Rahmen von dem Projekt Ptolemy wird ein Programmiermodell für verteilte Echtzeitsysteme vorgestellt (Programming Temporally Integrated Distributed Embedded Systems PTIDES) [144, 145]. Dieses sieht vor, auf Basis von diskreten Ereignissen, deren Ausführungszeitpunkt auf Applikationsebene formuliert wird, ein hart echtzeitfähiges Verhalten von verteilten Systemen zu erreichen. Der Ansatz basiert auf der Annahme, dass alle Knoten des verteilten Echtzeitsystems mit einem Netzwerk verbunden sind, für das eine Obergrenze für die Dauer der Übertragung von Paketen angegeben werden kann. Diese Annahme ist leider für den Datentransfer zwischen nicht echtzeitfähigen Rechnern nicht haltbar. Auch die Tatsache, dass die CPU-Architektur erweitert werden muss, schließt die Verwendung von Standard-Hardware und Software aus.

Die Arbeiten von Prof. Peleska an der Universität Bremen und der Firma verified [131, 106] zeigen, dass für den Test von echtzeitfähigen eingebetteten Systemen unter anderem folgende Punkte essentiell sind [84]:

- **Es muss möglich sein, bei der Beschreibung von Testfällen, deren Zeitverhalten zu modellieren.**
- **Die Testfälle müssen in Echtzeit ausgeführt werden.**
- **Sowohl die eingesetzte Hardware als auch die Software müssen ein präzises Zeitverhalten unterstützen.**

Insbesondere wird betont, dass für den Aufbau von Testsystemen Standard-Hardware und Software eingesetzt werden soll, da dies technische und finanzielle Vorteile bietet. Vorgeschlagen wird der Einsatz von PC-Hardware in Kom-

bination mit einem speziell erweiterten Linux-Betriebssystem. Aus Sicht der vorliegenden Arbeit geht dieser Schritt in die richtige Richtung. Das vorgestellte Konzept geht hierzu weiter und fordert unter anderem ein Betriebssystem ohne Erweiterungen.

Besonders intensiv wird der Aspekt diskutiert, wie das Zeitverhalten bei der Beschreibung von Tests berücksichtigt werden kann. Hierzu wird vorgeschlagen, einen modellbasierten Ansatz zu wählen, bei dem das Zeitverhalten mit Hilfe von Realzeitautomaten (timed automata) [43] beschrieben wird. Hierzu wird in [85] ein Testmodell auf Basis eines realen Beispiels aus der Automobilindustrie vorgestellt. Dieses Testmodell kann auf dem oben beschriebenen Testsystem ausgeführt werden. Da der Fokus dieser Arbeit auf dem Testsystem liegt und nicht auf der Beschreibung von Testfällen, ergänzen sich die Arbeiten und können in nachfolgenden Arbeiten verknüpft werden.

Die Universität Wien beschäftigt sich zusammen mit der Firma TTTech [113] in [100] mit dem Aufbau von Testsystemen für den Test von mehreren Steuergeräten im Verbund. Im Fokus der Arbeit steht die Synchronisation einzelner Rechenknoten des Testsystems untereinander und die Synchronisation dieses verteilten Testsystems mit dem Testobjekt. Es wird vorgeschlagen ein zeitgesteuertes Protokoll wie zum Beispiel das Time Triggered Protocol (TTP) [114] oder FlexRay [28] für die Synchronisationsaufgaben zu verwenden. Eine Besonderheit des in dieser Arbeit vorgestellten Ansatzes ist es, dass die Synchronität von Testsystem und Testobjekt nicht durch das dazwischenliegende Bussystem eingeschränkt wird.

Dass für Testsysteme eine Synchronisation wünschenswert ist, insbesondere wenn bei wiederholten Tests die selben Ergebnisse erreicht werden sollen, ist in [101] dargestellt. Schmidt wiederholt hier den funktionalen Test von Fahrerassistenzfunktionen. Auf Grund der dort fehlenden zeitlichen Synchronität und des nichtdeterministischen Verhaltens des CAN-Busses werden hier bei wiederholter Durchführung des selben Tests jedoch leider unterschiedliche Ergebnisse erzielt.

Werden Testfälle wiederholt durchgeführt, soll der Testfall bei jeder Iteration das spezifizierte Zeitverhalten innerhalb geringer Toleranzen aufweisen. Die Zeitbasis des Testsystems muss daher so stabil sein, dass die Testergebnisse reproduzierbar sind.

Der in dieser Arbeit vorgestellte Ansatz erlaubt es das Zeitverhalten des Testsystems dort zu definieren, wo dessen Datenverhalten festgelegt wird und löst aus Anwendersicht somit die Herausforderung einer Synchronisation mehrerer Systeme ähnlich wie PTIDES [145, 144]. Allerdings wird in der vorliegenden Arbeit besonders Wert darauf gelegt, ohne spezielle Rechnerhardware und ohne Realzeitbetriebssystem auszukommen. Zusätzlich zum Aufbau von Testsystemen für ein Steuergerät, wird auf Grund des ubiquitären Ansatzes der Test von Fahrerassistenzfunktionen in Fahrzeugen unterstützt.

3.3.2. Ubiquitäre Testsysteme

Da die Funktionalität von Fahrerassistenzsystemen in der Regel in Software umgesetzt wird, ist für das spätere Verhalten im Fahrzeug das Verhalten des Softwaremoduls auf der Zielhardware entscheidend. Die effiziente Absicherung des Zeitverhaltens ist daher auf der untersten Zerlegungsebene sinnvoll. Der Ansatz, ein Softwaremodul auf der Zielhardware zu testen, erfordert ein Testsystem für das gesamte Steuergerät. Um diese Vorgehensweise zeitlich effizient zu gestalten, ist ein flexibles, arbeitsplatznahes Testsystem für ein Steuergerät nötig. Da bei Erprobungsfahrten oftmals vor Ort Änderungen an der eingebetteten Software vorgenommen werden, bedeutet arbeitsplatznah, dass das Testsystem auf die Erprobungsfahrt mitgenommen werden soll, so wie der Entwicklerarbeitsplatz in Form eines Laptops mitgenommen werden kann. Dadurch können die Änderungen an der Software auch hier zunächst an einem Testsystem überprüft werden, bevor ein Test im Fahrzeug erfolgt. Dieses Vorgehen steigert die Effizienz und minimiert die Risiken für den Testfahrer. Ein ubiquitäres Testsystem ermöglicht den einfachen Transport im Falle einer Erprobungsfahrt.

Weiterhin muss das Testsystem so preisgünstig sein, dass möglichst viele Testsysteme parallel zur Verfügung gestellt werden können, so dass jeder Entwickler ein eigenes Testsystem hat. In diesem Fall entfallen Zeiten zum Umbau des Testsystems, wenn es zuvor von einem anderen Entwickler in einer anderen Version benötigt wurde und die Konfiguration des Testsystems ist dem Entwickler jederzeit bekannt. Idealerweise können die Testfälle für das Testsystem ohne Expertenwissen über das Testsystem formuliert werden. Insbesondere bedeutet dies, dass Manipulationen des Zeitverhaltens des CAN-Busses einfach formuliert und

umgesetzt werden können.

3.4. Zusammenfassung vom Stand der Technik

Die im Verlauf von Kapitel 3 aufgeführten Anforderungen an ein Testsystem für Fahrerassistenzsysteme sind in Tabelle 3.1 zusammengefasst. Zusätzlich wird deren Erfüllung durch die bestehenden Testsysteme RTOS-Testsystem und PC-Testsystem dargestellt.

Bei der Entwicklung von RTOS-Testsystemen steht die Echtzeitfähigkeit im Vordergrund (Anforderung 1), während beim PC-Testsystem das primäre Ziel eine preisgünstige und ubiquitäre Lösung ist, so dass viele Testsysteme parallel betrieben werden können und möglichst jeder Entwickler ein eigenes Testsystem hat, dessen Konfiguration für ihn optimiert ist (Anforderungen 3 und 4). Die Echtzeitfähigkeit kann hier jedoch nicht garantiert werden [101]. Bei der Durchführung von Tests muss daher immer berücksichtigt werden, dass der Test auf Grund eines Verstoßes gegen Echtzeitanforderungen fehlschlagen kann. Da keine zuverlässige Aussage über das Echtzeitverhalten getroffen werden kann, ist die Implementierung einer flexiblen Beeinflussung des Zeitverhaltens nicht sinnvoll (Anforderung 2). Prinzipiell ist es möglich, dies an einem RTOS-Testsystem durch eine angepasste Restbussimulation umzusetzen. Im Auslieferungszustand des Testsystems ist die Beeinflussung des Sendezeitpunkts einzelner CAN-Nachrichten jedoch nicht vorgesehen. Ein manueller Umbau der Restbussimulation wurde im Projektumfeld auf zehn Manntage geschätzt. In dieser Zeit kann das Testsystem nicht für die Durchführung von Tests eingesetzt werden.

In der Regel wird für das Aufzeichnen von CAN-Nachrichten im Fahrzeug eine andere CAN-Hardware eingesetzt als für Testsysteme. Daher unterstützen die bestehenden Testsysteme Anforderung 5, die sich auf die Anwendung im Fahrzeug bezieht, nicht. Auch aus kommerzieller Sicht ist es sinnvoll, für die Anwendungsfälle „Datenaufzeichnung im Fahrzeug“ und „Testsystem“ dieselbe CAN-Hardware zu verwenden. Einerseits können dadurch die Anschaffungskosten gesenkt werden, andererseits entfällt die Einarbeitung in unterschiedliche Systeme.

Bei den bestehenden Testsystemen ist die zeitliche Reproduzierbarkeit bei der

wiederholten Durchführung eines Testfalls nur im Rahmen deren Uhrgenauigkeit gewährleistet. Beide Testsysteme verwenden eine lokale Zeitbasis. Daher hängt die zeitliche Reproduzierbarkeit von Testfällen von der Stabilität der verwendeten Zeitbasis ab. Anforderung 6 wird daher von den bestehenden Testsystemen nicht erfüllt.

Da CAN eine Standardschnittstelle ist, wird sie in der Regel von allen Testsystemen unterstützt (Anforderung 7). Weder das RTOS-Testsystem noch das PC-Testsystem unterstützen im Auslieferungszustand das sichere Erkennen von Verletzungen des von der CAN-Kommunikation geforderten Zeitverhaltens (Anforderung 8).

In Kapitel 4 wird ein Ansatz vorgestellt, der ein echtzeitfähiges und ubiquitäres Testsystem nach den Anforderungen in Tabelle 3.1 ermöglicht. Es kombiniert die Stärken der bestehenden Testsysteme (Anforderungen 1, 3, 4, 7) und ergänzt die zusätzlich geforderten Eigenschaften (Anforderungen 2, 5, 6, 8).

Anforderung	RTOS-Testsystem	PC-Testsystem
1: Echtzeitfähigkeit	ja	nein
2: Flexible Beeinflussung des Zeitverhaltens	nein	nein
3: Preisgünstig	nein	ja
4: Ubiquitär	nein	ja
5: Aufzeichnung von Nachrichten mit Bezug zu einer globalen Uhrzeit	nein	nein
6: Abspielen von Nachrichten mit Bezug zu einer globalen Uhrzeit	nein	nein
7: Anbindung des Testobjekts über CAN	ja	ja
8: Erkennen einer Verletzung von Echtzeitanforderungen	nein	nein

Tabelle 3.1.: Nachteile der bestehenden Testsysteme in Bezug auf die gestellten Anforderungen

4. Konzept eines Zeitstempel-basierten Testsystems

Motiviert durch das Verwenden iterativer Vorgehensmodelle bei der Entwicklung eingebetteter Systeme und durch einen kontinuierlichen Kostendruck werden dynamische Tests zu einem möglichst frühen Zeitpunkt im Entwicklungsprozess – bei der Untersuchung des späteren Echtzeitverhaltens somit bevorzugt als Hardware-in-the-Loop Tests – durchgeführt. Durch die zunehmende Vernetzung mit verschiedenen Sensor- und Aktorsteuergeräten, wird es immer wichtiger, frühzeitig das Zeitverhalten des Testobjekts im Zusammenwirken mit dessen Umgebung zu untersuchen. Im folgenden Kapitel soll ein Konzept für ein preisgünstiges und flexibles Hardware-in-the-Loop Testsystem mit reproduzierbar präzisem Zeitverhalten vorgestellt werden.

Aus Kapitel 3 geht hervor, dass ein Testsystem zum entwicklungsbegleitenden Test von Fahrerassistenzsystemen folgende Punkte erfüllen muss:

1. Das Testsystem muss mit dem Testobjekt über **dessen Schnittstellen kommunizieren. Bei der Umsetzung im Rahmen dieser Arbeit handelt es sich hierbei um vier CAN-Busse** (Anforderung 7).

2. Das Testsystem muss mit dem Testobjekt in der vom Testobjekt geforderten Echtzeit kommunizieren (Anforderung 1).

3. Eine Verletzung von Echtzeitanforderungen außerhalb einer vorgegebenen Toleranz muss vom Testsystem erkannt werden (Anforderung 8).

4. Das Zeitverhalten muss reproduzierbar sein (Anforderungen 5 und 6).

5. Das Zeitverhalten jeder Nachricht, die über den CAN-Bus übertragen wird,

muss beeinflussbar sein (Anforderung 2).

6. Das Testsystem muss arbeitsplatznah verfügbar sein (Anforderung 4).

7. Das Testsystem muss preisgünstig sein (Anforderung 3).

Aus den Punkten 2. und 5. geht hervor, dass ein „zeitlich präzises Steuergerätetestsystem“ gefordert wird. Die Punkte 6 und 7 legen nahe, dass es „auf Basis einer nicht echtzeitfähigen Plattform“ aufgebaut wird, so dass es preisgünstig und somit in großer Stückzahl in eine Standard-Arbeitsplatzumgebung integriert werden kann. Beide Problemstellungen können mit Hilfe eines Zeitstempel-basierten Testsystems gelöst werden.

Kapitel 4.1 beschreibt zunächst die Methode eines Zeitstempel-basierten Testsystems **im Allgemeinen und dann für die in der prototypischen Umsetzung geforderten vier CAN-Schnittstellen.** Kapitel 4.2 veranschaulicht die Architektur der prototypischen Umsetzung. In Kapitel 4.3 wird beispielhaft erläutert, wie das Zeitverhalten des Testsystems aus einer Simulationssoftware heraus definiert und manipuliert werden kann. Kapitel 4.4 liefert Messungen des vom Prototypen erreichten Zeitverhaltens. Abschließend werden in Kapitel 4.5 die Kernpunkte des vorgestellten Konzepts zusammengefasst.

4.1. Methode des Zeitstempel-basierten Testsystems

Bei den bestehenden Testsystemen werden unterschiedliche Kriterien in den Vordergrund gestellt, deren Optimierung die nachrangigen Kriterien teilweise negativ beeinflusst. Beim RTOS-Testsystem steht die Echtzeitfähigkeit im Vordergrund. Dies führt unter anderem dazu, dass die gesamte für das Testsystem benötigte Software auf einem Echtzeitsystem ausgeführt wird. Die vorgestellten Implementierungen sehen weder eine flexible Beeinflussung des Zeitverhaltens noch eine Synchronisation zu GPS vor. Sie basieren auf teurer und komplexer Hard- und Software, die nicht für den Betrieb am Arbeitsplatz, auf Versuchsfahrten oder im Fahrzeug konzipiert ist.

Das PC-Testsystem hingegen ist für die zuletzt genannten Einsatzgebiete op-

timiert. Da hier besonderer Wert auf eine preisgünstige und kompakte Umsetzung ohne spezielle Betriebssysteme gelegt wird, kann mit den vorhandenen Bus-Schnittstellen nur eine sehr eingeschränkte Echtzeitfähigkeit realisiert werden. Bisher ist die Echtzeitfähigkeit ausreichend für funktionale Tests von Fahrerassistenzfunktionen. Bei fehlgeschlagenen Testfällen muss jedoch bei der Ursachenanalyse ein mögliches Fehlverhalten des Testsystems mit betrachtet werden.

Abstrahiert kann gesagt werden, dass ein Testsystem zum einen aus einer Einheit besteht, die das Daten- und Zeitverhalten gegenüber einem Kommunikationspartner – in diesem Fall dem Testobjekt, einem Echtzeitsystem – berechnet und zum anderen aus einer Kommunikationsschnittstelle, über die eine Kommunikation in Echtzeit stattfindet.

Bei bisherigen Ansätzen werden die beiden Belange „Bereitstellung von Daten“ und „In Echtzeit kommunizieren“ gemeinsam gelöst.

Bereits 1982 empfiehlt Dijkstra in [21] die Trennung von Belangen („Separation of concerns“) beim Entwurf von informationstechnischen Systemen. Auf die Problemstellung von kommunizierenden Echtzeitsystemen angewandt, sind die zwei unabhängige Belange zum einen die Bereitstellung von Daten, zum anderen eine echtzeitfähige Kommunikation zwischen den Kommunikationspartnern. Erstere fordert in der Regel eine hohe Rechenleistung und wird oftmals von Programmen erfüllt, die häufigen Softwareänderungen unterliegen, so dass das Datenverhalten an ein geändertes Verhalten des Kommunikationspartners angepasst werden kann.

Die Kommunikation hingegen soll reproduzierbar echtzeitfähig sein und deren Zeitverhalten in weiten Grenzen frei beeinflussbar sein. Abbildung 4.1 stellt die Trennung dieser Belange in einem Sequenzdiagramm dar. Anstelle die Verantwortung für das Daten- und das Zeitverhalten auf einem Rechner abzubilden, werden diese Belange zwischen dem nicht echtzeitfähigen Arbeitsplatzrechner und der Kommunikationsschnittstelle, die Hardware-nah implementiert ist, separiert. Der nicht echtzeitfähige Arbeitsplatzrechner berechnet den Dateninhalt (4) und den geforderten Zeitpunkt (5) für die Kommunikation mit dem Echtzeitsystem, während die Kommunikationsschnittstelle für eine präzise und reproduzierbare Einhaltung des geforderten Zeitverhaltens sorgt, indem sie schnellstmöglich die zu versendenden Daten erhält (7,8),

auf den festgelegten Übertragungszeitpunkt wartet (9) und dann die Nachricht zum Kommunikationspartner schickt (10).

Der vorgestellte Ansatz trennt die Belange „Rechenleistung“ und „Zeitverhalten“, so dass beide unabhängig voneinander optimiert werden können.

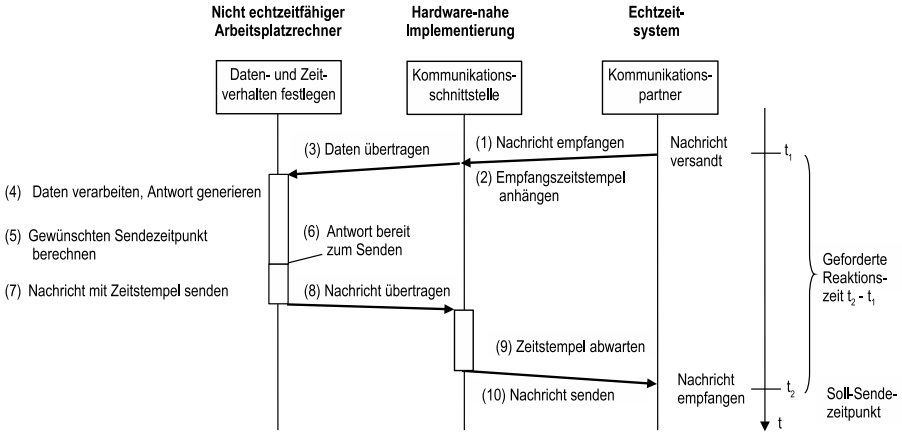


Abbildung 4.1.: Sequenzdiagramm eines Empfangs-/Sendezyklus mit Trennung von Verantwortlichkeiten

Auf Testsysteme im automobilen Umfeld angewandt bedeutet dies, dass zur Berechnung der Daten und des Zeitverhaltens die gängige Software aus Software-Si-mu-la-tions-um-ge-bung-en für Testsysteme eingesetzt werden können und diese auf einem Arbeitsplatzrechner ausgeführt werden. Als Kommunikationsschnittstelle dient zum Beispiel die weit verbreitete und für Echtzeitanwendungen im Fahrzeug häufig benutzte CAN-Schnittstelle. Aus Kostengründen [13] wird der CAN-Bus auch weiterhin bevorzugt eingesetzt, sofern dessen Bandbreite und Echtzeitfähigkeit ausreicht. Eine Möglichkeit zu schaffen, das Verhalten des CAN-Busses an seinen Grenzen zu testen wird als wichtiges Ergebnis dieser Arbeit angesehen. Die Arbeit kann hierdurch einen Beitrag leisten, in manchen Einsatzgebieten weiterhin mit dem einfachen und preisgünstigen CAN-Bus auszukommen.

Aber auch für elaboriertere (und damit teurere) Bussysteme wie FlexRay [28] die spezifikationsgemäß Echtzeitfähigkeit und eine garantierte Dienstgüte (Quality of Service) anbieten, sind die Ergebnisse der Arbeit anwendbar; Tests von Flottenfahrmanövern (siehe Kapitel 6.2) seien hier beispielhaft genannt.

Arbeitsplatzrechner verfügen in der Regel über eine hohe Rechenleistung und sind daher prinzipiell als Simulationsrechner geeignet. Um diese Rechner am Netzwerk zu betreiben, schreiben die Firmenrichtlinien in der Regel das Betriebssystem, Virenscanner und Firewall vor. Die standardisierte Bereitstellung macht einen solchen Rechner zu einer günstig verfügbaren Entwicklerplattform, die bis zum Software-Modultest die Entwicklungskette unterstützen kann. Sobald die entwickelte Software im Echtzeitkontext ausgeführt werden soll, muss sie auf der Zielhardware ausgeführt werden und idealer Weise über die bestehenden Modelle vom Software-Modultest stimuliert werden. Dies kann dadurch ermöglicht werden, dass die Zielhardware an den Arbeitsplatzrechner angeschlossen wird. Erfolgt diese Anbindung mittels einer Standard-Busschnittstelle, so ist der Arbeitsplatzrechner für das Zeitverhalten des Testsystems verantwortlich, wie es in Abbildung 3.8 dargestellt ist.

Der zeitliche Ablauf der Kommunikation bei dieser Lösung ist in Abbildung 4.2 dargestellt.

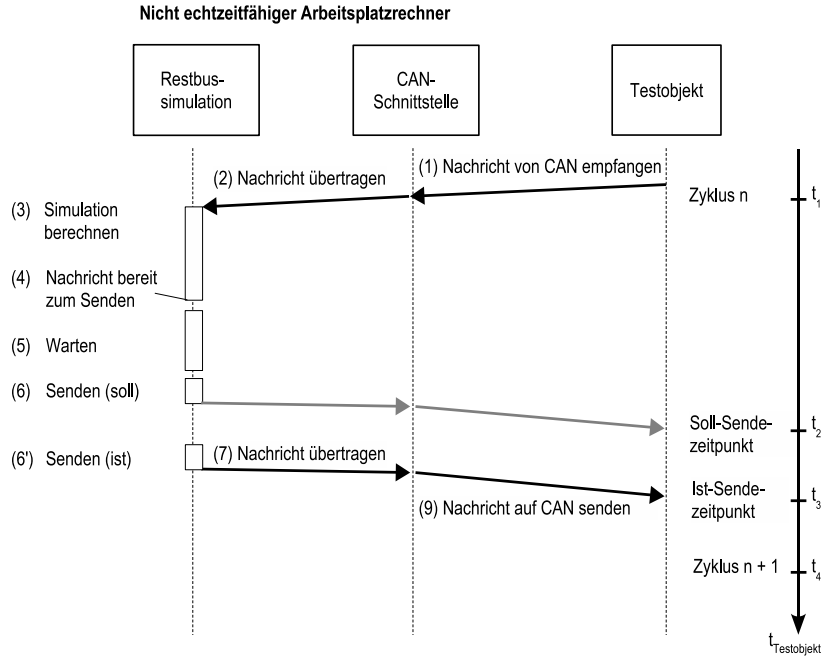


Abbildung 4.2.: Sequenzdiagramm eines Empfangs-/Sendezyklus mit Standard CAN-Schnittstelle

Eigene Versuche haben gezeigt, dass der Arbeitsplatzrechner nach Empfang einer CAN-Nachricht über die CAN-Schnittstelle (1) die Berechnung der Umgebungssimulation (2) in einem Bruchteil der Zeit durchführen kann, die durch die zyklische Kommunikation als Antwortzeit vorgegeben ist. Allerdings wird dies nicht durch das Betriebssystem garantiert. Kommt es zu unerwarteten Zeitverschiebungen, kann dies zu verfälschten Testergebnissen führen. Außerdem ist zu beobachten, dass die Berechnung eines Simulationsschrittes gewöhnlich so schnell erfolgt (3) und danach das zu versendende Ergebnis bereit steht (4), dass bis zum Versenden mehrere Millisekunden gewartet (5) werden muss. Dieses Warten führt auf einer nicht echtzeitfähigen Plattform zu deutlichen zeitlichen Unsicherheiten und es kann nicht garantiert werden, dass die Simulationsapplikation rechtzeitig vom Betriebssystem berücksichtigt wird (6). Der tatsächliche Zeitpunkt (6') des Aufrufs durch das Betriebssystem bestimmt, wann die Nachricht an die CAN-Schnittstelle übergeben wird (7) und somit wann die Nachricht auf dem Bus versendet wird (9). Die Abweichung von Soll-Sendezeitpunkt und Ist-Sendezeitpunkt hängt somit von der Aufweckgenauigkeit des Betriebssystems und vom Übertragungsjitter der CAN-Schnittstelle inklusive deren Anbindung an den Arbeitsplatzrechner ab.

Eine Lösungsmöglichkeit ist, das Verzögern einer Nachricht von deren Bereitstellung bis zu deren gewünschtem Sendezeitpunkt nicht auf dem Arbeitsplatzrechner, sondern auf der CAN-Schnittstelle durchzuführen. Dieser Ansatz wird in [119, 121, 120] **zum Patent angemeldet** und ist in Abbildung 4.3 bei (8) zu sehen.

Auf Seiten des Arbeitsplatzrechners wird das Warten auf den richtigen Sendezeitpunkt dadurch ersetzt, dass die Nachricht zusammen mit dem berechneten Soll-Sendezeitpunkt sofort an eine CAN-Schnittstelle mit der Fähigkeit, auf einen vorgegeben Zeitpunkt hin zu senden, übertragen wird (6). Diese CAN-Schnittstelle wird im Folgenden *Realzeitadapter* genannt. Die Möglichkeit des zeitgesteuerten Sendens sorgt dafür, dass die Nachricht exakt zum geforderten Zeitpunkt auf dem Bus übertragen wird (9).

Definition 2. *Realzeitadapter (engl.: Real-Time Adapter, RTA)*
Der Realzeitadapter ist eine Gerät zur Anbindung von CAN-Bussen an einen Computer. Das gewünschte Zeitverhalten auf den CAN-Bussen kann von einer Software auf dem Computer vorgegeben werden, während der Realzeitadapter für die Einhaltung der Zeitvorgaben sorgt. Sollten die Zeitvorgaben zuzüglich einer einstellbaren Toleranz nicht

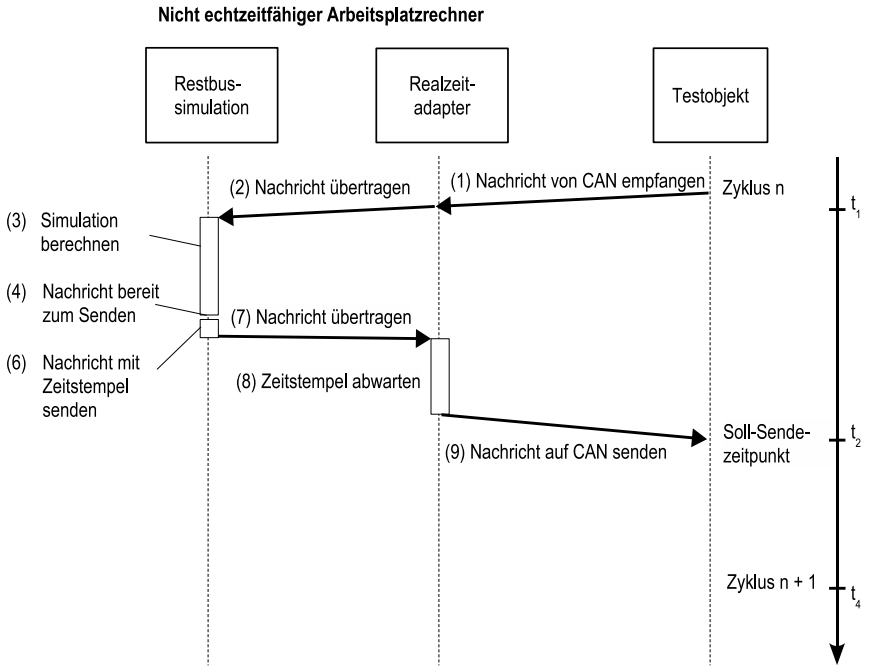


Abbildung 4.3.: Sequenzdiagramm eines Empfangs-/Sendezyklus mit dem Realzeitadapter

eingehalten werden, generiert der Realzeitadapter eine Fehlermeldung.

Die Zeitmessung auf dem Realzeitadapter kann auf Basis der weltweit eindeutigen globalen Zeit erfolgen, welche beispielsweise durch das Global Positioning System [143] bereitgestellt wird.

Der Ansatz des zeitgesteuerten Sendens birgt einen weiteren Vorteil. Es ist möglich, dass die Simulationssoftware durch das nicht echtzeitfähige Betriebssystem zu spät Zugriff auf den Prozessor erhält und somit eine Nachricht nicht mehr rechtzeitig an den Realzeitadapter übertragen werden kann. In diesem Fall liegt der Soll-Sendezeitpunkt in der Vergangenheit, was der Realzeitadapter zuverlässig erkennen und melden kann [120]. Durch diesen **im Rahmen der Arbeit entwickelten und zum Patent angemeldeten** Mechanismus ist zwar nicht garantiert, dass Soll-Sendezeitpunkte immer eingehalten werden, aber es ist sichergestellt, dass Zeitverletzungen sicher erkannt werden.

Zusammenfassend ist zu sagen, dass für das Zeitverhalten des gesamten Testsystems somit nicht mehr der Simulationsrechner sondern der Realzeitadapter verantwortlich ist. Wird dessen lokale Uhr auf eine weltweit eindeutige globale Zeit – wie in Kapitel 3.2.2 bereits für das Aufzeichnen von CAN-Nachrichten gefordert – synchronisiert, so ist das Zeitverhalten unabhängig von lokalen Bedingungen reproduzierbar und präzise. **Dieser der Arbeit zu Grunde liegende Ansatz ist Gegenstand der angemeldeten Patente [117, 118, 120]**

4.2. Architektur des Realzeitadapters (RTA)

Der Ansatz, das Zeitverhalten von der inhaltlichen Berechnung der Simulationsdaten zu entkoppeln, kann durch eine modifizierte CAN-Schnittstelle in Kombination mit einem nicht echtzeitfähigen Computer realisiert werden. In diesem Kapitel soll die Architektur einer solchen CAN-Schnittstelle, die nach Definition 2 Realzeitadapter genannt wird, beschrieben werden.

Der Realzeitadapter besteht aus einer Hardware, die mit einem Arbeitsplatzrechner verbunden wird und aus einer Programmierschnittstelle, die auf diesem Arbeitsplatzrechner zum Laufen kommt. **Für eine Beschreibung der Programmierschnittstelle des Prototypen sei auf Kapitel A verwiesen.**

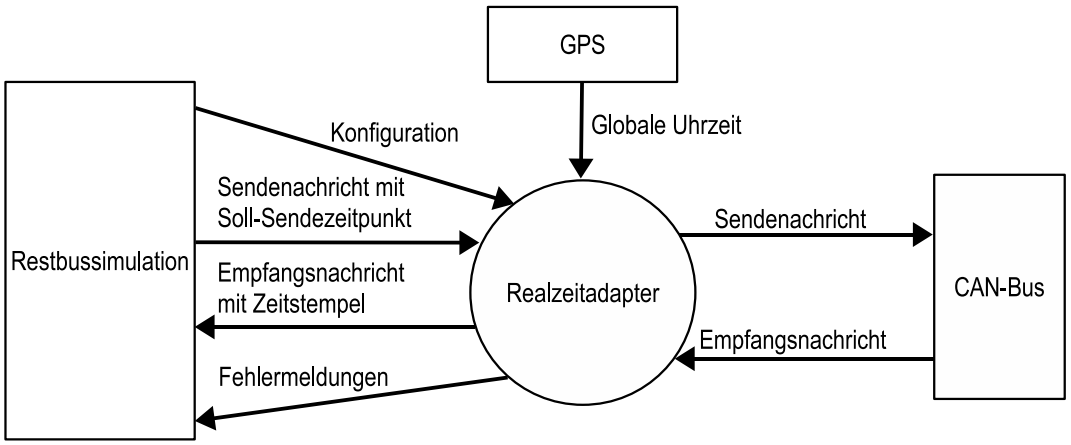


Abbildung 4.4.: Kontextdiagramm des Realzeitadapters

In Abbildung 4.4 ist das Kontextdiagramm des Realzeitadapters dargestellt. Das Kontextdiagramm zeigt, dass es möglich sein soll, über den CAN-Bus Nachrichten vom zu testenden System zu empfangen und im Gegenzug Nachrichten zu verschicken. Dabei soll das für die Buskommunikation geforderte Zeitverhalten eingehalten werden. Hierfür stehen die folgenden Anwendungsfunktionen zur Verfügung:

- Die über den CAN-Bus empfangenen Nachrichten können zusammen mit einem präzisen Eingangszeitstempel an der Programmierschnittstelle abgeholt werden.

- Über die Programmierschnittstelle können zu sendende Nachrichten mit einem Soll-Sendezeitpunkt an den Realzeitadapter übergeben werden, der diese dann zum gewünschten Zeitpunkt auf dem CAN-Bus verschickt.

- Über die Programmierschnittstelle können Nachrichten mit einem Soll-Sendezeitpunkt an den Realzeitadapter übergeben werden. Er stellt dieses dann zum gewünschten Zeitpunkt an der Programmierschnittstelle zur Verfügung.

- Vor dem Start des Systems kann über die Programmierschnittstelle festgelegt werden, wie genau das geforderte Zeitverhalten eingehalten werden soll. Wird der vorgegebene Wert überschritten, so wird dies über eine Fehlermeldung an der Programmierschnittstelle mitgeteilt.

- Über die Schnittstelle zum GPS kann die lokale Zeit des Realzeitadapters auf die globale Zeit synchronisiert werden.

Abbildung 4.5 zeigt, wie die im Kontextdiagramm dargestellten Schnittstellen durch die verschiedenen Module des Realzeitadapters umgesetzt werden. Aus Sicht einer Restbussimulation auf dem Arbeitsplatzrechner sind die drei Pfade Sendewarteschlange (engl.: Transmit Queue, TXQ), Fehlerwarteschlange (engl.: Error Queue, ErrQ) und Empfangswarteschlange (engl.: Receive Queue, RXQ) verfügbar. Die Warteschlangen werden auf der Eingangsseite befüllt und können auf der Verarbeitungsseite geleert werden. Die Warteschlangen entkoppeln dabei das Zeitverhalten der Restbussimulation vom Zeitverhalten auf dem CAN-Bus. Die Warteschlangen erstrecken sich von der Programmierschnittstelle des Realzeitadapters auf dem nicht echtzeitfähigen Computer bis zu der Sende- und

Empfangslogik auf der Realzeitadpater-Hardware.

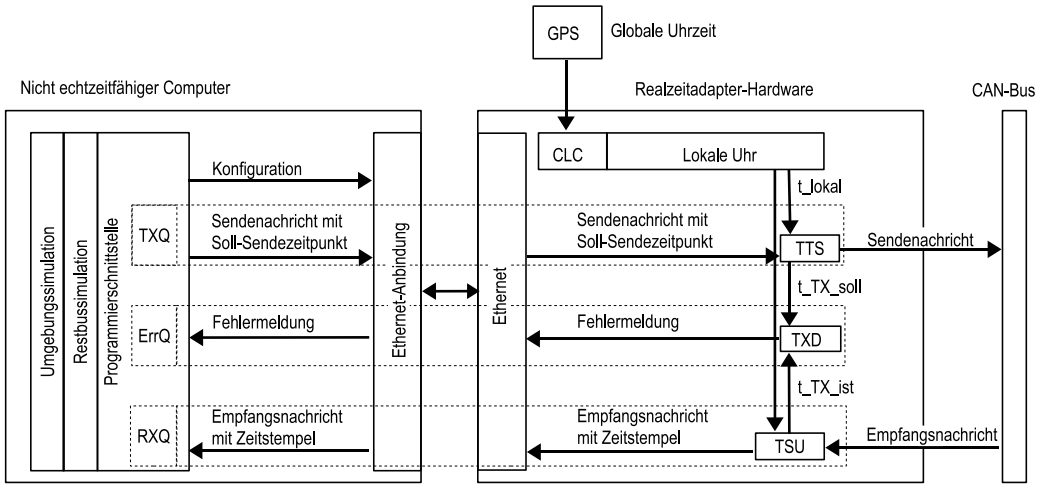


Abbildung 4.5.: Datenflussdiagramm des Realzeitadapters

Empfangsseitig wird die zeitliche Entkopplung dadurch realisiert, dass die Zeitstempeleinheit (TSU, engl.: Time Stamping Unit) zu jeder Nachricht den genauen Zeitpunkt des Nachrichtanfangs registriert und diesen zusammen mit dem Inhalt der Nachricht in der Empfangswarteschlange bereitstellt. Ab diesem Zeitpunkt kann jederzeit der Eintrittszeitpunkt der Nachricht in das System abgerufen werden, unabhängig davon, wann die Nachricht wieder aus der Empfangswarteschlange entnommen wird.

Sendeseitig kann die Restbussimulation für eine zu sendende Nachricht den Soll-Sendzeitpunkt auf Basis des gewünschten Zeitverhaltens auf dem CAN-Bus berechnen. Die Nachricht kann dann zu jedem Zeitpunkt vor dem gewünschten Sendezeitpunkt von der Restbussimulation an die Programmierschnittstelle übergeben werden. Die Programmierschnittstelle puffert die zu sendende Nachricht bis kurz vor dem Sendezeitpunkt und überträgt sie dann an die Realzeitadapter-Hardware. Die zeitgesteuerte Sendeeinheit (engl.: Time Triggered Send Unit, TTS) entnimmt die Nachricht genau dann aus der Sendewarteschlange, wenn der Soll-Sendezeitpunkt erreicht ist und versendet die Nachricht auf dem CAN-Bus.

Der exakte Sendezeitpunkt wird über die Zeitstempeleinheit bestimmt und über die Sendeverzugseinheit (engl.: Transmit Delay Unit, TXD) mit dem Soll-Zeit-

punkt verglichen. Ist die Abweichung größer als die vorkonfigurierte Toleranz, so wird in der Fehlerwarteschlange eine Fehlermeldung bereit gestellt, die von der Restbussimulation zu einem geeigneten Zeitpunkt abgerufen werden kann.

Über den Sendemechanismus der Programmierschnittstelle ist es nicht nur möglich, Nachrichten an einen CAN-Bus zu verschicken, sondern es können auch Nachrichten zu einem gewünschten Zeitpunkt an die Programmierschnittstelle verschickt werden. Über diesen Callback-Mechanismus kann die Restbussimulation zu einem bestimmten Zeitpunkt aufgeweckt werden. Der Zeitpunkt ist durch diesen Mechanismus unabhängig von der Uhr auf dem Arbeitsplatzrechner. Er basiert auf der Zeit im Realzeitadapter.

Wenn es aus Restbussimulationssicht erforderlich ist, statt einer lokalen Zeitbasis eine global genaue Zeitbasis zu verwenden, so kann die lokale Uhr über die Uhrregelung (engl.: Clock Controller, CLC) beispielsweise über eine GPS-Einheit auf die globale Zeit geregelt werden.

4.2.1. Uhrregelung

Die Uhrregelung besteht aus zwei Teilen, einem Hardware-Teil und einem Software-Teil. In Summe muss die Uhrregelung die Aufgaben aus Definition 3 erfüllen. Die zeitkritischen Teile wurden dabei in Hardware umgesetzt, während die unkritischen in Software implementiert sind.

Definition 3. *Uhrregelung (engl.: Clock Controller, CLC)*
Die Uhrregelung ist eine Einheit des Realzeitadapters, die die lokale Uhr beinhaltet und gegebenenfalls auf die GPS-Zeit regelt. Aufgabe der Uhrregelung ist es, die Abweichung und Drift zur GPS-Zeit zu berechnen und zu kompensieren.

Der Hardware-Teil ist in Abbildung 4.6 dargestellt. Die Eingangsgrößen t_global und Summand_ns werden von der Uhrregelungssoftware übergeben. Im Gegenzug wird ns@PPS an die Software geliefert. Der Trigger mit einer Periode von einer Sekunde (engl.: Pulse Per Second, PPS) wird direkt vom GPS-Schaltkreis bereitgestellt.

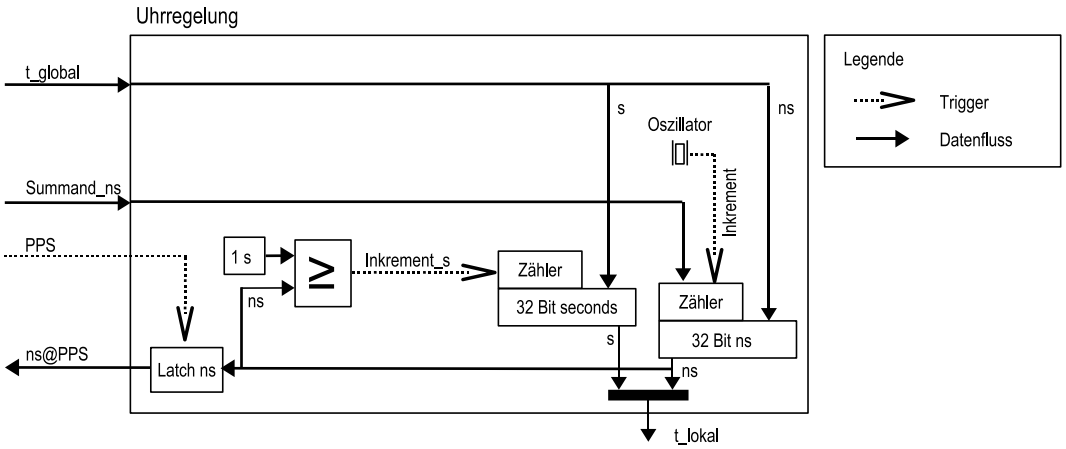


Abbildung 4.6.: Hardware-Anteil der Uhrregelung

Aufgabe der Uhrregelung ist es zunächst basierend auf dem lokalen Oszillator, eine lokale Uhr zur Verfügung zu stellen. Die lokale Uhr besteht aus zwei Zählern. Der eine bildet die Nanosekunden, der andere die Sekunden jeweils in einem 32 Bit Register ab. Ist beispielsweise ein Oszillator mit einer spezifizierten Frequenz von 10MHz verbaut entspricht dies einer Periodendauer von 100ns (4.1). Das heißt, mit jedem Impuls vom Oszillator muss das Register für die Nanosekunden um 100 erhöht werden. Dieser Wert wird über die Variable Summand_ns eingestellt.

$$Periodendauer\ T = \frac{1}{Frequenz\ f}. \tag{4.1}$$

Im Falle einer Synchronisation auf die globale GPS-Zeit, kann die fertigungsbedingte Ungenauigkeit des lokalen Oszillators durch eine Anpassung des Summanden kompensiert werden. Hierzu wird über ns@PPS der Nanosekundenanteil der lokalen Uhr zum Zeitpunkt eines GPS-Sekundenpulses gespeichert. Über die Differenz zwischen zwei aufeinanderfolgenden Werten in ns@PPS kann die Uhrregelungssoftware die Abweichung der lokalen Quarzfrequenz berechnen und über eine Anpassung des Summanden nach Gleichung 4.2 kompensieren.

$$Summand\_ns = Summand\_ns\_alt \cdot \frac{10^9}{ns@pps}. \tag{4.2}$$

4.2.2. Zeitstempeleinheit

Zur Erfüllung ihrer Aufgabe nach Definition 4 wird die Zeitstempeleinheit durch den Beginn einer CAN-Nachricht auf dem CAN-Bus getriggert. Dieser Trigger kommt vom CAN-Controller, sobald er das Signal Start Of Frame (SOF) auf dem CAN-Bus erkennt. Die Zeitstempeleinheit speichert daraufhin den aktuellen Wert der lokalen Uhr t_lokal im Register t_RX (Abbildung 4.7).

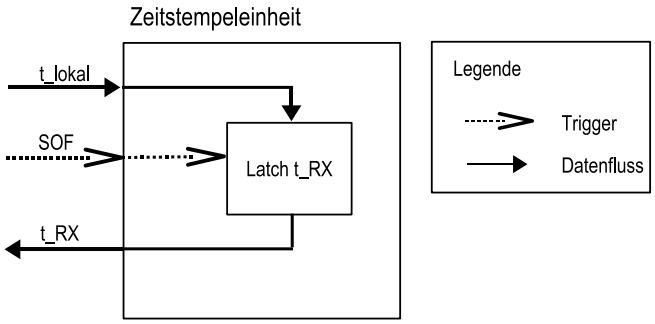


Abbildung 4.7.: Zeitstempeleinheit

Da beim CAN-Bus alle gesendeten Nachrichten über die Empfangseinheit zurückgelesen werden, entspricht dieser Wert auch dem Sendezeitpunkt einer ausgehenden Nachricht t_TX_ist. Es ist Aufgabe der Empfangs- und Sendefunktion im Mikrokontroller, die im Register abgelegten Zeitstempel der entsprechenden Nachricht zuzuordnen.

Definition 4. *Zeitstempeleinheit (engl.: Time Stamping Unit, TSU)*
Die Zeitstempeleinheit ist eine logische Einheit des Realzeitadapters, die zu eingehenden und ausgehenden Nachrichten eines CAN-Busses den Zeitpunkt des Start Of Frame (SOF) auf Basis der lokalen Zeit in einem Register speichert.

4.2.3. Empfangswarteschlange

Nach Definition 5 kann die Restbussimulation alle für sie bestimmten Nachrichten an der Empfangswarteschlange abholen. Zum einen können dies Nachrichten von einem CAN-Bus sein, die schnellstmöglich an die Restbussimulation weitergeleitet werden. Zum anderen können dies Nachrichten sein, die die Restbussimulation zu einem früheren Zeitpunkt an sich selbst verschickt hat und deren Sendezeitpunkt erreicht ist. Über diesen Mechanismus kann die Restbussimulation sich unabhängig von der Uhr auf dem Arbeitsplatzrechner triggern lassen.

Definition 5. *Empfangswarteschlange (engl.: Receive Queue, RXQ)*
Die Empfangswarteschlange ist eine logische Einheit des Realzeitadapters, über die Nachrichten, die an die Restbussimulation adressiert sind, von dieser abgeholt werden können.

4.2.4. Zeitgesteuerte Sendeeinheit

Abbildung 4.8 zeigt, dass die zeitgesteuerte Sendeeinheit nach Definition 6 über Nachricht_TX die zu sendende Nachricht erhält. Der gewünschten Sendezeitpunkt der Nachricht wird in t_TX_soll übergeben. Die zu sendende Nachricht aus Nachricht_TX wird in den Sendepuffer geladen und beim Erreichen des gewünschten Sendezeitpunkts über CC_TX an den CAN-Controller übergeben.

Definition 6. *Zeitgesteuerte Sendeeinheit (engl.: Time Triggered Send Unit, TTS)*
Die zeitgesteuerte Sendeeinheit ist ein Hardware-Modul des Realzeitadapters, das die zu versendenden Nachrichten zeitlich präzise auf dem CAN-Bus ausgibt.

4.2.5. Sendewarteschlange

Die Sendewarteschlange nach Definition 6 besteht aus Software-Anteilen, die sich auf die Programmierschnittstelle und den Mikrokontroller verteilen. Der Anteil in der Programmierschnittstelle bringt alle zu sendenden Nachrichten in eine zeitliche sortierte Reihenfolge und übergibt alle Nachrichten, die innerhalb der nächsten 30 Sekunden gesendet werden sollen, an den Mikrokontroller. Im Falle

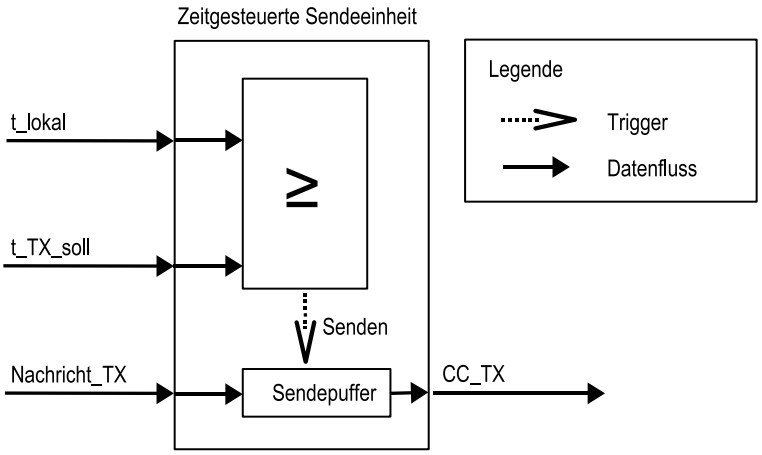


Abbildung 4.8.: Zeitgesteuerte Sendeeinheit

von CAN-Nachrichten puffert dessen Software die Nachrichten bis wenige Mikrosekunden vor der gewünschten Sendezeit und lädt die Nachricht dann über Nachricht_TX in den Sendepuffer der zeitgesteuerten Sendeeinheit. Gleichzeitig wird der gewünschte Sendezeitpunkt in t_TX_soll abgelegt. Danach ist es Aufgabe der zeitgesteuerten Sendeeinheit, bei Erreichen des Sendezeitpunkts die Nachricht an den CAN-Controller zu übergeben.

Durch die Implementierung des letzten Anteils der Sendewarteschlange in Hardware in Form der zeitgesteuerten Sendeeinheit, kann ein zeitlich hochpräzises Versenden der CAN-Nachrichten sichergestellt werden.

Im Falle einer Nachricht, die die Restbussimulation zu einem späteren Ankunftszeitpunkt an sich selbst adressiert hat, wartet der Mikrokontroller bis dieser Zeitpunkt erreicht ist und schickt die Nachricht dann über die Empfangswarteschlange wieder zurück zur Restbussimulation.

Definition 7. *Sendewarteschlange (engl.: Transmit Queue, TXQ)*
Die Sendewarteschlange ist eine logische Einheit des Realzeitadapters. Über die Programmierschnittstelle zur Sendewarteschlange kann von der Restbussimulation eine Nachricht mit einem Soll-Sendezeitpunkt übergeben werden.

4.2.6. Sendeverzugseinheit

Die Sendeverzugseinheit ist in Abbildung 4.9 dargestellt. Nach Definition 8 vergleicht sie die gewünschte Sendezeit t_TX_soll mit der tatsächlichen Sendezeit t_TX_ist. Übersteigt die Differenz die vorgegebene Toleranz, so wird der Wert im Register Verzug bereitgestellt.

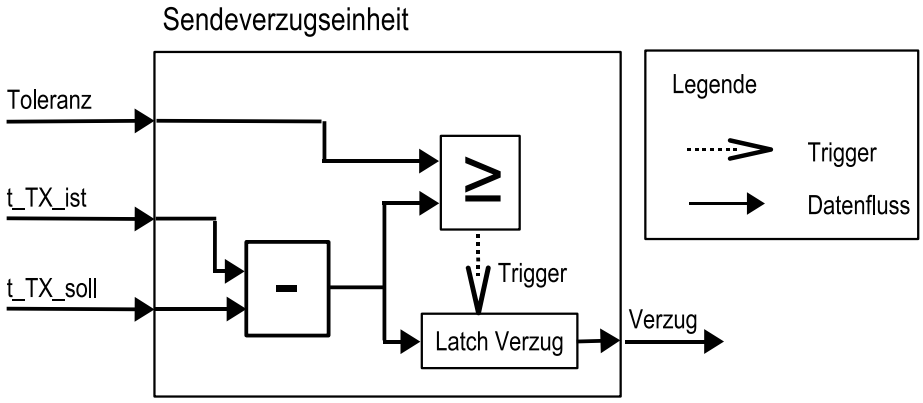


Abbildung 4.9.: Sendeverzugseinheit

Definition 8. *Sendeverzugseinheit (engl.: Transmit Delay Unit, TXD)*
Die Sendeverzugseinheit ist eine logische Einheit des Realzeitadapters, die bei ausgehenden Nachrichten den Soll-Sendezeitpunkt mit dem tatsächlichen Sendezeitpunkt vergleicht. Das Ergebnis wird über die Fehlerwarteschlange bereitgestellt, wenn es eine Toleranz überschreitet.

4.2.7. Fehlerwarteschlange

Die Aufgabe der Fehlerwarteschlange nach Definition 9 ist es, Fehler im Realzeitadapter an die Restbussimulation zu melden. Insbesondere sind dies Verzögerungen beim Versenden von Nachrichten, die von der Sendeverzugseinheit gemeldet wurden. Des weiteren werden Fehler vom GPS-Schaltkreis und vom CAN-Controller weitergeleitet. Außerdem werden Fehler aus der Mikrokontroller-Software gemeldet, beispielsweise wenn die Synchronität zur GPS Zeit nicht

mehr gewährleistet ist.

Definition 9. *Fehlerwarteschlange (engl.: Error Queue, ErrQ)*
Die Fehlerwarteschlange ist eine logische Einheit des Realzeitadapters, über die Fehler innerhalb des Realzeitadapters der Restbussimulation zur Verfügung gestellt werden. Dabei werden insbesondere Zeitverletzungen beim Senden von Nachrichten, die außerhalb einer vorgegebenen Toleranz liegen, der Restbussimulation gemeldet.

4.2.8. Realzeitadapter mit vier CAN-Bussen

Der im Rahmen dieser Arbeit aufgebaute Prototyp des Realzeitadapters verfügt über vier CAN-Controller und kann somit mit bis zu vier verschiedenen CAN-Bussen gekoppelt werden. In Abbildung 4.10 ist der Übersicht halber der Hardware-Aufbau eines Realzeitadapters mit lediglich zwei CAN-Controllern dargestellt. Es ist zu sehen, wie für jeden CAN-Bus der als Schaltkreis vorhandene CAN-Tranceiver mit dem CAN-Controller und dieser mit der Zeitstempeleinheit, der zeitgesteuerten Sendeeinheit, der Sendeverzugseinheit und dem Mikrokontroller im FPGA verbunden sind.

Das GPS-Modul, die Uhrregelung und die Anbindung zum Arbeitsplatzrechner über Ethernet sind jeweils nur einfach vorhanden.

Im Mikrokontroller läuft ein Programm ab, das beim Start zunächst die Sequenz in Listing 4.1 durchführt. Zuerst wird die an der Programmierschnittstelle definierte Toleranz für die Sendegenauigkeit an den verschiedenen CAN-Bussen in die entsprechenden Registern der Sendeverzugseinheiten geschrieben. Außerdem werden die CAN-Controller und der GPS-Schaltkreis initialisiert. Im Anschluss kann die globale Zeit vom GPS-Schaltkreis in die lokale Uhr übernommen werden. Nach Abschluss der Initialisierung und sobald von der Programmierschnittstelle das Signal Startup auf FALSE gesetzt wird, verlässt das Programm die Startsequenz.

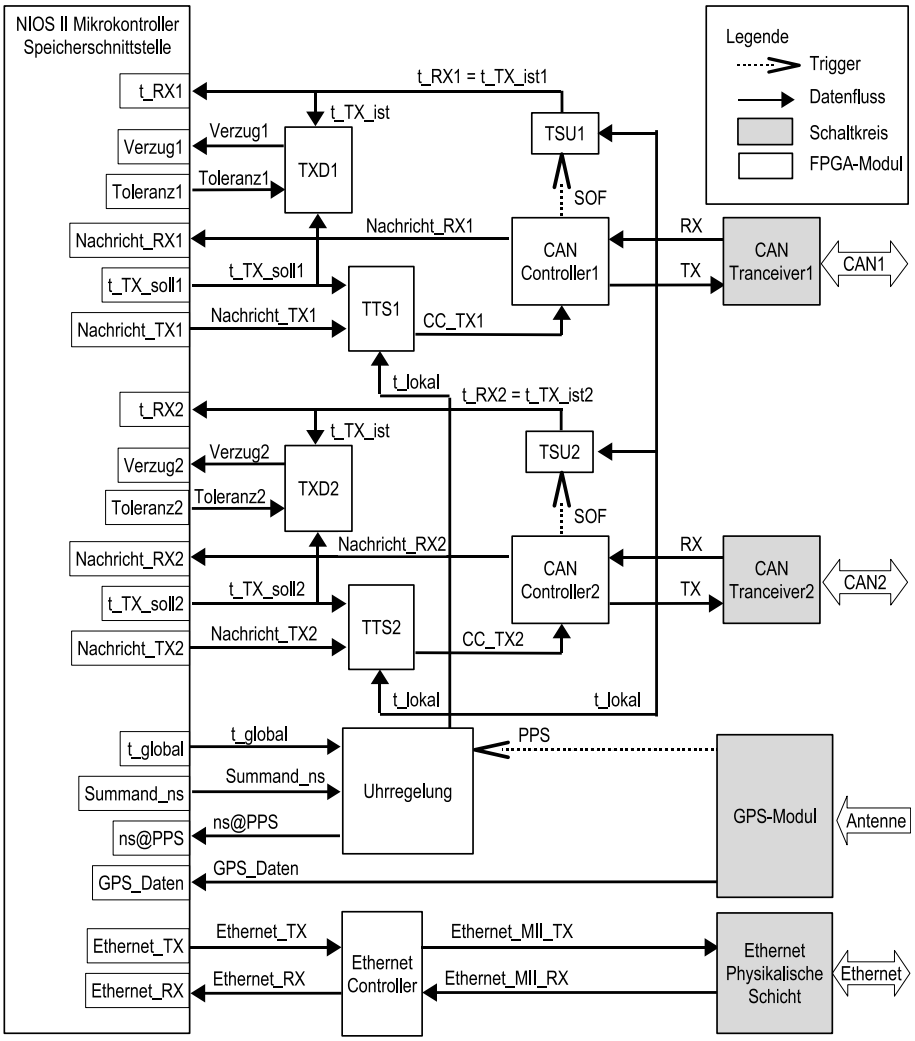


Abbildung 4.10.: Exemplarischer Aufbau eines Realzeitadapters mit zwei CAN-Bussen

```
bool Startup = TRUE;
do{
 verarbeite_Ethernet(Ethernet_RX, &Startup);
 setzte_Toleranzen(&Toleranz1, &Toleranz2);
 initialisiere_CAN1();
 initialisiere_CAN2();
 initialisiere_CAN3();
 initialisiere_CAN4();
 stelle_lokale_Uhr(GPS_Daten, &t_global);
} while(Startup)
```

Listing 4.1: Mikrokontrollerprogramm zur Initialisierung des Realzeitadapters

Danach geht das Programm in den in Listing 4.2 abgebildeten Betriebsmodus. Hier werden zunächst die Empfangspuffer der CAN-Busse geleert und eine empfangene Nachricht mit deren zugehörigen Zeitstempel verknüpft. Danach werden die Nachrichten mit ihren Zeitstempeln für den Versand zur Programmierschnittstelle intern abgelegt. Als nächstes werden die zum Versand über einen der CAN-Busse fälligen Nachrichten mit zugehörigen Soll-Sendezeitpunkten der zum jeweiligen CAN-Bus gehörigen zeitgesteuerten Sendeeinheit zur Verfügung gestellt. In verarbeite_Errors() werden anschließend eventuell aufgetretene Zeitverstöße an die Programmierschnittstelle weitergeleitet. In verarbeite_GPS() wird die Zeitsynchronität zu GPS verifiziert und die empfangenen Ortssignale an die Programmierschnittstelle verschickt. Danach wird in verarbeite_PPS() der neue Summand nach Gleichung 4.2 berechnet. Zum Schluss eines Zyklus wird die Kommunikation zur Programmierschnittstelle über Ethernet durchgeführt.

```
while(1){
    verarbeite_CAN1_RX(t_RX1, Nachricht_RX1);
    verarbeite_CAN2_RX(t_RX2, Nachricht_RX2);

    verarbeite_CAN1_TX(t_TX_soll1, Nachricht_TX1);
    verarbeite_CAN2_TX(t_TX_soll2, Nachricht_TX2);

    verarbeite_Errors(Verzug1, Verzug2);
    verarbeite_GPS(GPS_Daten);
    verarbeite_PPS(ns@PPS, &Summand_ns);
    verarbeite_Ethernet(Ethernet_RX ,&Ethernet_TX);
}
```

Listing 4.2: Mikrokontrollerprogramm im Betriebsmodus

An der Programmierschnittstelle können die Warteschlangen über folgende Funktionen verwendet werden:

- Die Funktion `RTA_receive(&Nachricht, &Zeitstempel)` gibt die erste Nachricht mit zugehörigem Zeitstempel aus der Empfangswarteschlange zurück.

- Die Funktion `RTA_send(Nachricht, Zeitstempel, Ziel)` verschickt eine Nachricht zum gewünschten Zeitpunkt an ein vorgegebenes Ziel (an einen der CAN-Busse oder an die Programmierschnittstelle).

- Die Funktion `RTA_getError(&Error)` liefert die erste Fehlermeldung aus der Fehlermeldungwarteschlange.

4.3. Definition des Zeitverhaltens

In Listing 4.3 ist beispielhaft dargestellt, wie ein Simulationsprogramm eine zyklische Kommunikation über einen CAN-Bus mit Hilfe des Realzeitadapters erzeugen kann. Sollen Nachrichten periodisch verschickt werden, so muss in der Simulation bei jedem Zyklus der Sendezeitpunkt einer Nachricht um eine Periodendauer erhöht werden.

```
// Aktuelle Zeit vom Realzeitadapter holen
int timer = RTA_getTime();
while(1){
    // Den timer um 10ms erhöhen
    timer = timer + 10;

    // Nachrichten von den CAN-Bussen empfangen
    RTA_receive(&NachrichtRX, &Zeitstempel);

    // Auf Basis der empfangenen Nachrichten die Simulation berechnen
    NachrichtTX = simulate(NachrichtRX);

    // Das Ergebnis verschicken
    RTA_send(NachrichtTX, timer, CAN1);

    // Fehler auslesen und bei einem Fehler Testfall abbrechen
    RTA_getError(&Error);
    if(Error) break;
}
```

Listing 4.3: Beispiel für zyklisches Versenden von Nachrichten

Soll von einem zyklischen Verhalten abgewichen werden, kann jeder einzelne Soll-Sendezeitpunkt manipuliert werden, wie es in Listing 4.4 dargestellt ist.

```
// Aktuelle Zeit vom Realzeitadapter holen
int timer = RTA_getTime();
// Den Startzeitpunkt merken
int start = timer;
while(1){
    // Den timer um 10ms erhöhen
    timer = timer + 10;

    // Nachrichten von den CAN-Bussen empfangen
    RTA_receive(&NachrichtRX, &Zeitstempel);

    // Auf Basis der empfangenen Nachrichten die Simulation berechnen
    NachrichtTX = simulate(NachrichtRX);

    // 2 Sekunden nach Simulationsstart eine Nachricht um 1 ms verzögern
    if(timer == start + 2000)
        RTA_send(NachrichtTX, timer + 1, CAN1)
    else
        RTA_send(NachrichtTX, timer, CAN1);

    // Fehler auslesen und bei einem Fehler Testfall abbrechen
    RTA_getError(&Error);
    if(Error) break;
}
```

Listing 4.4: Beispiel für die Manipulation eines einzelnen Sendezeitpunkts

Wenn der Sendezeitpunkt einer Nachricht immer relativ zum Empfangszeitpunkt einer anderen Nachricht sein soll, so kann dies wie in Listing 4.5 gezeigt umgesetzt werden.

```
while(1){
    // Nachrichten von den CAN-Bussen empfangen
    RTA_receive(&NachrichtRX, &Zeitstempel);

    // Auf Basis der empfangenen Nachrichten die Simulation berechnen
    NachrichtTX = simulate(NachrichtRX);

    // Das Ergebnis 5 ms nachdem die NachrichtRX einging verschicken
    RTA_send(NachrichtTX, Zeitstempel + 5, CAN1);

    //Fehler auslesen und bei einem Fehler Testfall abbrechen
    RTA_getError(&Error);
    if(Error) break;
}
```

Listing 4.5: Beispiel für das Versenden von Nachrichten relativ zu einem Empfangszeitpunkt

Ulmer und Wittel zeigen in [125], dass die Programmierschnittstelle des Realzeitadapters an grafische Modellierungswerkzeuge angebunden werden kann. Als Beispiel wird eine Anbindung an MATLAB/Simulink [71] vorgestellt. In diesem Fall kann in Simulink sowohl ein Datenmodell als auch ein Zeitmodell einzelner Nachrichten abgebildet werden. Es wird gezeigt, dass dieser Teil der Umgebungssimulation als Simulink-Modell integriert werden kann. Dies ermöglicht schnelle Änderungen im Modell, ohne dass das Modell im Anschluss auf eine Echtzeitumgebung geladen werden muss. Außerdem können die Modellgrößen während einer Echtzeitsimulation wie aus vorangegangenen Software-Simulationen direkt in Simulink betrachtet werden.

4.4. Messungen des am Prototypen erreichten Zeitverhaltens

Im Folgenden soll durch Messungen die im prototypischen Aufbau eines Realzeitadapters erreichte Genauigkeit beim Empfang und beim Senden von CAN-Nachrichten dargestellt werden. Nach einer kurzen Beschreibung des Versuchsaufbaus soll zunächst gezeigt werden, dass ein periodisches Senden von CAN-

Nachrichten durch den Realzeitadapter von einem nicht echtzeitfähigen Arbeitsplatzrechner in hoher zeitlicher Präzision möglich ist. Danach wird die Synchronität zweier über GPS synchronisierter Realzeitadapter hinsichtlich der Empfangszeitstempel untersucht. Im Anschluss wird die Sendegenauigkeit zweier Realzeitadapter, die über GPS synchronisiert sind, betrachtet. Die Versuche wurden für alle CAN-Schnittstellen des Realzeitadapters und für verschiedene Zykluszeiten bei den versendeten Nachrichten durchgeführt. Im Folgenden sollen die Ergebnisse für eine der vier CAN-Schnittstellen eines Realzeitadapters und für die bei Fahrerassistenzsteuergeräten verwendete minimale Zykluszeit von 10ms betrachtet werden.

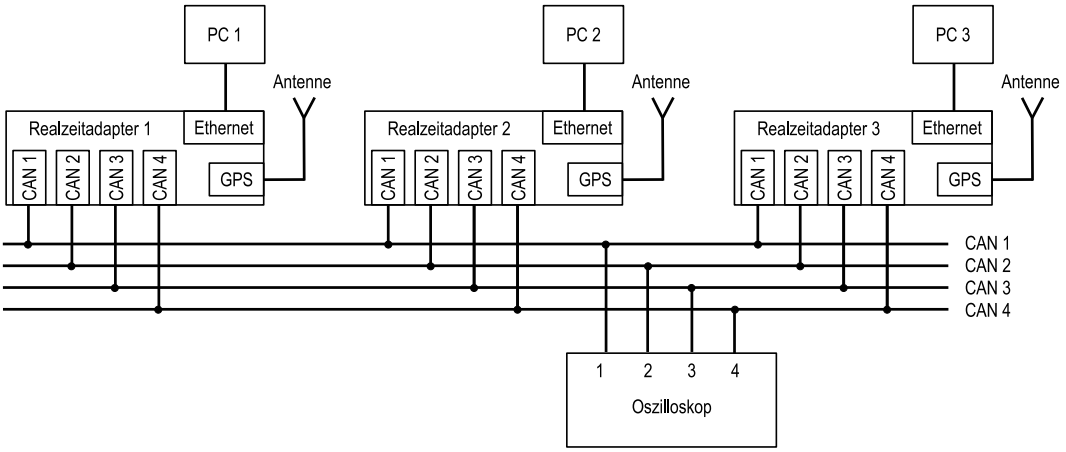


Abbildung 4.11.: Versuchsaufbau zur Messung der erreichten Sende- und Empfangsgenauigkeit

Abbildung 4.11 zeigt den Versuchsaufbau für die durchgeführten Versuche. Dabei wurden drei Realzeitadapter mit zugeordnetem Arbeitsplatzrechner (PC) über deren vier CAN-Busse verbunden. Zusätzlich ist ein Oszilloskop an alle vier CAN-Busse angeschlossen, so dass eine unabhängige Messung des Zeitverhaltens erfolgen kann. Je nach Versuch können die Realzeitadapter als Sender oder als Empfänger von CAN-Nachrichten konfiguriert werden. Die CAN-Busse sind unbelastet. Das heißt, dass es außer der im jeweiligen Versuch beschriebenen Kommunikation keine weitere Kommunikation gibt. Alle drei Realzeitadapter verfügen über eine Verbindung zum GPS und synchronisieren sich auf die empfangene globale Zeit.

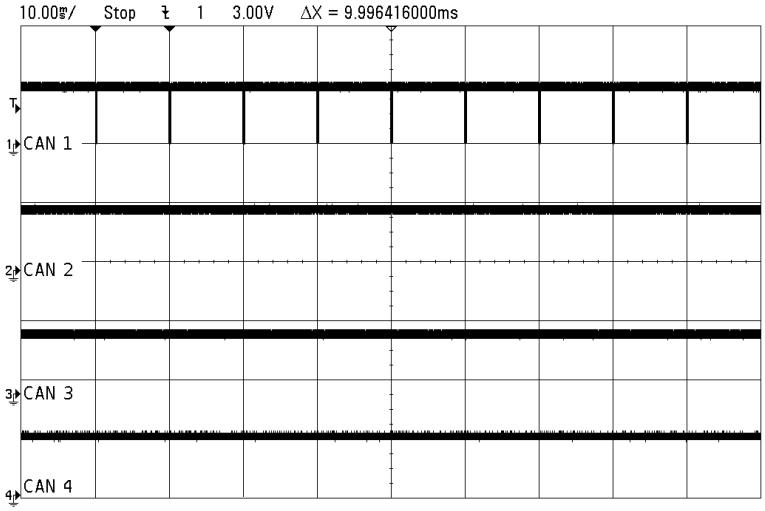


Abbildung 4.12.: Oszilloskopmessung der erreichten Zykluszeit (CAN 1 ist aktiv, CAN 2-4 sind inaktiv)

Im ersten Versuch soll über den Arbeitsplatzrechner und Realzeitadapter 1 auf CAN 1 eine CAN-Nachricht mit einer Zykluszeit von $10ms$ verschickt werden. Abbildung 4.12 zeigt das erreichte Zeitverhalten mit Hilfe einer Oszilloskopmessung. Das Oszilloskop wurde dabei so programmiert, dass es auf den Start-of-Frame einer CAN-Nachricht triggert. Die Messung mit dem Oszilloskop zeigt, dass zwischen den ersten beiden dargestellten Nachrichten $9,996ms$ liegen, was die in Definition 1 geforderte Präzision um Größenordnungen unterbietet. Für statistische Auswertungen der erreichten Verteilung über mehrere Nachrichten hinweg sei auf Kapitel 6.1 verwiesen, wo die Sendegenauigkeit des Realzeitadapters beim Vergleich verschiedener Testsysteme untersucht wird.

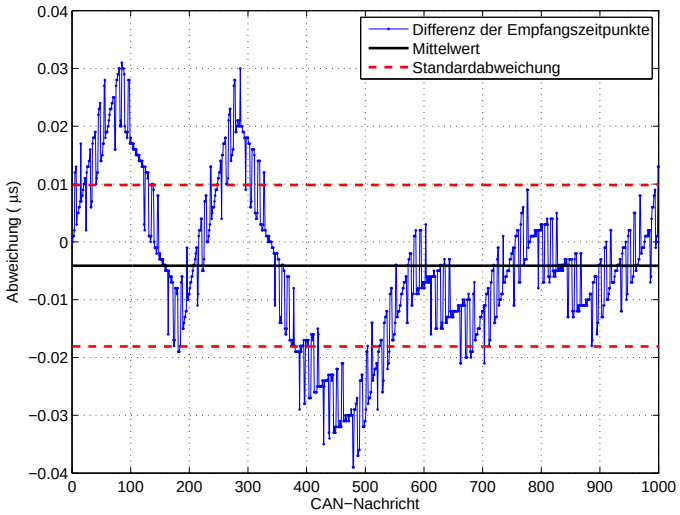


Abbildung 4.13.: Unterschied in den Empfangszeitstempeln zweier Realzeitadapter

Für die Untersuchung der Präzision der Empfangszeitstempel zweier Realzeitadapter wurden mit Realzeitadapter 1 CAN-Nachrichten auf CAN 1 mit einer Zykluszeit von $10ms$ verschickt und mit den Realzeitadaptern 2 und 3 empfangen. Letztere sind über GPS synchronisiert und versehen jede empfangene Nachricht mit einem Empfangszeitstempel.

Es soll der Empfangszeitstempel der Nachricht n an Realzeitadapter 2 mit $t_{n,RZA2}$ und mit $t_{n,RZA3}$ bei Realzeitadapter 3 bezeichnet werden. In Abbildung 4.13 ist die im Anschluss an die Messung berechnete Abweichung der Empfangszeitstempel $t_{n,RZA2} - t_{n,RZA3}$ der beiden Realzeitadapter für 1000 CAN-Nachrichten dargestellt. Jeder Punkt entspricht genau einer Nachricht.

Aus den eingezeichneten Geraden für den Mittelwert und der Standardabweichung sowie aus dem Histogramm in Abbildung 4.14 geht hervor, dass die durchschnittliche Abweichung der Empfangszeitstempel in diesem Experiment $4ns$ mit einer Standardabweichung von $14ns$ beträgt. Das in Abbildung 4.13 erkennbare schwingende Verhalten ist eine Überlagerung der Uhrregelungen in den beiden Realzeitadaptern. Auf Grund der geringen Amplitude ist dieser Effekt für das Testsystem nicht relevant.

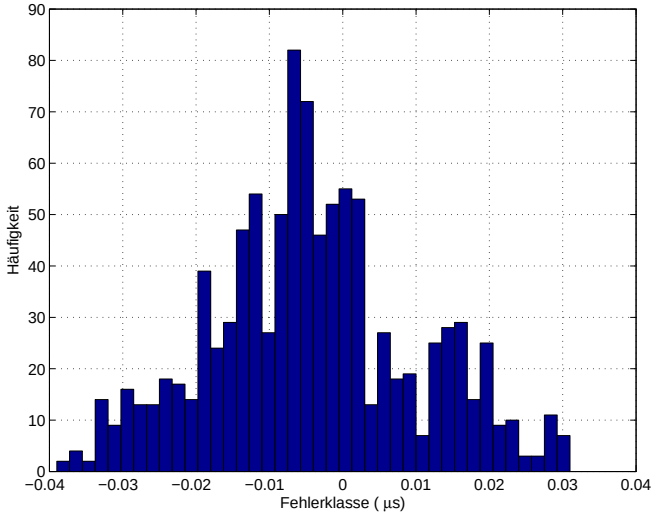


Abbildung 4.14.: Histogramm der Empfangsgenauigkeit zweier Realzeitadapter

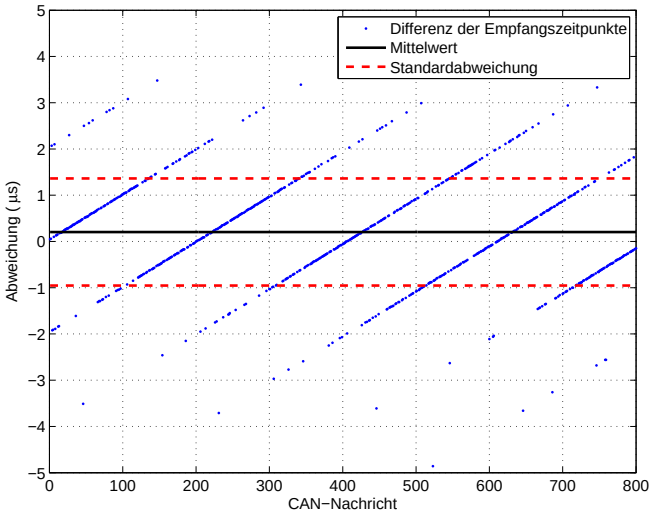


Abbildung 4.15.: Sendegenauigkeit zweier Realzeitadapter bei zeitgesteuertem Senden

Zur Untersuchung der Sendegenauigkeit zweier Realzeitadapter zueinander wurden Realzeitadapter 1 und Realzeitadapter 2 als Sender konfiguriert. Ersterer sendet mit einer Zykluszeit von $10ms$ auf CAN 1 und letzterer auf CAN 2. Beide versenden die selben CAN-Nachrichten mit den selben Soll-Sendezeitpunkten bezogen auf die vom GPS empfangene globale Zeit. Realzeitadapter 3 empfängt die Nachrichten beider Sender auf seinen CAN-Bussen 1 und 2 und versieht jede Nachricht mit einem Empfangszeitstempel. In Abbildung 4.15 ist die auf Basis der Aufzeichnungen von Realzeitadapter 3 berechnete Abweichung der tatsächlichen Sendezeitpunkte für 800 Nachrichten dargestellt. Jeder Punkt entspricht dabei genau einer CAN-Nachricht.

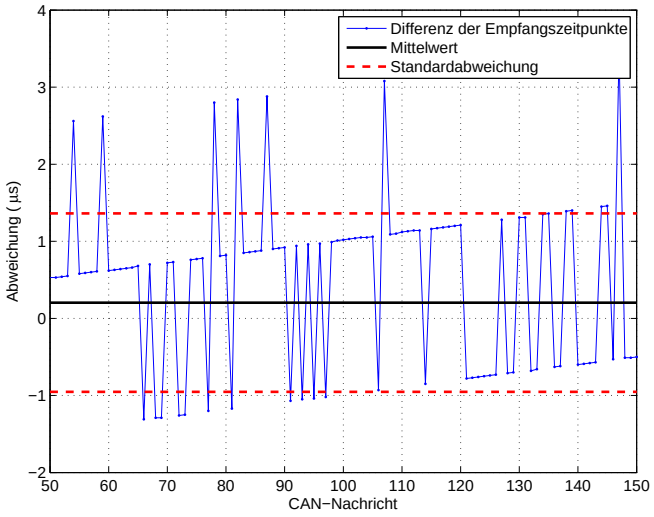


Abbildung 4.16.: Sendegenauigkeit zweier Realzeitadapter bei zeitgesteuertem Senden (Ausschnitt)

In der vergrößerten Darstellung des Bereichs von Nachricht 50 bis 150 in Abbildung 4.16 ist zu sehen, dass die Abweichung der Empfangszeitpunkte der Nachrichten mit einer Drift versehen sind und auf drei Geraden mit einem Abstand von circa $2\mu s$ liegen. Das Phänomen, dass sich die Abweichungen auf mehreren Geraden verteilen, ist der prototypischen Implementierung des Realzeitadapters zuzuschreiben und soll hier nicht näher untersucht werden, da sich die durchschnittliche Abweichung weit innerhalb der gesteckten Grenzen befindet. Der dargestellte Durchschnittswert liegt bei $204ns$ mit einer Standardabweichung von $1,16\mu s$. Die erreichte Verteilung über alle 800 Nachrichten ist im Histogramm

in Abbildung 4.17 dargestellt.

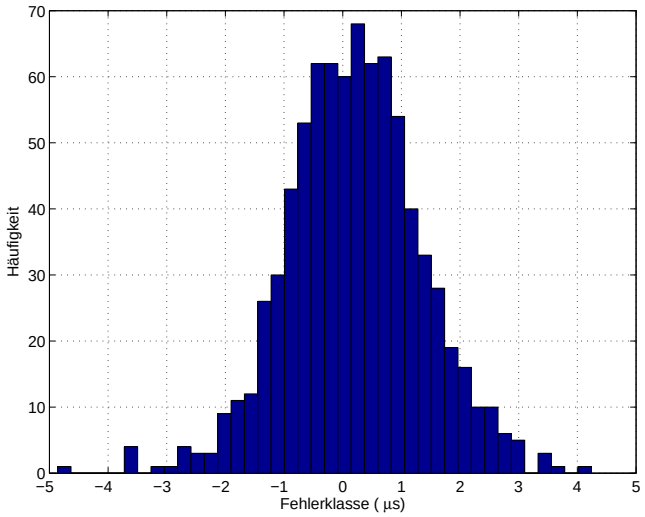


Abbildung 4.17.: Histogramm der Sendegenauigkeit zweier Realzeitadapter

4.5. Zusammenfassung des Konzepts

Der Realzeitadapter zusammen mit einem Arbeitsplatzrechner ermöglichen ein Zeitstempel-basiertes Testsystem, das ubiquitär eingesetzt werden kann. In Tabelle 4.1 ist zusammengefasst, wie das Zeitstempel-basierte Testsystem die Anforderungen von Kapitel 4 erfüllt. Zunächst erfüllt es die auf Basis der bestehenden Testsysteme geforderten Eigenschaften wie echtzeitfähig, preisgünstig, ubiquitär und für CAN geeignet aus den Anforderungen 1, 3, 4 und 7. Außerdem kann die geforderte Synchronität zu einer globalen Zeit sowohl für den Empfang von CAN-Nachrichten (Anforderung 5) als auch für deren Versenden (Anforderung 6) durch die Synchronisation der internen Uhr auf die GPS-Zeit gewährleistet werden. Durch die zusätzlich zur Zeitstempeleinheit und zeitgesteuerten Sendeeinheit eingebaute Sendeverzugseinheit wird, wie in Anforderung 8 gefordert, sichergestellt, dass ein Verstoß gegen Echtzeitanforderungen stets erkannt wird. Da beim zeitgesteuerten Senden für jede CAN-Nachricht der gewünschte

Sendezeitpunkt angegeben werden muss, kann dieser für jede einzelne Nachricht individuell gewählt werden. Hierdurch ist Anforderung 2 erfüllt.

Im Zuge der Arbeit hat sich herausgestellt, dass der Realzeitadapter in Kombination mit einem Arbeitsplatzrechner nicht nur für den Einsatz als Zeitstempel-basiertes Testsystem am Arbeitsplatz geeignet ist, sondern ebenso als Zeitstempel-basiertes Testsystem zur Mitnahme auf ein Prüfgelände und als Datenlogger für interagierende Fahrzeuge eingesetzt werden kann. Der Name *ubiquitäres Testsystem* scheint daher durchaus gerechtfertigt und soll somit im weiteren Verlauf für dieses Testsystem verwendet werden.

Anforderung	Kapitel	Lösung
1: Echtzeitfähigkeit	4.1, 4.4	Die Echtzeitfähigkeit wird durch das zeitgesteuerte Senden von Nachrichten sichergestellt.
2: Flexible Beeinflussung des Zeitverhaltens	4.3	Das Zeitverhalten jeder Nachricht kann in der Restbussimulation beeinflusst werden.
3: Preisgünstig	4.1	Ein Arbeitsplatzrechner und ein Realzeitadapter genügen für den Aufbau eines Testsystems.
4: Ubiquitär	4.1	Das Testsystem kann aus einem Laptop mit Realzeitadapter bestehen und ist somit transportabel.
5: Aufzeichnung von Nachrichten mit Bezug zu einer globalen Uhrzeit	4.2.1, 4.4	Sichergestellt durch die Synchronisationsmöglichkeit der lokalen Uhr zur GPS-Zeit.
6: Abspielen von Nachrichten mit Bezug zu einer globalen Uhrzeit	4.2.1, 4.4	Sichergestellt durch die Synchronisationsmöglichkeit der lokalen Uhr zur GPS-Zeit in Kombination mit zeitgesteuertem Senden von Nachrichten.
7: Anbindung des Testobjekts über CAN	4.2.8	Der vorgestellte Realzeitadapter hat vier CAN-Schnittstellen.
8: Erkennen einer Verletzung von Echtzeitanforderungen	4.2.6	Die Sendeverzugseinheit erkennt und meldet Abweichungen des Sendezeitpunkts außerhalb einer vorgegebenen Toleranz.

Tabelle 4.1.: Erfüllung der gestellten Anforderungen durch das vorgestellte Konzept

5. Anwendung des Konzepts bei Fahrerassistenzsystemen

In diesem Kapitel wird beschrieben, wie der Realzeitadapter für den Test auf den Integrationsstufen Fahrzeug und Steuergerät, die in Kapitel 3.1 beschrieben sind, eingesetzt werden kann und welche Auswirkungen der Einsatz des Realzeitadapters auf den Entwicklungsprozess hat. In Kapitel 6 sind Experimente zum Einsatz des Realzeitadapters beim Test von Fahrerassistenzfunktionen im Fahrzeug und an einem Steuergerätetestsystem zu finden.

Kapitel 5.1 beschreibt, wie der Realzeitadapter beim Test von Fahrerassistenzsystemen auf der Integrationsstufe Fahrzeug als Datenlogger eingesetzt werden kann. Auf Basis der gesammelten Daten aus verschiedenen Fahrzeugen kann die Bewertung des Integrationsschrittes erfolgen. Sobald ein zu testendes Softwaremodul oder die gesamte eingebettete Software auf ein Steuergerät geladen wird, kann deren zeitliches Verhalten auf der Zielhardware an einem Steuergerätetestsystem untersucht werden. Das Kapitel 5.2 beschreibt, wie ein solches Steuergerätetestsystem auf Basis eines Realzeitadapters aussehen kann. Die Auswirkungen des Einsatzes eines Realzeitadapters auf den Entwicklungsprozess von Fahrerassistenzsystemen werden in Kapitel 5.3 beschrieben.

5.1. Datenlogger für die Integrationsstufe Fahrzeug

Wie in Kapitel 3.2 beschrieben wird, ist bei der Untersuchung von Fahrerassistenzsystemen im Fahrzeug ein zeitsynchrones Aufzeichnen von CAN-Nachrichten in den beteiligten Fahrzeugen nötig.

Mit Hilfe der Synchronisation des Realzeitadapters zur GPS-Zeit kann die Relativgeschwindigkeit eines vorausfahrenden Fahrzeuges, die eine Radar-ECU liefert, innerhalb einer zeitlichen Abweichung von weniger als $1ms$ aus der Kombination zweier Nachrichten aus den beiden beteiligten Fahrzeugen mit jeweils fahrzeuginternen Geschwindigkeitsinformationen verifiziert werden. Durch die Synchronisation der Datenloggeruhr auf die GPS-Zeit kann der Datenlogger die Zeitstempel der Nachrichten direkt auf die dem System übergeordnete Zeit beziehen, statt auf die Zeit des Datenloggers. D.h. analog zur Erweiterung der Systemgrenze des Systemfahrzeugs, wie sie in [2] beschrieben wird, wird die Systemgrenze der Zeitbasis ausgeweitet.

In Abbildung 5.1 sind zwei Fahrzeuge dargestellt, deren Datenlogger für die im Fahrzeug verbauten CAN-Busse die GPS-Zeit als Zeitbasis verwenden. Dies wird durch den Einsatz des Realzeitadapters realisiert. Der Realzeitadapter ermöglicht es, hochpräzise, auf GPS-Zeit basierende Zeitstempel von den Eingangszeitpunkten der CAN-Nachrichten zu generieren. Anhand der Zeitstempel können CAN-Aufzeichnungen aus unabhängigen Systemen verglichen werden, was mit zeit-

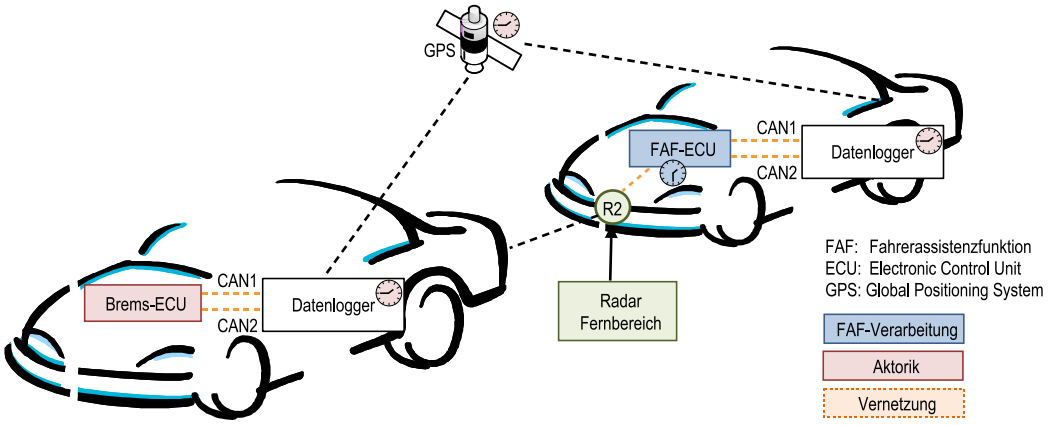


Abbildung 5.1.: Datenlogger für interagierende Fahrzeuge

lich nicht synchronen Datenloggern bisher nicht möglich war. In einem Fahrmanöver kann z. B. das hintere Fahrzeug (Systemfahrzeug) über seine Sensorik auf Aktionen des vorausfahrenden Fahrzeuges (Objektfahrzeug) reagieren. Durch den Realzeitadapter gelingt nun die Verschiebung der Zeitbasis vom lokalen System „Datenlogger im Fahrzeug“ auf das übergeordnete System „GPS“. Diese Verschiebung ermöglicht eine nachgelagerte Korrelation der aufgezeichne-

ten Daten. Die aufgezeichneten Daten der einzelnen Datenlogger sind also unabhängig von der lokalen Zeit des aufzeichnenden Datenloggers und die von verschiedenen Datenloggern gelieferten Zeitpunkte können innerhalb des übergeordneten Systems geordnet werden. Experimentelle Untersuchungen zu diesem Anwendungsfall des Realzeitadapters sind in Kapitel 6.2 zu finden.

5.2. Testsystem für die Integrationsstufe Steuergerät

Die Verwendung des Realzeitadapters an einem Testsystem ermöglicht das zeitlich hochpräzise Abspielen sowohl von Daten, die mit Hilfe des Realzeitadapters aufgezeichnet wurden, als auch von Daten, die auf Basis eines Modells in Simulationen generiert werden. Durch eine Synchronisation des Testsystems auf die GPS-Zeit wird das Testsystem ein Teil des übergeordneten Systems und kann die Aufzeichnung mit dessen Zeitbasis wiedergeben. In Abbildung 5.2 ist die Anwendung dieser Technik auf die Problemstellung aus Kapitel 3.2.3 dargestellt. Die Uhren des Datenloggers und des Testsystems sind über die GPS-Zeit synchronisiert. Dadurch kann sichergestellt werden, dass eine Aufzeichnung in derselben Geschwindigkeit abgespielt werden kann, in der sie aufgezeichnet wurde.

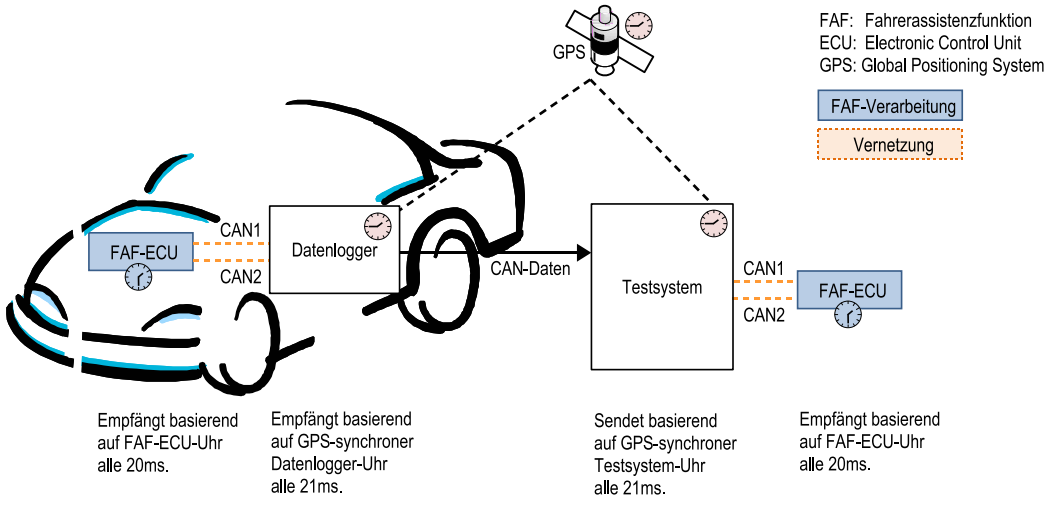


Abbildung 5.2.: Synchronisation der Uhren im Datenlogger und in einem Testsystem

In Abbildung 5.3 ist dargestellt, wie ein Steuergeräte-Testsystem mit einem Realzeitadapter realisiert werden kann. In der dargestellten Ausbaustufe können nicht nur aufgezeichnete Daten abgespielt werden, sondern auch In-the-Loop-Simulationen durchgeführt werden. Ein handelsüblicher Arbeitsplatzrechner berechnet in diesem Falle die Umgebungssimulation des Steuergerätes. Diese Umgebungssimulation besteht aus der Simulation des Fahrzeuges samt Sensorik und Aktorik, der Simulation des Fahrers und der Fahrzeugumgebung. Über die Restbussimulation werden die Simulationsergebnisse in CAN-Nachrichten verpackt, so dass sie über den CAN-Bus zum Steuergerät übertragen werden können. Der Realzeitadapter sorgt dafür, dass die CAN-Nachrichten in Echtzeit an das Steuergerät gesendet werden. Sollte z. B. auf Grund eines belegten Busses das zeitgerechte Versenden der Daten nicht möglich sein, wird dieser Fehler vom Realzeitadapter garantiert erkannt und an die Restbussimulation gemeldet. Es ist dann deren Aufgabe, zu entscheiden, wie mit der Zeitverletzung umgegangen werden soll. Bei einzelnen Zeitverletzungen innerhalb vorher festgelegter Schranken kann beispielsweise eine Warnung im Testergebnis vermerkt werden, während ab einer definierten Grenze Fehlermeldungen eingetragen werden. Eine weitere Möglichkeit ist es, den betroffenen Testfall automatisiert zu wiederholen und erst bei erneutem Auftreten der Zeitverletzungen eine Fehlermeldung einzutragen.

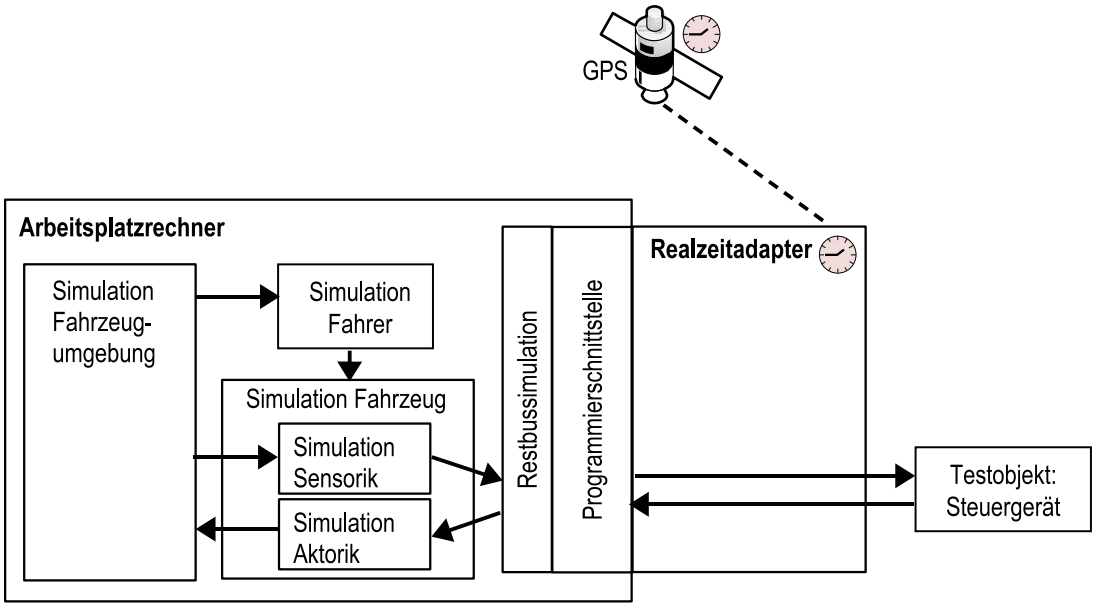


Abbildung 5.3.: Steuergeräte-Testsystem auf Basis eines Realzeitadapters

Ein Vorteil des dargestellten Testsystems ist somit, dass der auf dem Arbeitsplatzrechner implementierte Teil nicht echtzeitfähig sein muss. Der Realzeitadapter trägt die Verantwortung dafür, das Testsystem an die Zeitbasis des übergeordneten Systems *GPS* anzugleichen.

Die Kommunikation zwischen Testsystem und Testobjekt läuft folgendermaßen ab: Das Steuergerät sendet CAN-Nachrichten an das Testsystem. Diese Nachrichten werden vom Realzeitadapter empfangen. Beim Empfang einer CAN-Nachricht vergibt der Realzeitadapter einen Zeitstempel auf Basis der globalen Zeit und speichert die CAN-Nachricht zusammen mit diesem Zeitstempel ab. Der Zeitstempel enthält den genauen Empfangszeitpunkt basierend auf der GPS-Zeit. Der Realzeitadapter gibt die Informationen über den Inhalt der CAN-Nachricht inklusive Zeitstempel an den Arbeitsplatzrechner weiter. Basierend auf diesen Informationen berechnet der Arbeitsplatzrechner in seinen Modellen für Fahrer, Umgebung und Fahrzeug die Antwort für das Steuergerät, sowie den Zeitpunkt, wann diese Informationen in Form von CAN-Nachrichten an das Steuergerät gesendet werden müssen. Der gesamte Datensatz wird an den Realzeitadapter übertragen. Dieser sorgt dafür, dass die CAN-Nachrichten zum richtigen Zeitpunkt an das Steuergerät gesendet werden. D.h. für das zeitgenaue Anlegen der Ausgangsgröße an das Steuergerät und die Eskalation einer Zeitverletzung ist der Realzeitadapter verantwortlich.

Das Echtzeitverhalten wird dadurch erreicht, dass die Zeitschranken garantiert eingehalten werden, wenn der Arbeitsplatzrechner rechtzeitig die Ausgangsdaten an den Realzeitadapter liefert. Wenn diese Bedingung verletzt wird, entsteht kein größerer Schaden. Der Fehler kann sicher erkannt werden und es kann dafür gesorgt werden, dass der aktuelle Test wiederholt wird.

5.3. Einordnung in den Entwicklungsprozess

In Kapitel 3.1 wird ein Entwicklungsprozess für Fahrerassistenzsystem ohne ein ubiquitäres Testsystem beschrieben. Im Folgenden sollen die drei Kernpunkte beschrieben werden, in denen der Einsatz eines ubiquitären Testsystems auf Basis des Realzeitadapters den Entwicklungsprozess von Fahrerassistenzsystemen unterstützen kann.

Zum ersten ermöglicht der Bezug zur GPS-Zeit von Datenloggern in verschiedenen Fahrzeugen die zeitliche Korrelation und eine aufeinander bezogene Bewertung von CAN-Aufzeichnungen aus den Fahrzeugen. Zum zweiten können diese CAN-Aufzeichnungen aus Fahrzeugen mit hoher zeitlicher Präzision am Arbeitsplatz wieder abgespielt werden, so dass beispielsweise algorithmische Verbesserungen mit realen Daten getestet werden können. Zum dritten ermöglicht das preisgünstige Testsystem auf Basis des Realzeitadapters die Bereitstellung eines Testsystems für jeden Entwickler von Fahrerassistenzfunktionen. Der ubiquitäre Ansatz des Testsystems ermöglicht es, das Testsystem auf die Teststrecke mitzunehmen, so dass Änderungen am Testobjekt vor Ort zunächst am Testsystem untersucht werden können, bevor ein Fahrversuch erfolgt.

In den folgenden Kapiteln werden diese drei Aspekte erörtert und in Tabelle 5.1 zusammengefasst.

5.3.1. Bewertung von interagierenden Fahrzeugen

In Kapitel 3.1 wird dargestellt, dass der Test von Fahrerassistenzsystemen im Fahrzeug üblich und notwendig ist. Kapitel 3.2.1 erläutert, dass diese Tests aufwändig und kostenintensiv sind und somit möglichst auf ein notwendiges Minimum reduziert und effizient durchgeführt werden sollen. Kapitel 3.2.2 beschreibt, wie mit bekannten Ansätzen die Daten mehrerer Fahrzeuge mit einer zeitlich synchronen Zeitbasis aufgezeichnet werden können.

Der in [67] vorgestellte Ansatz zeigt, dass die Synchronisation der Zeitbasis oftmals erst nachträglich in bestehende Datenlogger integriert wurde und daher ungenau und aufwändig ist. In diesem Beispiel wird für die Zeitsynchronisation

eine WLAN-Verbindung mit Richtantennen zwischen den Fahrzeugen benötigt. Datenlogger auf Basis eines Realzeitadapters bestehen aus einer kleinen GPS-Antenne, einem Realzeitadapter und einem Laptop. Zwischen den Fahrzeugen muss für die Zeitsynchronisation keine Datenverbindung bestehen. Diese kostengünstige Lösung kann schnell in ein Fahrzeug integriert werden und in Kombination mit bereits verfügbaren Software-Werkzeugen für Arbeitsplatzrechner kombiniert werden, so dass die in verschiedenen Fahrzeugen aufgezeichneten Daten beispielsweise direkt auf dem Laptop ausgewertet werden können.

5.3.2. Bewertung von Fahrerassistenzfunktion im Labor

Wie in Kapitel 3.1 beschrieben ist, basiert der Entwicklungsprozess von Fahrerassistenzfunktionen oft auf einem Rapid Prototyping-Prozess. Sowohl Fahrerassistenzfunktionen, die für neue Fahrzeugtypen angepasst werden, als auch Neuentwicklungen können damit im Fahrzeug in kurzer Zeit implementiert und erlebt werden. Um Verbesserungen an der Implementierung der Fahrerassistenzfunktion bereits am Arbeitsplatz testen zu können, ist es sinnvoll, die in einem Fahrversuch aufgezeichneten CAN-Nachrichten an einem Testsystem als Eingangsgrößen für das Testobjekt zu verwenden. Wird dies mit Hilfe des Realzeitadapters am Arbeitsplatz durchgeführt, kann, basierend auf der bekannten Reaktion der Fahrerassistenzfunktion aus dem Fahrversuch, direkt am Arbeitsplatz eine Aussage getroffen werden, wie sich die Veränderungen an der Implementierung auswirken. Erst wenn diese Beurteilung positiv ausfällt, ist ein erneuter Fahrversuch sinnvoll. Wenn nur wenige Testsysteme für Fahrerassistenzsteuergeräte verfügbar sind, werden diese Tests meist erst später im Entwicklungsprozess und nur im Fahrversuch durchgeführt, was wie in Kapitel 3.2.1 dargestellt aufwändig und daher teuer ist.

Der Test eines Fahrerassistenzsteuergeräts auf Basis von aufgezeichneten CAN-Nachrichten ermöglicht nicht den Test aller Fahrerassistenzfunktionen. Sobald die Ausgangsgrößen der Fahrerassistenzfunktion wie in Abbildung 2.4 dargestellt über die Umgebung auf die Eingangsgrößen zurückwirken, ist in der Regel eine Umgebungssimulation notwendig, die das Verhalten der Umgebung in Echtzeit nachbildet. Diese Umgebungssimulation existiert oftmals in frühen Entwicklungsphasen für den Test der Software für das Fahrerassistenzsteuergerät

auf einem Arbeitsplatzrechner. Mit Hilfe eines Testsystems auf Basis des Realzeitadapters kann die für ein Software-in-the-Loop Testsystem existierende Umgebungssimulation für Hardware-in-the-Loop Tests wiederverwendet werden. In der Veröffentlichung [125] ist dargestellt, wie dies erfolgen kann.

Verschiedene Fahrerassistenzfunktionen für ein Fahrerassistenzsteuergerät werden meist parallel von mehreren Entwicklern umgesetzt (siehe Kapitel 3.1.2). Da deren Zieltermine für die Abgabe der Software für die Fahrerassistenzfunktion in der Regel mit der Zeitplanung derselben Fahrzeugbaureihe zusammenhängen, benötigen sie wie in Abbildung 5.4 dargestellt meist zum selben Zeitpunkt Zugang zu einem Testsystem. Mit Hilfe des ubiquitären Testsystems auf Basis des Realzeitadapters kann für jede Fahrerassistenzfunktion ein Testsystem am Arbeitsplatz des Entwicklers bereitgestellt werden, so dass die verschiedenen Fahrerassistenzfunktionen zeitlich voneinander unabhängig und parallel getestet werden können. Nachdem die einzelnen Softwaremodule funktional am ubiquitären Testsystem getestet worden sind, kann der Verbund aus Software und Hardware auf dem Steuergerät an einem RTOS-Testsystem getestet werden. Zum Beispiel können die am ubiquitären Testsystem nicht möglichen Tests der Reaktion der Software auf elektrische Fehler durchgeführt werden.

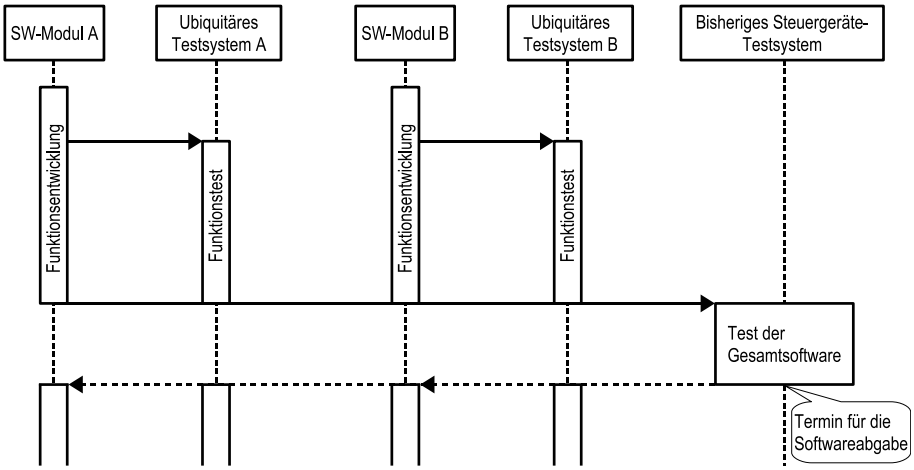


Abbildung 5.4.: Paralleler und entwicklungsbegleitender Test von Softwaremodulen an Ubiquitären Testsystemen

Zum Zeitpunkt dieser Arbeit wurden drei Testsysteme mit Arbeitsplatzrechnern für drei Fahrerassistenzfunktionen aufgebaut. Das heißt, die Entwickler der Fahrerassistenzfunktionen konnten unabhängig voneinander auf ihr eigenes Testsys-

tem zugreifen. Bisher mussten sie sich ein Testsystem teilen. Auf Grund des ubiquitären Ansatzes und der redundanten Verfügbarkeit werden diese Testsysteme oftmals auf Versuchsfahrten mitgenommen, so dass kurzfristige Änderungen zunächst am Testsystem untersucht werden können, bevor ein Fahrversuch erfolgt.

Auch für innovative Fahrerassistenzfunktionen ist es möglich, innerhalb weniger Arbeitstage ein ubiquitäres Testsystem zur Verfügung zu stellen. Insbesondere weil die Umgebungsmodelle bereits auf dem Arbeitsplatzrechner verfügbar sind, konnte bereits für die ersten Integrationsversuche der Fahrerassistenzfunktion auf der Steuergeräte-Hardware ein Testsystem bereitgestellt werden, was einen ersten Test der Fahrerassistenzfunktion ermöglichte. Der Aufbau eines neuen konventionellen Testsystems dauert in der Regel mehrere Monate, da die Verwendung von Spezial-Hardware und -Software aufwändige Planungs-, Aufbau- und Inbetriebnahmephasen erfordern.

Während für die Lösung auf Basis des Realzeitadapters wenige Arbeitstage und vierstellige Hardware-Kosten anfallen, werden für den Aufbau eines konventionellen Testsystems in der Regel Arbeitswochen bis Monate und fünfstellige Anschaffungskosten benötigt.

Der preisgünstige, ubiquitäre und zeitlich synchrone Ansatz eines Testsystems mit Hilfe des Realzeitadapters unterstützt den Test von Fahrerassistenzfunktionen auf den Integrationsebenen Steuergerät und Fahrzeug. Außerdem ermöglicht der Ansatz eine Effizienzsteigerung im Entwicklungsprozess durch die frühzeitige und parallele Durchführung von Tests der Fahrerassistenzsoftware auf dem Steuergerät, wie es für den Entwurf von eingebetteten Systemen empfohlen wird [108, 102, 41].

5.3.3. Ausblick für weitere Verbesserungen im Entwicklungsprozess

Eine wichtige, in Tabelle 5.1 aufgeführte Eigenschaft des ubiquitären Testsystems sind die deutlich geringeren Anschaffungskosten. Bei identischem Budget bedeutet dies, dass mehr Testsysteme bereitgestellt werden und diese für die Durchfüh-

Eigenschaft des Testsystems	RTOS-Testsystem	Ubiquitäres Testsystem	Verbesserung im Entwicklungsprozess
Einsatz für Fahrzeugtests	Nicht für den Fahrzeugeinsatz konzipiert	Verwendung derselben Hardware mit Synchronisation zu einer globalen Zeit	In verschiedenen Fahrzeugen aufgezeichnete Daten können zeitlich korreliert werden.
Abspielen von im Fahrzeug aufgezeichneten Daten	Mit Bezug zur lokalen Zeit möglich	Mit identischem Zeitverhalten wie bei der Aufzeichnung	Güte der Tests steigt
Standort	Labor	Arbeitsplatz, transportabel	Frühzeitiger Test von Softwareänderungen auch auf Versuchsfahrten
Anschaffungskosten	Fünfstellig	Vierstellig	Mehr Testressourcen für umfangreiche Tests der Software auf dem Steuergerät
Inbetriebnahmezeit des Testsystems	Mehrere Wochen	Mehrere Tage	Testsystem für den Test der Software auf dem Steuergerät in früher Entwicklungsphase verfügbar
Zeitverhalten beim Senden von CAN-Nachrichten	Präzise, pro Nachrichten-ID im ms-Raster definierbar	Hoch präzise, pro Nachricht im μs-Raster definierbar	Das Steuergerät kann auf die Reaktion auf ein bestimmtes Zeitverhalten der umgebenden Steuergeräte untersucht werden.
Umgang mit Steuergerätevarianten	Das Steuergerät muss manuell gewechselt werden.	Pro Variante ein Testsystem	Unterstützt eine fortlaufende Integration der Steuergerätesoftware (Softwarevarianten können automatisiert über Nacht gebaut und getestet werden).
Zukünftiger Einsatz für Fahrzeugtests	Nicht für den Fahrzeugeinsatz konzipiert	Verwendung von Teilen der Umgebungssimulation	Fahrzeugtests können ohne reale Objektfahrzeuge durchgeführt werden, was den Testaufwand senkt.

Tabelle 5.1.: Verbesserungen im Entwicklungsprozess durch ein ubiquitäres Testsystem

rung einer größeren Menge von Testfällen eingesetzt werden können. Weiterhin können für Steuergerätevarianten, die bisher an einem Testsystem geprüft wurden, jeweils separate Testsysteme angeschafft werden, so dass ein in der Regel manuell durchgeführter Wechsel der Steuergeräte am Testsystem entfällt. In Zukunft kann diese Vorgehensweise für den Aufbau einer Vielzahl von Testsystemen führen, die einen automatisierten Prozess zur kontinuierlichen Integration von eingebetteter Software ermöglichen.

Der Einsatz des Realzeitadapters sowohl an Testsystemen als auch im Fahrzeug kann zukünftig dazu benutzt werden, Teile der Umgebungssimulation auch bei Fahrzeugtests zu verwenden und dadurch dem realen Fahrzeug eine virtuelle Umgebung vorzuspielen. Statt mit hohem Aufwand reale Objektfahrzeuge auf

der Teststrecke zu bewegen, kann dann die Erlebbarkeit einer Fahrerassistenzfunktion im Fahrzeug mit geringerem Material- und Personeneinsatz untersucht werden. Werden die virtuellen Objektfahrzeuge nicht nur dem Fahrerassistenzsteuergerät vorgespielt sondern zusätzlich in das Sichtfeld des Fahrers projiziert, dann kann der Fahrer ohne große Gefährdung die Fahrerassistenzfunktion erleben.

6. Experimentelle Untersuchungen

Die Möglichkeiten des Realzeitadapters in Testsystemen für Fahrerassistenzsysteme werden in diesem Kapitel anhand der zwei Anwendungsgebiete „Steuergerätetestsystem für ein FAF-Steuergerät“ und „Auswertung von Fahrzeugdaten eines Abstandsregeltempomaten“ dargestellt.

In Kapitel 6.1 wird gezeigt, dass ein Steuergerätetestsystem auf Basis des Realzeitadapters ein präziseres Zeitverhalten auf dem CAN-Bus erreicht als bestehende Lösungen. Insbesondere wird die erreichte Genauigkeit von geforderten Zykluszeiten untersucht. Weiterhin wird gezeigt, wie mit Hilfe des Realzeitadapters auf einfache Art und Weise eine spezifizierte Reaktion des Testobjekts auf eine zeitliche Abfolge von Daten provoziert werden kann.

Das Kapitel 6.2 zeigt, die Auswertung von Daten, die mit Hilfe des Realzeitadapters und dessen Synchronisation auf die globale GPS-Zeit in verschiedenen Fahrzeugen aufgezeichnet wurden. Es wird dargestellt, wie anhand der zeitlich korrelierten Daten eine unabhängige Aussage über charakteristische Eigenschaften eines Sensorsteuergeräts getroffen werden kann und die Regelqualität eines Abstandsregeltempomaten betrachtet werden kann.

6.1. Steuergerätetestsystem für ein FAF-Steuergerät

In Kapitel 6.1.1 wird gezeigt, dass ein ubiquitäres Testsystem ein präziseres Zeitverhalten auf dem CAN-Bus erreicht als ein PC-Testsystem und als ein RTOS-Testsystem. Weitere Auswertungen sind in [126, 127] zu finden. In Kapitel 6.1.2 wird dargestellt, wie mit Hilfe des ubiquitären Testsystems auf einfache Art und

Weise eine spezifizierte Reaktion des Testobjekts auf eine zeitliche Abfolge von Daten provoziert werden kann.

6.1.1. Zeitverhalten im Vergleich zu existierenden Lösungen

Zum Vergleich der drei Testsysteme RTOS-Testsystem, PC-Testsystem und ubiquitäres Testsystem wird der in Abbildung 6.1 dargestellte Versuchsaufbau verwendet. An jedes Testsystem wird ein Fahrerassistenzsteuergerät angeschlossen. Dabei werden die CAN-Busse des Fahrerassistenzsteuergeräts mit denen des Testsystems verbunden. Es ist Aufgabe der auf dem Testsystem laufenden Simulation, die Nachrichteninhalte bereitzustellen, die über die CAN-Verbindungen übertragen werden. In Abbildung 6.1 ist exemplarisch eine CAN-Verbindung dargestellt. Mit einer zusätzlichen CAN-Schnittstelle – in diesem Versuchsaufbau ein CANcaseXL [129] der Firma Vector Informatik – und einem PC als unabhängigen Datenlogger werden die über die CAN-Verbindung übertragenen Nachrichten und deren Übertragungszeitpunkt aufgezeichnet.

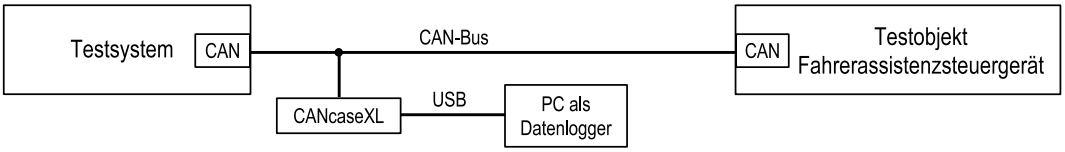


Abbildung 6.1.: Versuchsaufbau zur Messung der erreichten Genauigkeit bei der CAN-Kommunikation

Bei der Versuchsdurchführung wird an jedem Testsystem ein Fahrmanöver für einen Notbremsassistenten durchgeführt. Dabei wird das Fahrerassistenzsteuergerät, das die Software für den Notbremsassistenten enthält, mit dem folgenden Szenario stimuliert. Zunächst wird der Motor des Systemfahrzeugs gestartet und ein Objektfahrzeug 50 Meter vor dem Systemfahrzeug positioniert. Das Objektfahrzeug entfernt sich mit 15km/h vom Systemfahrzeug. Im nächsten Schritt beschleunigt der simulierte Fahrer des Systemfahrzeugs auf 65km/h. Sobald das Fahrerassistenzsteuergerät an Hand der Daten von den simulierten Sensorsteuergeräten erkennt, dass eine Kollision mit dem Objektfahrzeug droht, schickt es eine Nachricht mit einer Warnmeldung über den CAN-Bus. Ein entsprechendes Warnsignal soll dann von einem Aktorsteuergerät dem Fahrer übermittelt werden. Im vorliegenden Testfall reagiert der Fahrer nicht auf die Warnung. In die-

sem Fall muss der Notbremsassistent kurz vor vom Aufprall auf das Objektfahrzeug eine autonome Notbremsung durchführen, um die Unfallschwere zu vermindern [10].

Bei der Durchführung dieses Testfalls an allen drei Testsystemen zusammen mit dem Fahrerassistenzsteuergerät ist jeweils ein korrekter Eingriff der Fahrerassistenzfunktion erfolgt. Im Folgenden soll das während des Testfalls erreichte Zeitverhalten der verschiedenen Testsysteme auf den jeweiligen CAN-Bussen analysiert werden. Der untersuchte CAN-Bus ist zu ungefähr 50% belastet. Anhand der CAN-Nachricht mit der ID 3, die laut Spezifikation alle $10ms$ vom Testsystem an das Fahrerassistenzsteuergerät geschickt werden soll und die in diesem Versuch die höchste Priorität hat, wird das erreichte Zeitverhalten dargestellt.

Für jedes Testsystem werden zwei Diagramme aufgeführt. Zum einen wird der Abstand zwischen zwei aufeinanderfolgenden Nachrichten mit der ID 3 für jede Nachricht im Testfall auf der Y-Achse abgetragen. Da in der Regel, wenn eine Nachricht verspätet gesendet wird, die darauffolgende relativ gesehen einen zu kurzen zeitlichen Abstand zur vorhergehenden hat, wirken diese Grafiken achsensymmetrisch. Zum anderen werden die erzielten Abstände zwischen zwei Nachrichten in einem Histogramm dargestellt. Auch hier ist die Symmetrie erkennbar. Als Orientierung ist im Histogrammen eine auf Basis von Mittelwert und Standardabweichung errechnete Gauss-Verteilung eingezeichnet. Die folgenden 6 Abbildungen sind identisch skaliert, so dass die Ergebnisse nebeneinander gelegt werden können.

Abbildung 6.2 und Abbildung 6.3 zeigen das Ergebnis des RTOS-Testsystems[1]. Die 1600 CAN-Nachrichten werden mit einem durchschnittlichen Abstand von $9,999ms$ und einer Standardabweichung von $0,379ms$ vom Prüfstand verschickt. Der minimal aufgetretene Abstand beträgt $7,069ms$ und der maximale $12,870ms$. Es ist zu erkennen, dass periodisch einzelne Ausreißer auftreten. Zu diesen Zeitpunkten müssen mehr Nachrichten verschickt werden, da mehrere Nachrichten mit verschiedenen Zykluszeiten zum selben Zeitpunkt fällig sind.

In Abbildung 6.4 ist erkennbar, dass bei einem PC-Testsystem einzelne gravierende Ausreißer auftreten. Bei den 1563 verschickten Nachrichten liegt das Maximum bei $22,921ms$ und das Minimum bei $6,496ms$. Im Mittel erreicht das PC-

[1] Der Testfall wurde auf einem LabCar Testsystem der Firma ETAS durchgeführt [141].

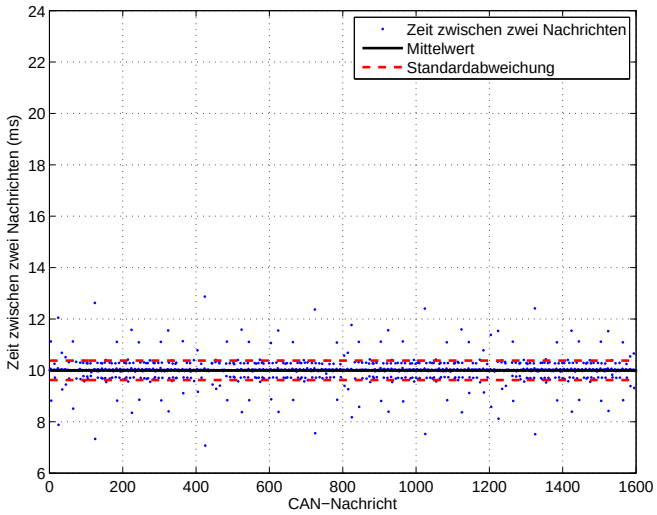


Abbildung 6.2.: Zeit zwischen zwei Nachrichten der ID 3 an einem RTOS-Testsystem

Testsystem $10,048ms$ bei einer Standardabweichung von $0,887ms$. Letztere entspricht ungefähr dem doppelten Wert des RTOS-Testsystems.

Die großen Ausreißer treten sporadisch auf und unterbrechen die Symmetrie zum Mittelwert, wie im Histogramm Abbildung 6.5 erkennbar ist. Da das PC-Testsystem auf Basis von Windows [76] arbeitet, hat vermutlich hier die Simulation erst zu spät die nötige Rechenzeit vom Betriebssystem erhalten.

Abbildung 6.6 zeigt, dass bei identischer Skalierung der Grafik keine Ausreißer beim ubiquitären Testsystem erkennbar sind. Im Histogramm in Abbildung 6.7 sind alle Werte innerhalb eines einzigen Containers. Daher werden in den Abbildungen 6.8 und 6.9 die selben Werte mit einer angepassten Skalierung dargestellt.

Die in Abbildung 6.8 dargestellten 1568 Nachrichten werden vom ubiquitäres Testsystem mit einem mittleren Abstand von $10,003ms$ und mit einer Standardabweichung von $0,004ms$ verschickt. Der minimale zeitliche Abstand zwischen zwei Nachrichten beträgt $9,986ms$ und der maximale $10,036ms$. Die Nachrichten werden somit mit einer Abweichung von wenigen μs vom geforderten Sende-

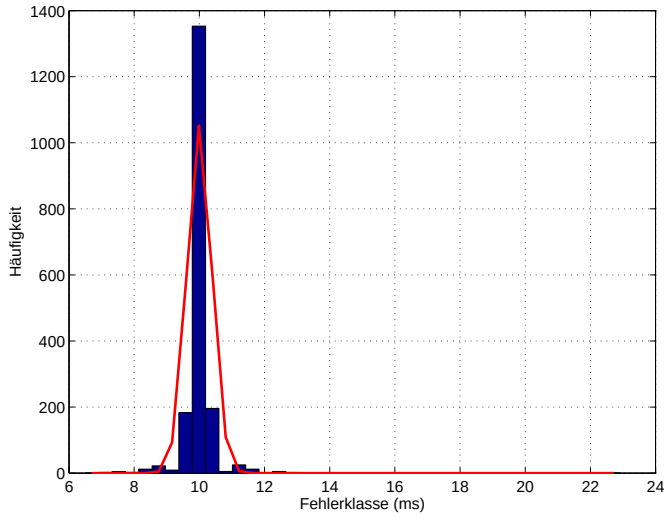


Abbildung 6.3.: Histogramm der zeitlichen Differenz zwischen zwei Nachrichten der ID 3 an einem RTOS-Testsystem

zeitpunkt verschickt.

In Tabelle 6.1 werden die statistischen Werte der untersuchten Testsysteme zusammengefasst. Das Ziel, ein einem RTOS-Testsystem ähnliches Zeitverhalten zu erreichen, während die Vorteile des PC-Testsystems erhalten bleiben, ist durch das ubiquitäre Testsystem erreicht. Insbesondere die um Faktor zehn bessere Standardabweichung zeigt die Präzision des vorgestellten Konzeptes. Zusätzlich ist erkennbar, dass die Amplitude der Ausreißer deutlich geringer ist als die des RTOS-Testsystems. Man kann argumentieren, dass ein solch präzises Zeitverhalten auch mit einem RTOS-Testsystem erreichbar ist. Dies hängt sicherlich auch davon ab, wie die Prüfstandssoftware konfiguriert wird. Umgekehrt kann auch das Zeitverhalten eines ubiquitären Testsystems ungeschickt konfiguriert werden. Allerdings wird dies durch die eingebaute Rückmeldung bei Verletzungen der Soll-Sendezeit zuverlässig gemeldet.

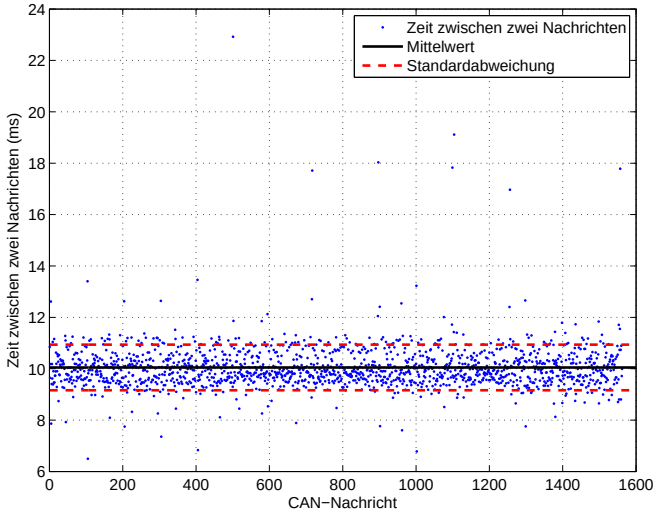


Abbildung 6.4.: Zeit zwischen zwei Nachrichten der ID 3 an einem PC-Testsystem

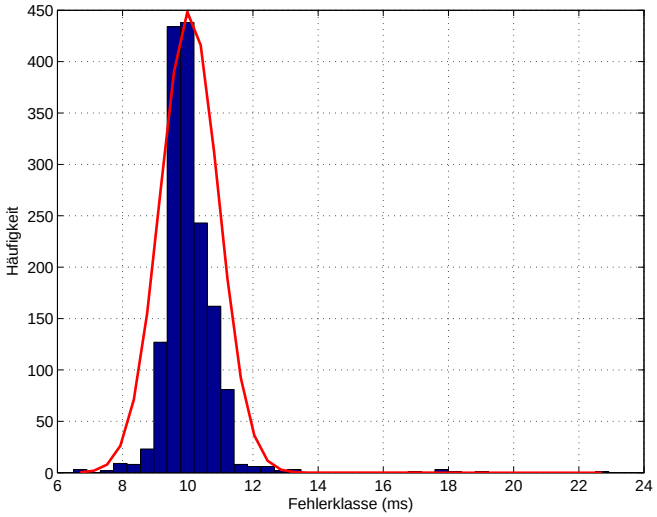


Abbildung 6.5.: Histogramm der zeitlichen Differenz zwischen zwei Nachrichten der ID 3 an einem PC-Testsystem

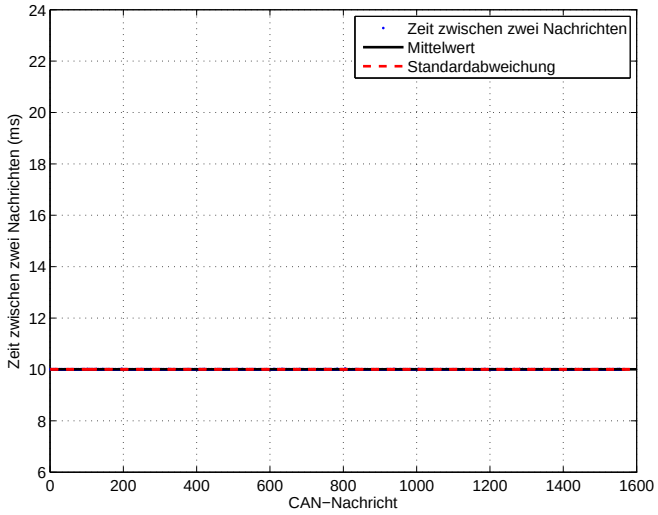


Abbildung 6.6.: Zeit zwischen zwei Nachrichten der ID 3 an einem ubiquitären Testsystem

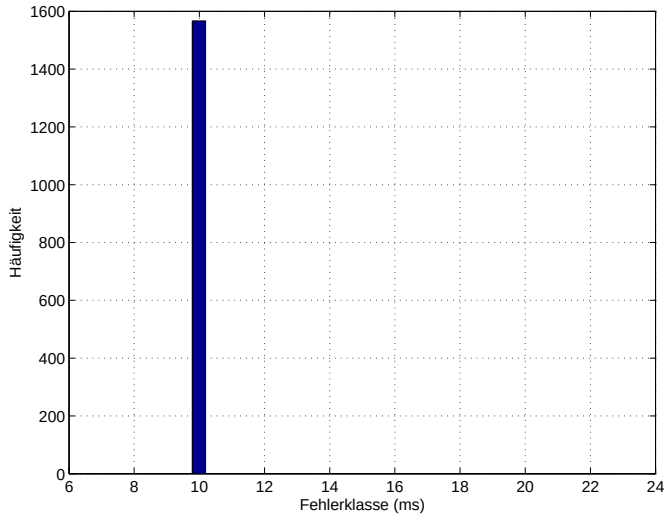


Abbildung 6.7.: Histogramm der zeitlichen Differenz zwischen zwei Nachrichten der ID 3 an einem ubiquitären Testsystem

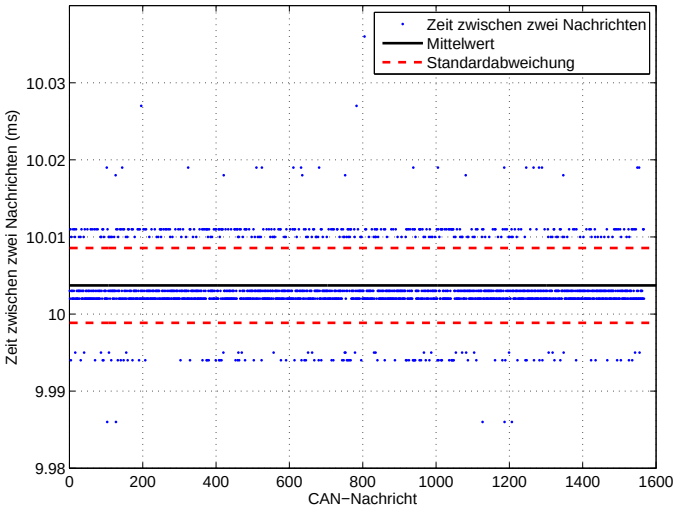


Abbildung 6.8.: Zeit zwischen zwei Nachrichten der ID 3 an einem ubiquitären Testsystem (vergrößert)

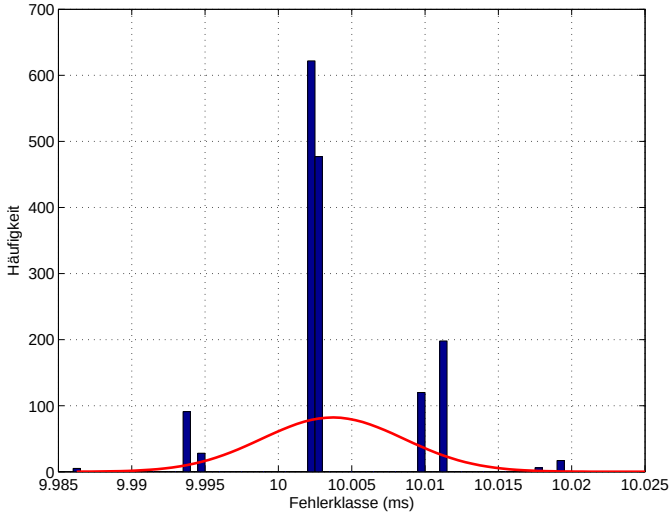


Abbildung 6.9.: Histogramm der zeitlichen Differenz zwischen zwei Nachrichten der ID 3 an einem ubiquitären Testsystem (vergrößert)

Eigenschaft	RTOS-Testsystem	PC-Testsystem	Ubiquitäres Testsystem
Minimaler zeitlicher Abstand zwischen zwei Nachrichten	$7,069ms$	$6,496ms$	$9,986ms$
Maximaler zeitlicher Abstand zwischen zwei Nachrichten	$12,870ms$	$22,921ms$	$10,036ms$
Mittlerer zeitlicher Abstand zwischen zwei Nachrichten	$9,999ms$	$10,048ms$	$10,003ms$
Standardabweichung des zeitlichen Abstands zwischen zwei Nachrichten	$0,379ms$	$0,887ms$	$0,004ms$
Anzahl der Nachrichten	1600	1563	1568

Tabelle 6.1.: Vergleich von statistischen Daten der drei Testsysteme

6.1.2. Test der Sensorplausibilisierung

In Kapitel 2.2 wird eine mögliche Steuergerätearchitektur für Fahrerassistenzsysteme beschrieben. In Abbildung 2.3 ist dargestellt, dass diese verschiedene Sensorsteuergeräte beinhaltet, auf Basis derer Informationen die Fahrerassistenzfunktionen im Fahrerassistenzsteuergerät arbeiten. Bevor eine Fahrerassistenzfunktion, wie zum Beispiel ein Totwinkelassistent [5], auf die Sensorinformationen mit einem Eingriff in das Fahrverhalten des Fahrzeugs reagiert, muss sichergestellt sein, dass die Informationen der Sensorsteuergeräte nachvollziehbar sind. Hierzu können die Informationen verschiedener Sensorsteuergeräte über dasselbe Objekt in der Fahrzeugumgebung in der Sensorfusion verglichen werden [34, 58]. Im Vorfeld ist es sinnvoll, zunächst die Nachvollziehbarkeit der Informationen eines einzelnen Sensorsteuergeräts zu prüfen. Dieser Schritt soll im weiteren Verlauf *Sensorplausibilisierung*[2] genannt werden.

Eine Möglichkeit ist es, die Informationen über die Zeit zu beobachten. Beispielsweise können die Informationen auf Verläufe geprüft werden, die in der realen Welt nicht vorkommen können und somit auf einen Fehler im Sensorsteuergerät hinweisen. Zum Beispiel ist es unplausibel, wenn das Sensorsteuergerät zunächst meldet, dass ein Objekt in der Umgebung erkannt wird, danach, dass keines erkannt wird, dann wieder, dass eines erkannt wird und so weiter. Um dieses Verhalten in einem Echtzeitkontext erkennen zu können, müssen den einzelnen Zuständen Zeiten zugeordnet werden. Als Beispiel soll definiert werden, dass der Zustand „Objekt erkannt“ jeweils für $t_{Obj} = 40ms$ anliegen soll und der Zustand „Objekt nicht erkannt“ für $t_{noObj} = 60ms$. Daraus ergibt sich das in Abbildung 6.10 dargestellte Muster für ein ungültiges Verhalten des Sensorsteuergeräts.

Werden die Informationen zwischen Sensorsteuergerät und Fahrerassistenzsteuergerät zyklisch alle $20ms$ übertragen, so bedeutet dies, dass zwei Nachrichten mit dem Inhalt „Objekt erkannt“ und drei Nachrichten mit dem Inhalt „Objekt nicht erkannt“ jeweils im Wechsel vom Sensorsteuergerät verschickt werden.

Die korrekte Funktion dieser Überwachung soll nicht nur in Softwaretests nachgewiesen werden, sondern auch auf dem Steuergerät. Hierzu soll das Steuer-

[2] Bei der Sensorplausibilisierung wird geprüft, ob die Informationen vom Sensorsteuergerät plausibel sind.

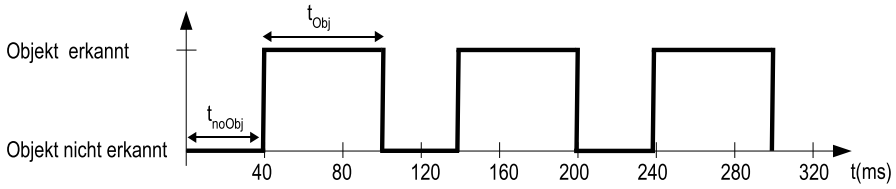


Abbildung 6.10.: Unplausibles Verhaltensmuster eines Sensorsteuergeräts

gerät an einem Testsystem untersucht werden. Das heißt, in diesem Versuch ist das Testobjekt das Fahrerassistenzsteuergerät, insbesondere das darin enthaltene Softwaremodul Sensorplausibilisierung.

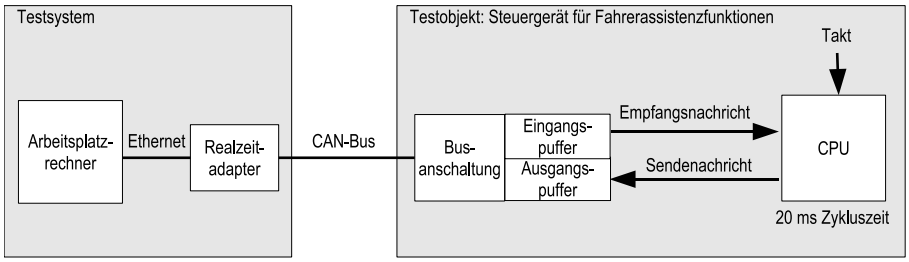


Abbildung 6.11.: Versuchsaufbau zum Test der Sensorplausibilisierung

In Abbildung 6.11 ist der Versuchsaufbau zum Test eines Fahrerassistenzsteuergeräts an einem Testsystem auf Basis des Realzeitadapters dargestellt. Zum Verständnis, warum sich dieses Testsystem hier besonders eignet, ist der innere Aufbau des Fahrerassistenzsteuergeräts skizziert. Das Steuergerät besitzt eine Busanschaltung an den CAN-Bus und einen Mikrokontroller (CPU), auf dem die Fahrerassistenzsoftware inklusive des Softwaremoduls zur Sensorplausibilisierung läuft. Die Fahrerassistenzsoftware wird zyklisch, alle 20*ms* ausgeführt. Ein solcher Zyklus beinhaltet zunächst das Abholen aller Empfangsnachrichten aus dem Eingangspuffer der Busanschaltung. Hier werden die kontinuierlich eintreffenden Nachrichten vom CAN-Bus zwischengespeichert, bis sie von der Fahrerassistenzsoftware abgeholt werden. Im nächsten Schritt werden auf Basis der empfangenen Nachrichten die Fahrerassistenzfunktionen berechnet, wozu auch die Ausführung des Softwaremoduls zur Sensorplausibilisierung gehört. Am Ende des Zyklus werden errechnete Ergebnisse als Sendenachrichten an den Ausgangspuffer übergeben und dann über die Busanschaltung verschickt.

Der zeitliche Verlauf dieser Verarbeitungskette basiert auf dem im Steuergerät generierten Takt. Dies bedeutet, dass die Dauer eines Zyklus zwar mit 20*ms* spezifiziert ist, die tatsächliche Dauer bezogen auf eine globale Zeit aber auf Grund von Toleranzen des Taktgenerators abweicht (siehe Kapitel 3.2). Dies bedeutet, dass das Verhaltensmuster über den CAN-Bus mit zeitlichem Bezug zum Takt des Steuergeräts angelegt werden muss. Das Testsystem muss sich somit auf den Takt des Steuergeräts synchronisieren. Weil dies mit dem Realzeitadapter besonders einfach möglich ist, konnte dieser Test mit geringem Aufwand umgesetzt werden.

Da das Steuergerät seine Sendenachricht ebenfalls mit seinem internen Takt verschickt, kann mit Hilfe der vom Realzeitadapter bereitgestellten Empfangszeitstempel dieser Nachricht der Sendezeitpunkt für die Nachricht vom Testsystem ermittelt werden. Im Listing 6.1 ist dargestellt, wie das geforderte Verhaltensmuster zum Steuergerätetakt synchron verschickt werden kann. Es wird angenommen, dass das Steuergerät nur eine CAN-Nachricht verschickt. Das Testsystem wartet zunächst auf diese Nachricht und merkt sich deren Eingangszeitstempel. Relativ zu diesem Zeitstempel wird dann die Nachricht mit der Information über das Objekt an das Steuergerät verschickt. Im gezeigten Fall mit einem Offset von 5*ms* zum Empfangszeitpunkt. Der Inhalt der Antwort enthält immer im Wechsel zwei Mal den Wert „NO_OBJ“ (Objekt nicht erkannt), danach drei Mal den Wert „OBJ“ (Objekt erkannt).

```
enum ObjState
{
  NO_OBJ,  //Objekt nicht erkannt
  OBJ      //Objekt erkannt
};
int i = 0;
while(1){
    // Auf Triggernachricht vom Steuergerät warten
    RTA_receive(&NachrichtRX, &Zeitstempel);

    // Das vorgegebene Verhaltensmuster verschicken
    if(i < 2){
      // "Kein Objekt erkannt" versenden
      RTA_send(NO_OBJ, Zeitstempel + 5); }
    else if(iObj < 5){
      // "Objekt erkannt" versenden
      RTA_send(OBJ, Zeitstempel + 5); }
    i++;
    if (iObj >= 5) i = 0;
}
```

Listing 6.1: Zum Takt des Steuergerätes synchrones Versenden von CAN"

Prinzipiell ist es möglich, das in Listing 6.1 dargestellte Verfahren an einem RTOS-Testsystem umzusetzen. In der Regel ist die zyklische CAN-Kommunikation an einem RTOS-Testsystem dadurch realisiert, dass das Betriebssystem Funktionen für die unterschiedlichen Zykluszeiten der CAN-Nachrichten aufruft. In dieser Funktion werden dann alle CAN-Nachrichten verschickt, die in der für die Funktion festgelegten Zykluszeit verschickt werden sollen. Beispielsweise kann im Realzeitbetriebssystem festgelegt werden, dass eine Funktion alle 10*ms* aufgerufen wird. In dieser Funktion werden alle CAN-Nachrichten mit einer spezifizierten Zykluszeit von 10*ms* verschickt. Der zyklische Aufruf dieser Funktion erfolgt aber leider auf Basis der Zeitbasis des Testsystems und daher nicht notwendig synchron zum Takt des Testobjekts.

Neben den zyklisch aufgerufenen Funktionen stellt das Realzeitbetriebssystem Funktionen bereit, die bei einem Ereignis aufgerufen werden. Das heißt, es ist prinzipiell möglich, die zyklische Implementierung der Restbussimulation auf einen reaktiven Ansatz umzustellen und diesen mit Hilfe von Ereignissen auf den Takt des Testobjekts zu synchronisieren. Allerdings bedeutet dies eine Umstrukturierung der Restbussimulation. Diese Umstrukturierung muss am Testsystem durchgeführt und verifiziert werden. Während dieser Umbauphase steht das Testsystem, das als einzelne Instanz existiert, nicht für den Test von Fahrerassistenzfunktionen zur Verfügung. Da das ubiquitäre Testsystem mehrfach vorhanden ist, hat hier die Umbauphase keine Auswirkungen auf den Testbetrieb. Aus diesem Grund und weil der Aufwand für die Umsetzung für geringer eingeschätzt wurde, wurde im Industrieprojekt entschieden, die geforderten Tests am ubiquitären Testsystem umzusetzen.

6.2. Auswertung von Fahrzeugdaten eines Abstandsregeltempomaten

Im Folgenden sollen die Ergebnisse der Untersuchung eines Abstandsregeltempomaten im Fahrzeug dargestellt werden. Der Versuchsaufbau erfolgte wie in Abbildung 5.1 in Kapitel 5.1 dargestellt. Dabei werden mit Hilfe des Realzeitadapters zwei Datenlogger aufgebaut. Einer zeichnet die Geschwindigkeit des Objektfahrzeugs auf, während der andere Realzeitadapter im Systemfahrzeug die

Geschwindigkeit und die vom Fernbereichsradar gelieferte Relativgeschwindigkeit zum vorausfahrenden Objektfahrzeug aufzeichnet. Folgende vier Versuche wurden durchgeführt und sollen im weiteren Verlauf erläutert werden:

1. Anhaltevorgang auf ein zum Stillstand bremsendes Objektfahrzeug durchgeführt von einem defensiven Fahrer im Systemfahrzeug.

2. Anhaltevorgang auf ein zum Stillstand bremsendes Objektfahrzeug durchgeführt von einem aggressiven Fahrer im Systemfahrzeug.

3. Anhaltevorgang auf ein zum Stillstand bremsendes Objektfahrzeug durchgeführt von einem Abstandsregeltempomaten im Systemfahrzeug mit maximaler Abstandsvorgabe.

4. Anhaltevorgang auf ein zum Stillstand bremsendes Objektfahrzeug durchgeführt von einem Abstandsregeltempomaten im Systemfahrzeug mit minimaler Abstandsvorgabe.

Die von den unabhängigen, über GPS synchronisierten Datenlogger aufgezeichneten Geschwindigkeitsinformationen werden nach der Versuchsdurchführung auf eine gemeinsame Zeitachse zusammengeführt und daraufhin visualisiert. Abbildung 6.12 zeigt den Geschwindigkeitsverlauf des Systemfahrzeugs in Abhängigkeit zu dem des Objektfahrzeugs, sowie deren errechnete Differenz und die vom Fernbereichsradar ermittelte Differenzgeschwindigkeit. Es ist zu erkennen, dass der defensive Fahrer des Systemfahrzeugs schon sehr früh auf die Verzögerung des Objektfahrzeugs reagiert, deutlich stärker verzögert als das vorausfahrende Fahrzeug und daraufhin sogar vor dem Objektfahrzeug zum Stillstand kommt.

Abbildung 6.13 zeigt den Anhaltevorgang, wenn er von einem aggressiveren Fahrer durchgeführt wird. Im Fahrversuch kam das Systemfahrzeug erst knapp hinter dem Objektfahrzeug und beinahe gleichzeitig mit Objektfahrzeug zum Stillstand. Dies ist in der Grafik daran zu erkennen, dass die Geschwindigkeitskurven beider Fahrzeuge beinahe zeitgleich die $0km/h$ Linie erreichen.

Zum Zeitpunkt $t = 55s$ ist erkennbar, dass vom Fernbereichsradar keine Informationen zur Differenzgeschwindigkeit geliefert werden. Dies bedeutet, dass der Fernbereichsradar in dieser Situation das Objektfahrzeug nicht erkennen konnte.

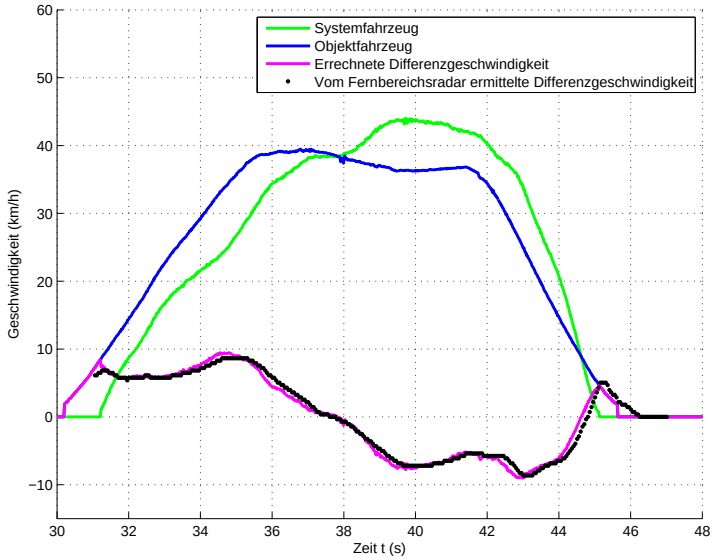


Abbildung 6.12.: Von einem defensiven Fahrer durchgeführter Anhaltevorgang

Sobald dies wieder möglich war, wurden wieder korrekte Informationen geliefert, was an der guten Übereinstimmung der ermittelten Werte zu der errechneten Kurve erkennbar ist.

Bei den meisten Abstandsregeltempomaten kann der vom Fahrer gewünschte Abstand zum vorausfahrenden Fahrzeug eingestellt werden. Im Rahmen der Versuchsreihe wurde jeweils ein Versuch mit minimal und maximal möglicher Abstandsvorgabe durchgeführt. Abbildung 6.14 zeigt einen vom Abstandsregeltempomaten durchgeführten Anhaltevorgang mit maximaler Abstandsvorgabe. Aufgabe der Assistenzfunktion ist es, abhängig vom vorausfahrenden Fahrzeug das Systemfahrzeug bis zum Stillstand abzubremsen. Es ist zu sehen, dass das Systemfahrzeug mit einem zeitlichen Verzug von $1,7s$ auf die Verzögerung des Objektfahrzeugs reagiert. Der beinahe parallele Verlauf der Geschwindigkeitskurven zeigt, dass der Abstandsregeltempomat sich genau an die Verzögerungsvorgabe des Objektfahrzeugs hält. Im Gegensatz hierzu hat der defensive Fahrer in Abbildung 6.12 stärker verzögert als das Objektfahrzeug, was an den sich annähernden Geschwindigkeitskurven erkennbar ist.

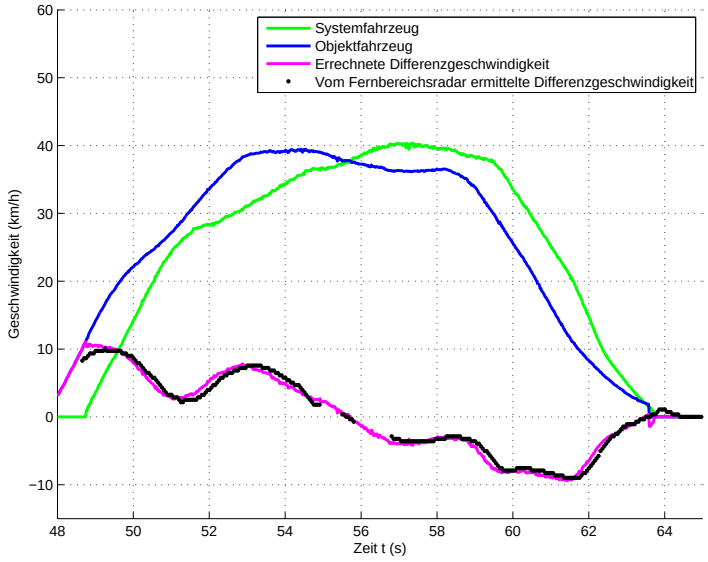


Abbildung 6.13.: Von einem aggresiven Fahrer durchgeführter Anhaltevorgang

Abbildung 6.15 zeigt den Anhaltevorgang mit einem Abstandsregeltempomaten mit minimaler Abstandsvorgabe. Es ist zu erkennen, dass die beiden Geschwindigkeitskurven mit einem höheren zeitlichen Abstand von $2,9s$ parallel zueinander verlaufen. Mit entsprechend höherem zeitlichen Verzug zum Objektfahrzeug kommt das Systemfahrzeug zum Stillstand. Kurz bevor das Systemfahrzeug zum Stillstand kommt, kann zum Zeitpunkt $t = 58s$ am Verlauf der vom Fernbereichsradar ermittelten Differenzgeschwindigkeit erkannt werden, dass in dieser Situation die Informationen vom Fernbereichsradar ungenau werden.

Nach [138] wird zur Einhaltung der Regelqualität von einem Abstandsregeltempomaten gefordert, dass die Informationen des Fernbereichsradar mit einer maximale Verzögerung von $0,25s$ bereitgestellt werden. Abbildung 6.16 zeigt eine Vergrößerung aus Abbildung 6.15 in der der Verzug der Informationen des Fernbereichsradars zu der errechneten Differenzgeschwindigkeit erkennbar ist. Der im Fahrversuch ermittelte Wert von circa $200ms$ konnte von den für den Fernbereichsradar verantwortlichen Personen bestätigt werden.

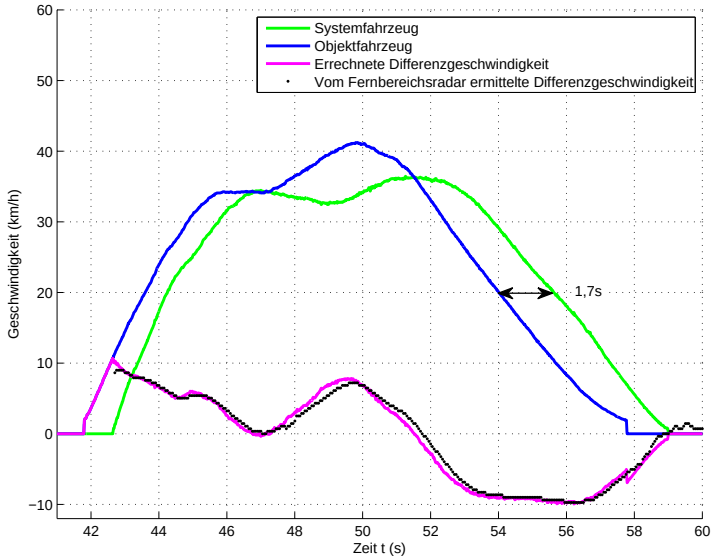


Abbildung 6.14.: Anhaltevorgang mit einem Abstandsregeltempomaten mit maximaler Abstandsvorgabe

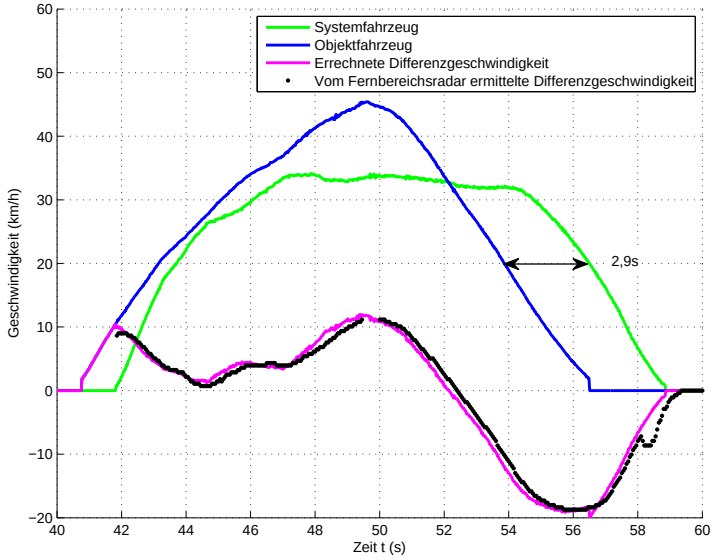


Abbildung 6.15.: Anhaltevorgang mit einem Abstandsregeltempomaten mit minimaler Abstandsvorgabe

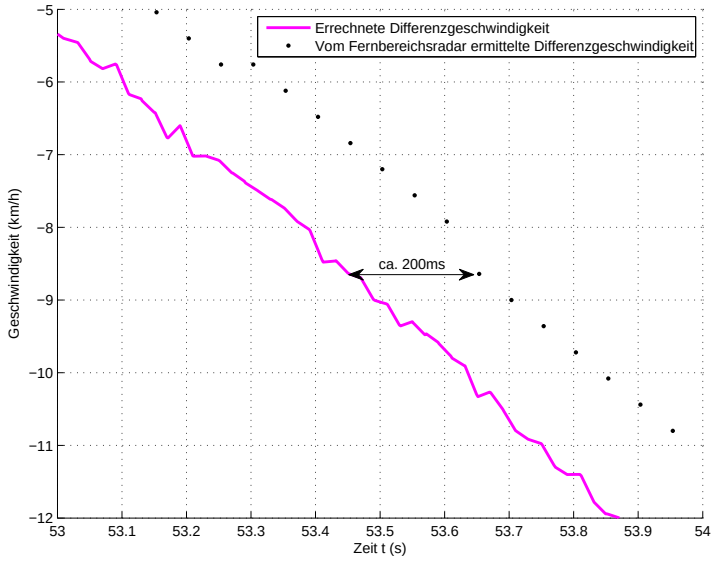


Abbildung 6.16.: Vergleich der vom Radar-Steuergerät gelieferten Relativgeschwindigkeit mit der auf Basis von Busdaten berechneten Relativgeschwindigkeit

6.3. Bewertung

Mit den experimentellen Untersuchungen in Kapitel 6.1 konnte gezeigt werden, dass ein Testsystem, bestehend aus einem Realzeitadapter und einem nicht echtzeitfähigen Arbeitsplatzrechner, existierende Testsysteme für Fahrerassistenzsteuergeräte in der Präzision des Zeitverhaltens auf den CAN-Bussen übertrifft.

Bei Testfällen, die ein beliebig spezifiziertes aber dennoch präzises und reproduzierbares Zeitverhalten auf dem CAN-Bus fordern (gezeigt in Kapitel 6.1.2), sorgt der Ansatz des zeitgesteuerten Sendens für eine einfache und schnelle Umsetzung des geforderten Verhaltens am Testsystem. Mit den existierenden Lösungen ist eine Umsetzung der Zeitanforderungen nicht in einem zeitlich sinnvollen Rahmen möglich. Teilweise unterstützen die vom Hersteller des Testsystems mitgelieferten Konfigurationsprogramme weder den in Kapitel 6.1.2 gezeigten Anwendungsfall noch die geforderte zeitliche Präzision und Flexibilität.

Werden die präzisen Zeitstempel im Realzeitadapter synchron zu einer globalen Zeit an eingehende CAN-Nachrichten angehängt, so kann mit Hilfe des Realzeitadapters und einem Arbeitsplatzrechner in Form eines Laptops ein hochpräziser Datenlogger für die Untersuchung von Fahrerassistenzfunktionen im Fahrzeug aufgebaut werden. Kapitel 6.2 zeigt, wie es die GPS-synchrone Aufzeichnung der CAN-Nachrichten erlaubt, die Informationen aus verschiedenen Fahrzeugen auf eine Zeitachse zu legen. Mit diesem Ansatz ist es möglich, die Reaktion des Systemfahrzeugs auf eine Aktion eines Objektfahrzeugs quantitativ zu bestimmen. Sowohl Latenzzeiten in der Sensorik als auch die Qualität eines Reglers oder einer Assistenzfunktion lassen sich über diesen Ansatz quantitativ bestimmen.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Hardware-in-the-Loop Tests sind eine weit verbreitete Technik, um die Funktionalität von Steuergeräten zu untersuchen. Mit der steigenden Komplexität von Fahrerassistenzfunktionen werden sie zunehmend auch entwicklungsbegleitend eingesetzt. Ziel ist es, frühzeitig Fehler zu finden. Auf Grund der steigenden Anzahl und Komplexität von Fahrerassistenzfunktionen steigt die Nachfrage nach Testsystemen zur entwicklungsbegleitenden Untersuchung von Entwicklungsergebnissen.

Ein weiterer Aspekt von Fahrerassistenzfunktionen ist, dass sie oftmals auf Objektfahrzeuge im Umfeld des Systemfahrzeugs reagieren. Dies bedeutet in der Regel, dass für die Auswertung eines Tests auf der Integrationsstufe Fahrzeug einer solchen Fahrerassistenzfunktion Informationen von den Objektfahrzeugen mit einbezogen werden müssen. Es existieren Ansätze, die Daten mehrerer Fahrzeuge zusammen zu führen. Diese sind, wie in Kapitel 3.2.2 beschrieben ist, aufwändig, in ihrer Präzision eingeschränkt und basieren größten Teils auf einer anderen Hard- und Software als derzeitig existierende Testsysteme.

Aus diesen Gründen war es Ziel dieser Arbeit, ein zeitlich präzises Testsystem für Steuergeräte zu entwickeln, das sowohl entwicklungsbegleitend am Arbeitsplatz als auch für den Test von Fahrerassistenzfunktionen im Fahrzeug eingesetzt werden kann und eine nahtlose Verwendung der im Fahrzeug aufgezeichneten Daten an einem Testsystem erlaubt.

Dazu wurde im ersten Teil der Arbeit ein Konzept entwickelt, das mit Hilfe von Zeitstempeln ein echtzeitfähiges Testsystem auf Basis eines nicht echtzeitfähi-

gen Arbeitsplatzrechners ermöglicht. Dabei wird die Rechenleistung des Arbeitsplatzrechners dazu verwendet, die für das Testsystem nötigen Simulationen für Fahrer, Fahrzeug und Fahrzeugumgebung sowie die Restbussimulation zu rechnen und die Ergebnisse in Form von CAN-Nachrichten mit einem Soll-Sendezeitpunkt an den Realzeitadapter zu übergeben. Aufgabe des Realzeitadapters ist es, die CAN-Nachrichten zum Soll-Sendezeitpunkt an das zu testende Steuergerät zu schicken. Sollte die Simulation auf dem Arbeitsplatzrechner die Nachrichten nicht rechtzeitig an den Realzeitadapter schicken, erkennt dieser den Fehler, weil der Soll-Sendezeitpunkt in der Vergangenheit liegt und kann den Fehler zurückmelden. Weiterhin empfängt der Realzeitadapter alle Nachrichten vom Steuergerät, versieht diese mit einem Eingangszeitstempel und leitet sie an den Arbeitsplatzrechner weiter, wo sie als Eingangsgrößen für die Simulationen dienen.

Wird die Uhr des Realzeitadapters auf eine globale Zeit synchronisiert, so kann zusätzlich erreicht werden, dass der Realzeitadapter für die Bewertung von Fahrerassistenzfunktionen im Fahrzeug geeignet ist. Es ist somit möglich, aufgezeichnete Nachrichten aus verschiedenen Fahrzeugen mit Hilfe der zur globalen Zeit synchronen Zeitstempel zeitlich zu korrelieren. Als weiterer Vorteil sorgt die Synchronität zur globalen Zeit dafür, dass das Zeitverhalten des Testsystems bei einer wiederholten Durchführung von Testfällen nahezu identisch ist.

Auf Basis dieses Konzeptes wurde die Architektur des Realzeitadapters entworfen. Die wichtigsten Bestandteile sind:

- Eine Zeitstempeleinheit, die eingehende Nachrichten mit einem präzisen Zeitstempel bezogen auf die Uhr des Realzeitadapters versieht.

- Eine zeitgesteuerte Sendeeinheit, die bezogen auf die Uhr des Realzeitadapters CAN-Nachrichten zu einem Soll-Sendezeitpunkt verschicken kann.

- Eine Sendeverzugseinheit, die den Soll-Sendezeitpunkt mit der tatsächlichen Sendezeit vergleicht und bei einer Abweichung außerhalb einer vorgegebenen Toleranz eine Fehlermeldung generiert.

- Eine Uhrregelung, die die Uhr des Realzeitadapters auf die GPS-Zeit als globale Zeit synchronisiert.

Da ein Testsystem auf Basis des Realzeitadapters flexibel und transportabel ein-

gesetzt werden kann, wird für ein solches Testsystem der Begriff ubiquitäres Testsystem eingeführt.

Es wird gezeigt, dass das ubiquitäre Testsystem die Vorteile der bestehenden Testsysteme RTOS-Testsystem und PC-Testsystem vereint und um zusätzliche positive Eigenschaften ergänzt.

Das ubiquitäre Testsystem zeichnet sich durch geringe Hardware-Kosten aus, so dass jedem Entwickler ein persönliches Testsystem bereitgestellt werden kann. Hierdurch hat er nicht nur jederzeit Zugriff auf sein Testsystem, verglichen mit einem gemeinsamen Testsystem entfallen auch Umrüstzeiten, wenn einzelne Entwickler das Testsystem in einer Variante benötigen und jeder Entwickler kennt immer die Konfiguration seines Testsystems. Außerdem kann er sein Testsystem auf eine Erprobung mitnehmen, so dass er kurzfristige Änderungen am Testsystem untersuchen kann, bevor er mit einem Versuchsfahrzeug auf die Teststrecke fährt. Des Weiteren kann er das ubiquitäre Testsystem – in diesem Fall als Datenlogger – verwenden, um das Verhalten von Fahrerassistenzsystemen im Fahrzeug zu analysieren.

Essenziell für ein Testsystem für Fahrerassistenzfunktionen ist dessen Echtzeitfähigkeit. In den experimentellen Untersuchungen konnte gezeigt werden, dass das ubiquitäre Testsystem ein RTOS-Testsystem beim Einhalten der geforderten Zykluszeiten von CAN-Nachrichten sogar übertrifft. Außerdem ist es am ubiquitären Testsystem im Gegensatz zum RTOS-Testsystem auf einfache Art und Weise möglich, das Zeitverhalten jeder einzelnen CAN-Nachricht nahezu beliebig zu beeinflussen.

In Tabelle 7.1 werden die Kerneigenschaften des ubiquitären Testsystems zusammengefasst und gegen die in dieser Arbeit gestellten Anforderungen gespiegelt. Die vorgestellten Ansätze wurden zum Patent angemeldet und auf verschiedenen Konferenzen veröffentlicht. Im Rahmen dieser wissenschaftlichen Vorgehensweise sind wertvolle Rückmeldungen in den Bau des Prototypen eingeflossen und wurden in mehreren Iterationen umgesetzt. Ebenso dienten die Veröffentlichungen zur Verbreitung des ubiquitären Testsystems, wovon bereits drei Instanzen im Einsatz und zwei weitere in Planung sind. Die Testsysteme werden von verschiedenen Firmen und Abteilungen genutzt. Bereits mehrfach bewährt hat sich das Testsystem, wenn spezielle Zeitanforderungen an die CAN-Kommunikation innerhalb kürzester Zeit umgesetzt werden müssen, ohne den laufenden

Betrieb zu beeinflussen.

Der Entwicklungsprozess von Fahrerassistenzfunktionen konnte nachhaltig beeinflusst werden, da es für die Entwickler zwischenzeitlich einfacher ist, eine geänderte Funktion sofort am Testsystem und dann im Fahrzeug zu untersuchen. Dies ermöglicht das frühe Finden von Fehlern. Durch den ubiquitären Ansatz gilt dies sowohl am Arbeitsplatz als auch auf einer Erprobungsfahrt, zu der seither in der Regel immer ein Testsystem mitgenommen wird. Selbst für neue Fahrerassistenzfunktionen, die für Baureihen vorgesehen sind, die noch in einem sehr frühen Entwicklungsstadium sind, können zeitnah Testsysteme bereitgestellt werden.

Anforderung	Lösung
1: Echtzeitfähigkeit	Das ubiquitäre Testsystem übertrifft selbst ein RTOS-Testsystem um Faktor 10 in der Standardabweichung beim Einhalten von auf dem CAN-Bus geforderten Zykluszeiten.
2: Flexible Beeinflussung des Zeitverhaltens	Mit dem ubiquitären Testsystem ist es mit wenig Implementierungsaufwand möglich, ein nahezu beliebiges Zeitverhalten auf dem CAN-Bus zu realisieren. Dadurch ist es beispielsweise möglich, das Testsystem auf das Testobjekt zu synchronisieren.
3: Preisgünstig	Die Hardware-Kosten für ein ubiquitäres Testsystem liegen zum Zeitpunkt der Arbeit bei circa 500 Euro für einen Arbeitsplatzrechner plus 5000 Euro für den Realzeitadapter.
4: Ubiquitär	Das Testsystem kann aus einem Laptop mit Realzeitadapter bestehen und ist somit transportabel. Während der Arbeit wurde es mehrfach auf Versuchsfahrten mitgenommen.
5: Aufzeichnung von Nachrichten mit Bezug zu einer globalen Uhrzeit	Die durchschnittliche Abweichung der Empfangszeitstempel zweier Realzeitadapter beträgt circa $4ns$ mit einer Standardabweichung von $14ns$.
6: Abspielen von Nachrichten mit Bezug zu einer globalen Uhrzeit	Die durchschnittliche Abweichung zwischen zwei Realzeitadaptern beim Senden von CAN-Nachrichten liegt bei $204ns$ mit einer Standardabweichung von $1,16\mu s$.
7: Anbindung des Testobjekts über CAN	Das ubiquitäre Testsystem hat vier CAN-Schnittstellen.
8: Erkennen einer Verletzung von Echtzeitanforderungen	Die Sendeverzugseinheit erkennt und meldet Abweichungen des Sendezeitpunkts außerhalb einer vorgegebenen Toleranz. Das ubiquitäre Testsystem vermerkt diese Verstöße zusammen mit dem Testergebnis.

Tabelle 7.1.: Erfüllung der gestellten Anforderungen durch das ubiquitäre Testsystem

Insbesondere der Test eines Algorithmus zur Plausibilisierung von Sensordaten und der preisgünstige Ansatz haben das Konzept des ubiquitären Testsystems im Projektumfeld bestätigt, so dass eine Marktreife des Realzeitadapters für das Jahr 2012 angestrebt wird.

7.2. Ausblick

Der Chef von Daimler Dieter Zetsche hält es für denkbar, dass unfallfreies Fahren in zehn Jahren in Deutschland möglich sein wird [111]. Ein wichtiger Bestandteil dieser Vision sind Fahrerassistenzsysteme. Um die Entwicklung dieser Fahrerassistenzsysteme über den in dieser Arbeit vorgestellten Rahmen hinweg zu unterstützen, sind folgende Weiterentwicklungen des ubiquitären Testsystems vorstellbar.

Wie bereits in [125] prototypisch vorgestellt wird, kann das ubiquitäre Testsystem zu einer Rapid Prototyping-Plattform ausgebaut werden. Hierbei werden Modelle aus MATLAB/Simulink über einen Realzeitadapter direkt an einen CAN-Bus angebunden, so dass der modellierte Datenfluss inklusive dessen modellierte Zeitverhalten auf dem CAN-Bus umgesetzt wird. Mit diesem Ansatz kann eine Rapid Prototyping-Plattform erstellt werden, die Entwicklungsmodelle im Fahrzeug erlebbar machen kann ohne einen Plattformwechsel auf ein spezielles Echtzeitsystem.

Ein weiteres Arbeitsfeld kann der Aufbau einer fortlaufenden Integration (engl.: Continuous Integration) für die eingebettete Software von Fahrerassistenzfunktionen sein. Durch den preisgünstigen Ansatz des ubiquitären Testsystems können viele Testsysteme parallel aufgebaut werden, so dass Softwareänderungen nach erfolgreichen Softwaretests automatisch auf ein Steuergerät geladen und getestet werden können. Insbesondere kann dann kontinuierlich und automatisiert das Zeitverhalten der Software auf dem Steuergerät untersucht werden und es können die Varianten für verschiedene Baureihen parallel getestet werden.

Die Eigenschaft, dass jede CAN-Nachricht in ihrem Zeitverhalten individuell beeinflusst werden kann, kann für Robustheitstests verwendet werden. In heutigen Restbussimulationen liegt allen verschickten Nachrichten die selbe Zeitbasis zu Grunde, obwohl innerhalb der Restbussimulation mehrere Steuergeräte simuliert werden. Am ubiquitären Testsystem ist es möglich, den CAN-Nachrichten von verschiedenen simulierten Steuergeräten ein unterschiedliches Zeitverhalten zu geben. So können beispielsweise verschiedene zeitliche Drifts simuliert werden und die Reaktion des Testobjekts hierauf getestet werden. Dieser Ansatz kann zu einem evolutionären Test ausgebaut werden, bei dem das Zeitverhalten so lange verändert wird, bis ein Fehler in der Fahrerassistenzfunktion auftritt. Mit diesem

Ansatz können die Grenzen von nicht synchronen Netzwerken ausgelotet und eventuell sogar erweitert werden.

Die Einführung von FlexRay-Bussystemen im Fahrzeug bedeutet, dass zukünftige ubiquitäre Testsysteme auch dieses Bussystem unterstützen müssen. Im Projektumfeld wird bereits an Lösungen für einen Mischverbau von CAN und FlexRay gearbeitet.

A. Programmierschnittstelle des Prototypen

Im vorliegenden Kapitel wird ein Auszug der für den prototypischen Aufbau des Realzeitadapters verwendeten Programmierschnittstelle dargestellt. Das Kapitel endet mit einem realen Programmierbeispiel.

Nachdem der Realzeitadapter an die Versorgungsspannung angeschlossen ist und über eine Ethernetleitung mit einem PC verbunden ist, kann er über seine Programmierschnittstelle angesprochen werden. Die CAN-Busse können entweder mit geeigneten Kabeln untereinander verbunden werden oder einem bestehenden CAN-Bus zugeschaltet werden. Wenn eine Synchronisation der Uhr auf die globale GPS-Zeit erfolgen soll, muss zusätzlich noch eine GPS-Antenne angeschlossen werden.

Im Folgenden werden die Bibliotheksfunktionen der Programmierschnittstelle aufgeführt. Sie gliedern sich in drei Rubriken. Die erste Rubrik bezieht sich auf das Initialisieren und Beenden der Programmierschnittstelle und der Realzeitadapter-Hardware. In der zweiten Rubrik ist der Umgang mit CAN-Nachrichten abgebildet. In der prototypischen Umsetzungen werden Informationen vom GPS wie Ort, Geschwindigkeit und Anzahl der empfangenen Satelliten als CAN-Nachrichten über einen zusätzlichen CAN-Port übertragen. Die dritte Rubrik bietet die Möglichkeit, das Zeitverhalten der Applikation zu steuern, die den Realzeitadapter verwendet. Die letzte Rubrik ermöglicht es, die Fehlerwarteschlange zu bearbeiten. Im weiteren Verlauf sollen die grundlegenden Funktionen skizziert werden. Für eine detaillierte Beschreibung sei auf die Benutzerdokumentation der aktuellen Implementierung verwiesen.

A.1. Initialisieren und Beenden

rtaOpenDevice()

Über die Funktion `LIBRTA_STATUS rtaOpenDevice(RTA_PCONFIGURATION pConfiguration)` kann die Kommunikation zu einem Realzeitadapter gestartet werden. Als Parameter werden die Eigenschaften für die Kommunikation zwischen der Realzeitadapter-Hardware und dem Computer übergeben, beispielsweise die ID der Netzwerkverbindung.

rtaCloseDevice()

Die Funktion `LIBRTA_STATUS rtaCloseDevice()` setzt die Realzeitadapter-Hardware auf deren Standardeinstellungen zurück und beendet die Programmierschnittstelle.

rtaActivatePorts()

Mit den Funktionen `LIBRTA_STATUS rtaActivatePorts()` werden die vier CAN-Schnittstellen aktiviert. Nur im aktivierten Zustand ist eine Kommunikation über die CAN-Schnittstellen möglich. Ansonsten werden sie galvanisch vom Bus getrennt.

rtaDeactivatePorts()

Mit den Funktionen `RTA_STATUS rtaDeactivatePorts()` werden die CAN-Schnittstellen deaktiviert und galvanisch vom Bus getrennt.

A.2. CAN-Nachrichten senden und empfangen

rtaSendTimeTriggeredMessage()

Mit der Funktion `rtaSendTimeTriggeredMessage(LIBRTA_PORT Port, LIBRTA_PCANMSG pCANMsg)` wird eine Nachricht mit Soll-Sendezeitpunkt an die Sendewarteschlange nach Definition 6 auf Seite 62 übergeben. Die Nachricht wird auf der angegebenen CAN-Schnittstelle (Port) verschickt. Der Soll-Sendezeitpunkt ist ein Bestandteil der Struktur `pCANMsg`. Über diesen Mechanismus kann das Zeitverhalten jeder Nachricht individuell und flexibel gestaltet werden (Anforderung 2).

rtaSendMessage()

Mit der Funktion `LIBRTA_STATUS rtaSendMessage(LIBRTA_PORT Port, LIBRTA_PCANMSG pCANMsg)` kann eine Nachricht so schnell wie möglich verschickt werden.

rtaGetMessage()

Mit `LIBRTA_STATUS rtaGetMessage(LIBRTA_QUEUE Queue, LIBRTA_PPORT pPort, LIBRTA_PCANMSG pCANMsg)` kann eine Nachricht aus der Empfangswarteschlange nach Definition 5 auf Seite 62 für den angegebenen Port ausgelesen werden. Über den Parameter `pPort` kann ausgewählt werden, welche Schnittstelle angesprochen werden soll. Es können sowohl die empfangenen CAN-Nachrichten an den vier verbauten CAN-Schnittstellen ausgelesen werden als auch die empfangenen GPS-Informationen in Form von CAN-Nachrichten.

A.3. Ereignisse und Fehlermeldungen

rtaCreateEvent()

Mit `LIBRTA_STATUS rtaCreateEvent(RTA_EVENT Event, LIBRTA_PTIMESTAMP pPointInTime)` kann ein zeitliches Ereignis auf dem Realzeitadapter hinterlegt werden, um beispielsweise den zeitlichen Ablauf der Software zu steuern, die den Realzeitadapter verwendet. Hierdurch wird beim ubiquitären Testsystem die Echtzeifähigkeit trotz nicht echtzeitfähiger Plattform sichergestellt (Anforderung 1). Durch das zeitliche Ereignis wird der in `Event` festgelegte Zähler zum in `pPointInTime` angegebenen Zeitpunkt einmalig inkrementiert. Mit `rtaGetEventInformation()` kann der aktuelle Zählerstand und wann er das letzte Mal inkrementiert wurde, ausgelesen werden.

rtaCreateCyclicEvent()

Die Funktion `LIBRTA_STATUS rtaCreateCyclicEvent(LIBRTA_EVENT Event, LIBRTA_PTIMESTAMP pTimeInterval)` inkrementiert den in `Event` festgelegten Zähler zyklisch mit einer Periodendauer von `pTimeInterval`. Mit `rtaGetEventInformation()` kann der aktuelle Zählerstand und wann er das letzte Mal inkrementiert wurde ausgelesen werden.

rtaGetEventInformation()

Die Funktion `void rtaGetEventInformation(LIBRTA_EVENT Event, LIBRTA_PEVENTINFORMATION pEventInfo)` kann der aktuelle Zählerstand und wann er das letzte Mal inkrementiert wurde ausgelesen werden.

rtaGetStatistic()

Mit `LIBRTA_STATUS rtaGetStatistic(LIBRTA_PSTATISTIC pStatistic)` kann eine statistische Zusammenfassung der Einträge in der Fehlerwarteschlange nach Definition 9 auf Seite 65 ausgelesen werden. Aus den zurückgelieferten Informationen geht unter anderem hervor, ob CAN-Nachrichten zu spät gesendet wurden.

rtaClearStatistic()

Mit `LIBRTA_STATUS rtaClearStatistic()` kann die statistische Auswertung der Fehlerwarteschlange gelöscht werden.

Reales Programmierbeispiel

Während im Hauptteil der Arbeit in den Listings der besseren Lesbarkeit halber Pseudocode verwendet wird, ist in Listing A.1 ein reales Beispiel dargestellt. Analog zur Pseudocodedarstellung in Listing 4.3 wird gezeigt, wie eine Nachricht zyklisch verschickt werden kann.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <librta.h>
const char *nicId = "\\DEVICE\\{6427158C-D0A9-4C6E-991A-63AAB4DAF602}";

int main(void)
{
    LIBRTA_CONFIGURATION Configuration;
    LIBRTA_STATUS Status;
    LIBRTA_TIMESTAMP Time;
    LIBRTA_CANMESSAGE Message;
    int i;

    Configuration.pNetworkInterfaceIdentifier = (void *)nicId;
    Configuration.DispatcherPriority = LIBRTA_PRIORITY_HIGHEST;
    Configuration.ReceiveMessagePoolElements = 100;
    Configuration.SendMessagePoolElements = 100;
    Configuration.HoldBackTime = 500000;
    Configuration.pPort[0].BaudRate = LIBRTA_BAUDRATE_500K;
    Configuration.pPort[0].Mode = LIBRTA_MODE_SENDONLY;
    Configuration.pPort[0].Tolerance = 1000000;
    Configuration.pPort[0].Queue = LIBRTA_QUEUE_0;

    Status = rtaOpenDevice(&Configuration);
    if (LIBRTA_ERROR(Status)) return EXIT_FAILURE;

    Time.Seconds = 0;
    Time.Nanoseconds = 0;
    rtaSetTime(&Time);
    rtaActivatePorts();

    Message.ID = 0x123;
    Message.Length = 8;
    Message.Timestamp.Seconds = 15;

    //CAN-Paket mit Daten füllen
    for(i=0; i < 8; i++) Message.pData[i] = i;

    for(i=1; i < 11; i) {
        Message.Timestamp.Nanoseconds = i * 10000;
        rtaSendTimeTriggeredMessage(LIBRTA_PORT_0, &Message);
    }

    Sleep(16000); //[ms]

    rtaDeactivatePorts();
    rtaCloseDevice();

    return EXIT_SUCCESS;
}
```

Listing A.1: Zyklisches Versenden einer Nachricht mit dem Realzeitadapter

B. Begriffsverzeichnis

Anwendungsfunktion

Eine Anwendungsfunktion (engl.: Use Case) beschreibt eine Leistung eines Systems. Nach Bühler [9] wird in dieser Arbeit der Begriff Anwendungsfunktion für eine durch den Kunden erlebbare Funktionalität eines Fahrzeugs verwendet. Für die Anwendungsfunktionen von Fahrerassistenzsystemen wird der Begriff Fahrerassistenzfunktion verwendet. Fahrerassistenzfunktionen sind oft sicherheitskritisch, weil sie teilweise autonom in das Fahrzeugverhalten eingreifen.

Arbeitsplatzrechner

In dieser Arbeit wird mit dem Begriff Arbeitsplatzrechner ein Computer bezeichnet, wie er in großen Firmen standardmäßig von einer zentralen Stelle installiert, ausgeliefert und gewartet wird. In der Regel besteht ein solcher Rechner aus einer PC-Hardware der mittleren bis gehobenen Leistungsklasse, einem nicht echtzeitfähigen Betriebssystem (in der Regel Microsoft Windows [76]), einer Verbindung zum Hausnetz und Standardanwendungen wie Mailprogramm, Textverarbeitung, Virenscanner und Entwicklungswerkzeuge. In der Regel verfügen diese Geräte über genügend Rechenleistung und Hauptspeicher für komplexe Simulationen, so dass Software-in-the-Loop-Tests durchgeführt werden können. Die Installation einer echtzeitfähigen Betriebssystemlösung ist meist nur in Ausnahmefällen und zu deutlich höheren Kosten möglich.

Eingebettetes System (engl.: Embedded System)

Ein Teilsystem, das in eine Umgebung oder in ein größeres System eingebunden ist, wird als eingebettetes System bezeichnet. Die Funktionalität eines eingebetteten Systems wird in der Regel durch das Zusammenspiel von Hardware und Software erzeugt [96].

Fahrerassistenzsystem (FAS)

Ein Fahrerassistenzsystem soll den Fahrer eines Fahrzeugs entlasten und dadurch die Fahrsicherheit erhöhen [73]. Ein Fahrerassistenzsystem besteht in der Regel aus mehreren Steuergeräten mit unterschiedlichen Teilaufgaben. Diese sind Erfassung der Umgebung des Fahrezugs (Sensorik), Verarbeitung der Daten und Eingriff in das Fahrverhalten (Aktorik).

Funktionsorientierter Test

Nach Veenedaal in [130] basiert ein Funktionsorientierter Test auf einer Analyse der funktionalen Spezifikation eines Systems oder eines Teilsystems.

Global Positioning System (GPS)

Das Global Positioning System (GPS) ist ein satellitengestütztes System zur weltweiten Positionsbestimmung. Die Positionsbestimmung basiert auf einer hochpräzisen Laufzeitmessung zwischen dem Empfänger, dessen Position bestimmt werden soll, und den Satelliten. Für die Laufzeitmessung wird die lokale Uhr im Empfänger auf die Atomuhren in den Satelliten synchronisiert. Neben den Ortskoordinaten wird von einem GPS-Empfänger eine weltweit eindeutige und präzise Zeitinformation geliefert [143].

Hardware-in-the-Loop (HiL)

Bei einer Hardware-in-the-Loop Simulation werden physikalische Teile eines realen Systems durch eine Simulation ersetzt. Für die HiL Simulation wird der zu testende Teil des realen Systems – hier ein Steuergerät – über seine Schnittstellen an das HiL Testsystem angeschlossen [64, 75, 99]. Das HiL Testsystem kann dabei Open-Loop oder Closed-Loop betrieben werden [99]. Bei ersterem wird das Testobjekt mit vorgefertigten Daten stimuliert. Diese können beispielsweise in einem Fahrversuch aufgezeichnet worden sein [63]. Beim Closed-Loop Betrieb basiert die Berechnung der Simulationsdaten auf den vorherigen Ausgabegrößen des Testobjekts [99].

Interagierende Fahrzeuge

Interagieren bedeutet „aufeinander bezogen handeln“ [135]. Durch die umgebungserfassende Sensorik können Fahrerassistenzsysteme ohne direktes Zutun des Fahrers auf umgebende Fahrzeuge (Objektfahrzeuge) bezogen handeln.

Objektfahrzeug

Als Objektfahrzeug wird ein Fahrzeug bezeichnet, das sich innerhalb der Reichweite der umgebungserfassenden Sensorik befindet und als Objekt erfasst wird.

PC-Testsystem

Der Begriff PC-Testsystem bezeichnet im Rahmen dieser Arbeit eine derzeit existierende projektspezifische Lösung für ein Testsystem auf Basis eines nicht echtzeitfähigen Betriebssystems auf einem Arbeitsplatzrechner (engl.: Personal Computer (PC)) und einer einfachen CAN-Schnittstelle (in vorliegenden Fall ein CANcaseXL der Firma Vector [129]). Ein PC-Testsystem erfüllt die Anforderungen 3, 4 und 7 der acht Anforderungen, die im Rahmen dieser Arbeit an Testsysteme für Fahrerassistenzsysteme gestellt werden (siehe Kapitel 4).

Realzeitadapter

Ein Realzeitadapter ermöglicht es, das Zeitverhalten auf einer Schnittstelle unabhängig vom Dateninhalt einer Nachricht reproduzierbar festzulegen. In der vorliegenden Arbeit wird dieser Ansatz für eine CAN-Schnittstelle betrachtet. Der Realzeitadapter synchronisiert seine lokale Uhr auf die vom GPS bereitgestellte globale Zeit und bietet somit eine eindeutige und reproduzierbare Zeitbasis. Eine empfangene CAN-Nachricht wird mit einem Zeitstempel auf Basis der lokalen Uhr versehen. Der Sendezeitpunkt einer zu sendenden Nachricht kann bezogen auf die lokale Uhr des Realzeitadapters vorgegeben werden und wird vom Realzeitadapter entweder innerhalb einer einstellbaren Toleranz eingehalten oder ein Verstoß wird zuverlässig detektiert.

RTOS-Testsystem

Der Begriff RTOS-Testsystem bezeichnet im Rahmen dieser Arbeit ein derzeitiges Testsystem, das auf Basis eines Realzeitbetriebssystems (engl.: Real-Time Operating System (RTOS)) aufgebaut ist, wie es beispielsweise von den Firmen ETAS [141], dSPACE [142], Berner&Mattner [64] und weiteren angeboten wird. Im Besonderen wird in der vorliegenden Arbeit das Zeitverhalten eines RTOS-Testsystems der Firma ETAS betrachtet. Das für die CAN-Busse spezifizierte Verhalten wird hier durch das Realzeitbetriebssystem umgesetzt. Ein RTOS-Testsystem erfüllt die Anforderungen 1 und 7 der acht Anforderungen, die im Rahmen dieser Arbeit an Testsysteme für Fahrerassistenzsysteme gestellt werden (siehe Kapitel 4).

Steuergerät, engl.: Electronic Control Unit (ECU)

Ein im Fahrzeug eingebettetes System setzt sich zumeist aus mehreren Sensoren und Aktoren, sowie einem oder mehrerer Steuergeräte zusammen. Aufgabe der Steuergeräte ist die Verarbeitung der durch die Sensoren gelieferten Informationen und einer darauf basierenden Ansteuerung der Aktoren. Ein Steuergerät

besteht dabei aus einem oder mehreren Prozessoren und deren Peripherie, die in einem Gehäuse untergebracht sind [39].

Systemfahrzeug

Das Systemfahrzeug ist das Fahrzeug, in dem das Fahrerassistenzsystem von Interesse verbaut ist.

Testmethode

Nach Spillner et al. in [110] ist eine Testmethode ein planmäßiges und regelbasiertes Vorgehen zur systematischen Ermittlung von Testfällen.

Testsystem

Ein Testsystem besteht im Rahmen dieser Arbeit aus einem Rechnersystem, das die Umgebung des Testobjekts simuliert, sowie einer Kommunikationsanbindung zum Testobjekt über CAN.

Ubiquitäres Testsystem

Die Kombination aus einem Realzeitadapter und einem Arbeitsplatzrechner wird im Rahmen dieser Arbeit ubiquitäres Testsystem genannt. Dieses Testsystem ist geeignet für folgende drei Einsatzgebiete:

- Zeitstempel-basiertes Testsystem am Arbeitsplatz
- Zeitstempel-basiertes Testsystem zur Mitnahme auf ein Prüfgelände
- Datenlogger für interagierende Fahrzeuge

Ein ubiquitäres Testsystem erfüllt alle acht Anforderungen, die im Rahmen dieser Arbeit an Testsysteme für Fahrerassistenzsysteme gefordert werden (siehe Kapitel 4).

Ubiquität

Nach Brockhaus kommt der Begriff Ubiquität vom lateinischen Wort *ubique*, welches „überall“ bedeutet. Für die Wirtschaft definiert Brockhaus mit Ubiquität „die Erhätlichkeit eines Gutes an jedem Ort“, während in der Biologie „das Nichtgebundensein an einen Standort“ gemeint ist [104]. Das Adjektiv ubiquitär wird bildungssprachlich im Sinne von „überall verbreitet“ verwendet.

Literaturverzeichnis

[1] American National Standards Institute, 25 West 43rd Street, New York, New York 10036. *ISO/IEC 2382-1, Information technology – Vocabulary – Part 1: Fundamental terms* , 3. Ausgabe, November 1993.

[2] D. Ammon. Künftige Fahrdynamik- und Assistenzsysteme. *atp 46(2004) Heft 6*, 2004.

[3] K. Athanasas. *Fast Prototyping Methodology for the Verification of Complex Vehicle Systems*. Dissertation, Brunel University, London, UK, March 2005.

[4] J. Badstübner. Die Kamera sieht mehr. *Automobil Industrie*, 5/2010:58–59, 2010.

[5] A. Bartels, S. Steinmeyer, S. Brosig, and C. Spichalsky. Fahrstreifenwechselassistent. *Handbuch Fahrerassistenzsysteme*, Seiten 562–571, 2009.

[6] D. Barz and Drews R. Kompatible Messsyteme – Unterschiedliche Fahrdynamikaufgaben synchron messen. *ATZelektronik Heft 4*, 2008.

[7] A. Bauch, P. Hetzel, and D. Piester. Zeit- und Frequenzverbreitung mit DCF77: 1959 – 2009 und darüber hinaus. *PTB-Mitteilungen*, Heft 3(119):217–240, 2009.

[8] K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.

[9] O. Bühler. *Evolutionärer Funktionstest von eingebetteten Systemen für abstandsbasierte Fahrerassistenzfunktionen im Automobil*. Dissertation, Eberhard Karls Universität Tübingen, 2007.

[10] T. Breitling, J. Breuer, L. Dragon, M. Leucht, J. Pasquini, and U. Peterson. Sicheres Fahren. *ATZextra*, Januar 2009:72–78, 2009.

[11] J. Breuer. Bewertungsverfahren von Fahrerassistenzsystemen. *Handbuch Fahrerassistenzsysteme*, Seiten 55–68, 2009.

[12] B. Broekman and E. Notenboom. *Testing Embedded Software*. Addison-Wesley, London, 2003.

[13] M. Broy, G. Reichart, and L. Rothhardt. Architekturen softwarebasierter Funktionen im Fahrzeug: von den Anforderungen zur Umsetzung. *Informatik Spektrum*, 34:42–59, 2011.

[14] B. Buehm. *Software Engineering Economics*. Prentice Hall, 1982.

[15] R. Charette. This car runs on code. *IEEE Spectrum*, 2009.

[16] M. Conrad. *Modell-basierter Test eingebetteter Software im Automobil*. Dissertation, Technische Universität Berlin, 2004.

[17] Crash Avoidance Metrics Partnership on behalf of the Crash Imminent Braking Consortium, 39255 Country Club Drive, Suite B-40 Farmington Hills, MI 48331. *Objective Tests for Imminent Crash Automatic Braking Systems, Final Report*, 2010.

[18] S. Dangel, H. Keller, and D. Ulmer. Wie sag' ich's meinem Prüfstand? *RD Inside*, April/Mai:7, 2010.

[19] M. Darms. Fusion umfelderfassender Sensoren. *Handbuch Fahrerassistenzsysteme*, Seiten 237–248, 2009.

[20] M. Dausmann, U. Bröckl, and J. Goll. *C als erste Programmiersprache*. Teubner, Wiesbaden, 5. Ausgabe, 2005.

[21] E. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, Seiten 60–66. Springer-Verlag, 1982.

[22] E. W. Dijkstra, O. J. Dahl, and C. A. R. Hoare. *Structured programming*. Academic Press, 1972.

[23] E. Donges. Aspekte der Aktiven Sicherheit bei der Führung von Personenkraftwagen. *Automobilindustrie*, 27:183–190, 1982.

[24] E. Donges. Fahrerverhaltensmodelle. *Handbuch Fahrerassistenzsysteme*, Seiten 15–23, 2009.

[25] E. Donner, T. Winkle, R. Walz, and J. Schwarz. RESPONSE 3 – Code of Practice für die Entwicklung, Validierung und Markteinführung von Fahrerassistenzsystemen. *VDA. Technischer Kongress*, 2007.

[26] K. Etschberger. *Controller-Area-Network*. hanser, München, 3. Ausgabe, 2002.

[27] M. Fischer and J. Bihlmayr. Update der bestehenden Entwicklung von Fahrerassistenzsystemen auf TargetLink 3.0. *6. dSPACE Anwenderkonferenz*, 2010.

[28] *FlexRay Communications System - Protocol Specification*. http://www.flexray.com, Oktober 2010.

[29] G. Forkenbrock and B. O'Harra. A Forward Collision Warning (FCW) Performance Evaluation. *National Highway Traffic Safety Administration*, Paper No. 09-0561, 2009.

[30] K. Frühauf, J. Ludewig, and H. Sandmayr. *Software-Prüfung, eine Fiebel*. Teubner, Stuttgart, 1991.

[31] J. Friedrich, U. Hammerschall, M. Kuhrmann, and M. Sihling. *Das V-Modell XT*. Springer, Berlin Heidelberg, 2009.

[32] T. Galla. *Cluster Simulation in Time-Triggered Real-Time Systems*. Dissertation, Technische Universität Wien, Austria, 1999.

[33] J. Gayko. Lane Departure Warning. *Handbuch Fahrerassistenzsysteme*, Seiten 543–553, 2009.

[34] A. Gern. *Multisensorielle Spurerkennung für Fahrerassistenzsysteme*. Dissertation, Universität Stuttgart, Oktober 2005.

[35] J. Goll. *Methoden der Softwaretechnik*. Vieweg+Teubner, 1. Ausgabe, 2011.

[36] G. Goppelt. Assistenzfunktionen im neuen Passat - Der Fortschritt im Mittelklassemodell findet vor allem bei der Komfortelektronik statt. *Heise Zeitschriften Verlag*, 2011.

[37] M. Gröbel. *Datenaufzeichnung im Motorsport*. Diplomica, Hamburg, 2007.

[38] E. Hanser. *Agile Prozesse: von XP über Scrum bis MAP*. Springer, Heidelberg, 2010.

[39] C. Heinisch. *Konfigurationsmodell und Architektur für eine automatisierte Software-Aktualisierung von Steuergeräten im Automobil*. Dissertation, Eberhard Karls Universität Tübingen, 2004.

[40] L. Henle, U. Regensburger, B. Danner, E. Hentschel, and C. Hämmerling. Fahrerassistenzsysteme. *ATZextra*, Januar 2009:56–62, 2009.

[41] R. Höhn and S. Höppner. *Das V-Modell XT*. Springer, Heidelberg, 2008.

[42] K. Hünlich. Code generation in MATLAB Simulink optimized for an embedded distributed measurement system. Master's thesis, Brunel University, November 2008.

[43] A. Hoffmann. Timed Automata, 2006.

[44] J. Hoffmann and H. Winner. EVITA – Das Prüfverfahren zur Beurteilung von Antikollisionssystemen. *Handbuch Fahrerassistenzsysteme*, Seiten 69–75, 2009.

[45] A. Hohm, R. Mannale, K. Schmitt, and C. Wojek. Vermeidung von Überholunfällen. *ATZ*, 10:712–718, 2010.

[46] Robert Bosch GmbH (Hrsg.). *Sicherheits- und Komfortsysteme*. Vieweg, Wiesbaden, 3. Ausgabe, 2004.

[47] IEEE, Watling Street, Nuneaton, Warwickshire. *IEEE1588, Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*.

[48] IEEE Standards Board. *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990*, Januar 1991. IEEE Publications.

[49] The Institute of Electrical and Electronics Engineers, New York. *IEEE754: IEEE Standard for Binary Floating-Point Arithmetic*, 1985.

[50] International Electrotechnical Commission, Genf. *IEC61508: Functional Safety of E/E/PES Systems*, Dezember 1998.

[51] International Electrotechnical Commission TC204/WG14. *ISO 22179: Intelligent transport systems – Full speed range adaptive cruise control (FSRA) systems – Performance requirements and test procedures*, 2008.

[52] International Organization for Standardization. *ISO/DIS 26262: Road vehicles – Functionl safety – Part 1-10*.

[53] International Organization for Standardization. *ISO 11898-1; Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling*, 2003.

[54] International Organization for Standardization. *ISO 11898-2; Road vehicles - Controller area network (CAN) - Part 2: High speed medium access unit*, 2003.

[55] International Organization for Standardization. *ISO-WD 11898-4; Road vehicles - Controller area network (CAN) - Part 4: Time-triggered communication*, 2004.

[56] International Organization for Standardization. *ISO 11898-3; Road vehicles - Controller area network (CAN) - Part 3: Low speed medium access unit*, 2006.

[57] International Organization for Standardization TC204/WG14. *ISO 15622: Transport information and control syystems – Adaptive Cruise Control systems – Performance requirements and test procedures*, 2002.

[58] R. Isermann, E. Bender, R. Bruder, M. Darms, M. Schorn, U. Stählin, and H. Winner. Antikollisionssystem PRORETA – Integrierte Lösung für ein unfallvermeidendes Fahrzeug. *Handbuch Fahrerassistenzsysteme*, Seiten 632–646, 2009.

[59] R. Isermann, B. Schiele, H. Winner, A. Hohm, R. Mannale, K. Schmitt, C. Wojek, and S. Lüke. Elektronische Fahrerassistenz zur Vermeidung von Überholunfällen – PRORETA 2. *VDI-Berichte*, 2075:525–541, 2009.

[60] M. Keßler and Mangin B. Nutzungsorientierte Auslegung von teilautomatisierten Einparksystemen. *Tagungsband zur 4. VDI-Tagung „Fahrer im 21. Jahrhundert", Braunschweig*, 2007.

[61] H. Kubin, H. Preiß, C. Vetter, and D. Walther. Rapid-Prototyping mit VME-Bus-Echtzeitrechner unter Nutzung von MATLAB/Simulink. *atp 44 (2002) Heft 9*, Seiten 66–72, September 2002.

[62] H. Lang. Virtuelle Integration. *Insight Automotive*, 2:3–4, 2010.

[63] H. Lang. Vom Fahrversuch ins Labor. *Insight Automotive*, 2:6–7, 2010.

[64] F. Langner, U. Alsmann, and J. Meyer. Fahrversuch versus HiL-Test. *White-Paper Berner & Mattner*, 2008.

[65] R. Lerch. *Elektrische Messtechnik*. Springer, Berlin Heidelberg, 2006.

[66] P. Liggesmeyer and D. Rombach. *Software Engineering eingebetteter Systeme*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 2005.

[67] J. Luther and H. Schaal. Luftbrücke für Busdaten. *Elektronik automotive*, 7/8:38–41, 2010.

[68] H. Machens, R. Kroger, S. Jell, and K. Grabenweger. Replay-basiertes Testen von MOST-Bus-Anwendungen im Automotive-Umfeld. *Software Engineering*, 121:85–91, 2008.

[69] W. Marrison. The Evolution of the Quartz Crystal Clock. *Reprinted from The Bell System Technical Journal*, XXVII:510–588, 1948.

[70] C. Marscholik and P. Subke. *Datenkommunikation im Automobil*. Hüthig, Berlin Heidelberg, 2007.

[71] MathWorks. *MATLAB/Simulink*. http://www.mathworks.de.

[72] M. Maurer. Entwurf und Test von Fahrerassistenzsystemen. *Handbuch Fahrerassistenzsysteme*, Seiten 43–54, 2009.

[73] M. Maurer and C. Stiller (Hrsg.). *Fahrerassistenzsysteme mit maschineller Wahrnehmung*. Springer, Berlin Heidelberg, 2005.

[74] U. Mellinghoff. Virtuelle Erprobung, Prüfstände, Straßenerprobung - der neue Weg der Entwicklung. *RD Inside*, März/April:5, 2011.

[75] J. Meyer, F. Langner, and U. Alsmann. Fahrversuch versus HiL-Test. *elektroniknet*, Juni 2008.

[76] Microsoft. *Windows*. http://www.microsoft.com/windows.

[77] C. Müller. *Praxishandbuch Medien-, IT- und Urheberrecht*. Hüthig Jehle Rehm, Heidelberg, 2008.

[78] K.H. Möller. Ausgangsdaten für Qualitätsmetriken – Eine Fundgrube für Analysen. *Softwaremetriken in der Praxis*, Seite 105–116, 1996.

[79] K.D. Müller-Glasser, A. Burst, B. Spitzer, and M. Kühl. Rapid Prototyping von eingebetteten elektronischen Systemen. *it + ti - Informationstechnik und Technische Informatik 42 (2000) 2*, Seiten 8–15, 2/2000.

[80] K.D. Müller-Glasser, A. Burst, B. Spitzer, and S. Schmerler. Rapid Prototyping von Informationssystemen für Kraftfahrzeuge. *it + ti - Informationstechnik und Technische Informatik 41 (1999) 5*, Seiten 12–18, 5/1999.

[81] G.J. Myers. *Methodisches Testen von Programmen*. R. Oldenbourg Verlag, Wien, 4. Ausgabe, 1991.

[82] National Highway Traffic Safety Administration, Washington DC. *Forward Collision Warning System Confirmation Test*, 2010.

[83] R. Otterbach, M. Eckmann, and F. Mertens. Rapid Control Prototyping - neue Möglichkeiten und Werkzeuge. *atp 46(2004) Heft 6*, 2004.

[84] J. Peleska and Zahlten C. Hard Real-Time Test Tools - Concepts and Implementation. In *Proceedings of the 4th ICSTEST, International Conference on Software Testing*, 2003.

[85] J. Peleska, A. Honisch, F. Lapschies, H. Löding, H. Schmid, P. Smuda, E. Vorobev, and C. Zahlten. A Real-World Benchmark Model for Testing Concurrent Real-Time Systems in the Automotive Domain. In *Burkhart Wolff and Fatiha Zaidi (Eds.): Testing Software and Systems. Proceedings of the 23rd IFIP WG 6.1 International Conference, ICTSS*, Seiten 146–161, 2011.

[86] R. Isermann (Hrsg.). *Fahrdynamik-Regelung*. Vieweg, Wiesbaden, 2006.

[87] W. Raju. Fundamentals of GPS. *Satellite Remote Sensing and GIS Applications in Agricultural Meteorology*, Seiten 121–150, 2004.

[88] J. Rasmussen. Skills, Rules and Knowledge; Signals, Signs and Symbols and other Distinctions in Human Performance Models. *IEEE Transactions on Systems, Man and Cybernetics*, SMC 13(3):257–266, 1983.

[89] A. Rau. *Modellbasierte Entwicklung von eingebetteten Regelungssystemen in der Automobilindustrie*. Dissertation, Eberhard Karls Universität Tübingen, 2002.

[90] J. Rees and A. Menn. Das Auto-Net. *WirtschaftsWoche*, 48:100–103, November 2010.

[91] G. Reichart and J. Bielefeld. Einflüsse von Fahrerassistenzsystemen auf die Systemarchitektur im Kraftfahrzeug. *Handbuch Fahrerassistenzsysteme*, Seiten 84–92, 2009.

[92] C. Reichmann, M. Kühl, Graf P., and K. D. Müller-Glaser. Durchgängige Entwurfsmethodik dezentraler Steuerungselemente für mechatronische Systeme in der Automatisierungstechnik (DESSY). *Abschlussbericht DFG-Schwerpunktprogramm 1040, Universität Tübingen*, 1997-2003.

[93] K. Reif. *Automobilelektronik*. Vieweg+Teubner, Wiesbaden, 3. Ausgabe, 2009.

[94] Robert Bosch GmbH (Hrsg.). *Kraftfahrtechnisches Taschenbuch*. Vieweg, Wiesbaden, 25. Ausgabe, 2004.

[95] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1. Ausgabe, 1999.

[96] W. Rosenstiel. Entwurf und Entwurfsmethodik eingebetteter Systeme. *Abschlussbericht DFG-Schwerpunktprogramm 1040, Universität Tübingen*, 1997-2003.

[97] B. Ruf, H. Keller, D. Ulmer, and M. Dausmann. Ereignisbasierte Testfallbedatung - ein MINT-Projekt der Daimler AG und der Fakultät Informationstechnik. *spektrum*, 33:68–70, 2011.

[98] V. Schindler and I. Sievers. *Forschung für das Auto von Morgen*. Springer, Berlin Heidelberg, 2008.

[99] M. Schlager. *Hardware-in-the-Loop Simulation*. VDM Verlag Dr. Müller, Dudweiler, 2008.

[100] M. Schlager, R.n Obermaisser, and Wilfried Elmenreich. A framework for hardware-in-the-loop testing of an integrated architecture. In *Proceedings of the 5th IFIP WG 10.2 International Conference on Software Technologies for Embedded and Ubiquitous Systems, SEUS'07*, volume 4761, Seiten 159–170. Springer Berlin / Heidelberg, 2007.

[101] C. Schmidt. *Hardware-in-the-Loop gestützte Entwicklungsplattform für Fahrerassistenzsysteme*. Dissertation, Universität Kassel, 2010.

[102] J. Schmitt. Entwicklungsumgebung mit echtzeitfähigen Gesamtfahrzeugmodellen für sicherheitsrelevante Fahrerassistenzsysteme. *Fahrdynamik-Regelung: Modellbildung, Fahrerassistenzsysteme, Mechatronik*, Seiten 365–375, 2006.

[103] P. Scholz. *Softwareentwicklung eingebetteter Systeme*. Springer, Berlin Heidelberg, 2005.

[104] W. Scholze-Stubenrecht et al. *Brockhaus die Enzyklopädie*. F.A.Brockhaus, Leipzig Mannheim, 20. Ausgabe, 1996.

[105] K. Schwaber. *Scrum im Unternehmen*. Microsoft Press, 1. Ausgabe, 2008.

[106] M. Schwarze, O. Meyer, and R. Scholz. Next Generation Test Engine Environment for Aircraft Cabin Systems Testing. In *Proceedings of the 3rd SQC, Software and Systems Quality Conferences*, 2008.

[107] J. Seekircher, B. Woltermann, A. Gern, R. Janssen, D. Mehren, and M. Lallinger. Das Auto lernt sehen. *ATZextra*, Januar 2009:64–70, 2009.

[108] A. Siebenborn, S. Schmitt, and W. Rosenstiel. Entwurf und Bewertung eingebetteter Systeme. *Abschlussbericht DFG-Schwerpunktprogramm 1040, Universität Tübingen*, 1997-2003.

[109] D. Simmes. *Entwicklungsbegleitender Systemtest für elektronische Fahrzeugsteuergeräte*. Dissertation, Technische Universität München, 1997.

[110] A. Spillner and T. Linz. *Basiswissen Softwaretest*. dpunkt.verlag, Heidelberg, 2003.

[111] Stuttgarter Nachrichten. *Zetsche: Bald unfallfreies Fahren*. 04.07.2011.

[112] P. Subke. *Handbuch Kfz-Elektronik*. vieweg, Wiesbaden, 2006.

[113] *TTTech Automotive GmbH*. http://http://www.tttech.com.

[114] TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria. *Time-Triggered Protcol TTP/C - High Level Specification Document*, Juli 2002.

[115] UC Berkeley Center for Hybrid and Embedded Software Systems. *Precision Timed (PRET) Machines*. http://chess.eecs.berkeley.edu/pret/.

[116] UC Berkeley EECS Dept. *Ptolemy Project*. http://ptolemy.eecs.berkeley.edu/index.htm.

[117] D. Ulmer. Datenaufzeichnungssystem und Verfahren zur Erfassung von Daten mittels eines Datenaufzeichnungssystems. *Schutzrecht DE 10 2007 015 762*, 02.10.2008.

[118] D. Ulmer. Datenaufzeichnungssystem und Verfahren zur Erfassung von Daten mittels eines Datenaufzeichnungssystems. *Europäisches Patent EP 2 132 716*, 16.12.2009.

[119] D. Ulmer and O. Bühler. Vorrichtung zur Funktionsprüfung eines Zielsystems. *Schutzrecht DE 10 2007 043 267*, 12.03.2009.

[120] D. Ulmer and O. Bühler. Vorrichtung zur Funktionsprüfung eines Zielsystems. *Schutzrecht DE 10 2008 031 829*, 21.01.2010.

[121] D. Ulmer and O. Bühler. Vorrichtung zur Funktionsprüfung eines Zielsystems. *Europäisches Patent EP 2 040 169*, 25.03.2009.

[122] D. Ulmer, O. Bühler, and K. Hünlich. Vorrichtung zur Funktionsprüfung eines Fahrzeugs. *Europäisches Patent EP 2 088 439*, 12.08.2009.

[123] D. Ulmer, O. Bühler, and K. Hünlich. Vorrichtung zur Funktionsprüfung eines Fahrzeugs. *Schutzrecht DE 10 2008 008 012 (erteilt)*, 15.10.2009.

[124] D. Ulmer and A. Theissler. Anwendung des V-Modells für die Entwicklung interagierender Fahrzeuge und daraus resultierende Anforderungen an Testwerkzeuge. *Software and Systems Quality Conferences*, 2009.

[125] D. Ulmer and S. Wittel. Approach for a Real-Time Hardware-in-the-Loop System Based on a Variable Step-Size Simulation. *International Conference on Testing Software and Systems*, 22nd, November 2010.

[126] D. Ulmer, S. Wittel, K. Hünlich, and W. Rosenstiel. A Hardware-in-the-Loop Testing Platform Based on a Common Off-The-Shelf Non-Real-Time Simulation PC. *International Conference on Systems*, 2011.

[127] D. Ulmer, S. Wittel, K. Hünlich, and W. Rosenstiel. Testing Platform for Hardware-in-the-Loop and In-Vehicle Testing Based on a Common Off-

The-Shelf Non-Real-Time PC. angenommen bei: International Journal On Advances in Systems and Measurements, 2011.

[128] *V-Modell XT*. vmodell.iabg.de, 2007.

[129] Vector Informatik GmbH, Ingersheimer Str. 24 70499 Stuttgart. *CANcaseXL - USB 2.0 Interface for CAN and LIN*.

[130] E. v. Veenendaal. Standard glossary of terms used in Software Testing. *Version 1.2, Produced by the 'Glossary Working Party', International Software Testing Qualification Board*, Juni 2006.

[131] *Verified Systems International GmbH*. http://www.verified.de.

[132] R. Vig. *Introduction to Quartz Frequency Standards*. http://www.ieee-uffc.org/freqcontrol/quartz/vig/vigtoc.htm, SLCET-TR-92-1 (Rev. 1), 1992.

[133] C. von Glasner and S. Micke. Driver Assistance Functions. Status 2010. In *33rd FISITA World Automotive Congress*. FISITA, 2010.

[134] R. von Häfen. Realzeit-Restbussimulation mit eingebetteten Systemen für eine effiziente Steuergeräteentwicklung und Systemintegration. *VDI-Berichte*, 2075:367–380, 2009.

[135] R. Wahrig-Burfeind. *Fremdwörterlexikon*. Wissen Mesia Verlag GmbH, 5. Ausgabe, 2004.

[136] T. Weber. Neue Ära optischer Technologien. *Automobil Industrie*, 5/2010:56–57, 2010.

[137] J. Wegener, H. Sthamer, B. Jones, and D. Eyers. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6:127–135, 1997.

[138] H. Winner, B. Danner, and J. Steinle. Adaptive Cruise Control. *Handbuch Fahrerassistenzsysteme*, Seiten 478–521, 2009.

[139] H. Winner, S. Hakuli, and G. Wolf (Hrsg.). *Handbuch Fahrerassistenzsysteme*. Vieweg+Teubner, Wiesbaden, 1. Ausgabe, 2009.

[140] G. Wittler. HiL-Tests mit PC-Technologie. *AUTOMOBIL-ELEKTRONIK Juni*, 2008.

[141] G. Wittler. PC-Standardtechnologie im praktischen Einsatz für HiL-Testsysteme. *ETAS GmbH*, 2008.

[142] P. Wältermann. „Hardware-in-the-Loop“: Die Technologie zum Test elektronischer Steuerungen und Regelungen in der Fahrzeugtechnik. *6. Paderborner Workshop „Entwurf mechatronischer Systeme“*, 2009.

[143] G. Xu. *GPS*. Springer, Berlin Heidelberg, 2. Ausgabe, 2007.

[144] Y. Zhao. *On the Design of Concurrent, Distributed Real-Time Systems*. Dissertation, UC Berkeley, 2009.

[145] Y. Zhao, J. Liu, and E. Lee. A Programming Model for Time-Synchronized Distributed Real-Time Systems. In *13th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, 2007.

[146] W. Zimmermann and R. Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Vieweg+Teubner, Wiesbaden, 2007.