Projektgruppe RCCARS

Realtime Controlled Cooperative Autonomous Racing System

Endbericht

Nikolai Bräuer
Michael Bukowski
Anatolij Fandrich
Tom Reske

CARL VON OSSIETZKY UNIVERSITÄT OLDENBURG
DEPARTMENT FÜR INFORMATIK

Version 1.0.1
20. November 2016

Veranstalter:
Prof. Dr. Werner Damm
Prof. Dr. Martin Fränzle

Betreuer:
Dipl.- Inform. Günter Ehmen
Dipl.- Inform. Stefan Puch

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Projekt *RCCARS* wurde von den Abteilungen *Hybride Systeme* und *Sicherheitskritische Eingebettete Systeme* der Carl von Ossietzky Universität Oldenburg initiiert. *RCCARS* steht dabei für **R**ealtime **C**ontrolled **C**ooperative **A**utonomous **R**acing **S**ystem. Ziel des Projektes ist die Entwicklung eines funktionsfähigen, sicherheitskritischen Systems, dessen Aufgabe in der Beobachtung und Steuerung autonom agierender Modellfahrzeuge auf einer Rennstrecke besteht.
Autonome Fahrzeugsteuerungssysteme sind – sowohl in der Wissenschaft als auch in der Wirtschaft – derzeit einer der aktivsten Forschungsbereiche. Die Entwicklung solcher zukünftiger Systeme ist jedoch mit hohen Kosten verbunden. So gab beispielsweise der Automobilhersteller *General-Motors* bekannt, dass der Konzern im Jahr 2016 ca. 500 Mio. USD in die Weiterentwicklung des autonom gesteuerten Taxidienstes *Lyft* investieren wird [Lyf16].
Das Projekt *RCCARS* arbeitet daher in einem verkleinerten Maßstab unter Verwendung von ferngesteuerten Modellfahrzeugen, da der Fokus auf der Entwicklung von Regelungsalgorithmen zur Steuerung der Fahrzeuge liegt und nicht auf der Realisierung eines realen autonomen Fahrzeugsystems.

Im Rahmen der Projektgruppe, einer Lehrveranstaltung in den Masterstudiengängen der Informatik, haben *wir* – ein Team von vier Studierenden – ein Jahr lang Zeit, alle grundlegenden Bestandteile dieses Systems zu realisieren und die Voraussetzungen für mögliche Folgeprojekte zu schaffen. Diese erste Ausbaustufe beinhaltet in erster Linie den Aufbau und die Konfiguration der benötigten Hardwarekomponenten und die Implementierung einer zuverlässigen und ausreichend präzisen autonomen Fahrzeugsteuerung.

1.1 Rechtliches

Alle in diesem Dokument verwendeten Grafiken und Textabschnitte sowie für das Projekt *RCCARS* relevante Markennamen, Produkte und Projekte anderer Personen, Gruppen und Firmen werden - beim ersten Auftauchen innerhalb eines Kapitels - durch entsprechende Referenzen gekennzeichnet, so dass die Urheberrechte gewahrt werden.

1.2 Struktur des Dokuments

Die Dokumentation setzt sich im wesentlichen aus drei großen Teilberichten zusammen, die auf einander aufbauen. Zunächst wurde die Anforderungsdefinition erstellt, in der das System zum Einen beschrieben wird und zum Anderen die Anforderungen an dieses formuliert werden. Darauf aufbauend entstand das Architektur und Schnittstellen Dokument. Dort wurde die verwendete Hardware evaluiert und die Schnittstellen zwischen den Subsystemen dokumentiert. Darüber hinaus wurden die ersten Prototypen vorgestellt.
Der dritte Bericht beschreibt die Entwicklung des Systems und den aktuellen Stand. Zusätzlich wurden zu diesem Endbericht ein *ScrVm* Dokument, welches die Sprintplanung und Umsetzung dokumentiert, und ein Testbericht angefertigt. Der Testbericht enthält Informationen zum Testvorgehen und die Ergebnisse der durchgeführten Tests. Die entwickelte Software wurde mittels Doxygen dokumentiert.
Der Endbericht ist folgendermaßen aufgebaut. Nach dieser Einleitung werden im folgenden Kapitel 2 zunächst einige bekannte Referenzprojekte vorgestellt, welche das autonome Rennfahren mit RC-Cars behandeln und somit grundlegendes Vorwissen für dieses Projekt bereitstellen. Das nachfolgende Kapitel 3 beschreibt das Projektmanagement. Daraufhin werden im Kapitel 4 die Systemkomponenten vorgestellt. Auf der Systembeschreibung baut im Anschluss das Kapitel 5, welches die gestellten Anforderungen an das System beinhaltet, auf. Kapitel 6 beschreibt die Anwendungsfälle.
Im Anschluss wird in Kapitel 7 die Hardware evaluiert und im Kapitel 8 die Schnittstellen zwischen den Subsystemen definiert. Diesbezüglich wird im nachfolgenden Kapitel 9 das Sicherheitskonzept erläutert.
Die Kapitel 10 und 11 beschreiben die Softwareentwicklung und die Evaluation der Latenzen. Danach werden in Kapitel 12 die durchgeführten Tests erläutert. Das Dokument endet mit dem Kapitel 13. Dort wird unter anderem ein Ausblick auf eine mögliche Weiterentwicklung des Systems gegeben.

Häufig verwendete Fachbegriffe und Abkürzungen werden im Glossar aufgeführt. Produkt, Firmen- und Projektnamen werden *kursiv* gedruckt, um diese optisch hervorzuheben und dadurch die Lesbarkeit des Textes zu verbessern.

2 Autonomous Racing

Das Ziel dieses Kapitels ist es die grundlegenden Informationen zusammenzutragen, welche für eine effiziente Durchführung des Projektes *RCCARS* benötigt werden. Die Inhalte des Kapitels entsprechen den Tätigkeiten, welche in den ersten Wochen der Projektgruppe durchgeführt wurden.
Der erste Abschnitt ”State of the Art” zeigt die Ergebnisse der Recherchen zu den Arbeiten anderer Universitäten und Forschungseinrichtungen, welche bereits Projekte zum autonomen Rennfahren mit RC-Cars durchgeführt haben. Auf der Grundlage dieses Vorwissens werden schließlich die Ziele des eigenen Projektes bestimmt. Zum Einen wird die langfristige Zielsetzung des Projektes definiert und zum Anderen die Aufgabenstellung der ersten Ausbaustufe festgelegt und durch ein Szenario veranschaulicht.

2.1 State of the Art

Bei der Realisierung des eigenen Ansatzes orientiert sich die Projektgruppe *RCCARS* an Vorarbeiten und Erfahrungen anderer Universitäten und Forschungseinrichtungen. Diese Projekte werden im Folgenden vorgestellt.

2.1.1 Eidgenössische Technische Hochschule Zürich

Die ETH Zürich bietet aufgrund der jahrelangen und umfassenden Forschung im Bereich *Autonomous Racing System* fortgeschrittene Ergebnisse. Das Real Time Racing System wurde von mehreren Projektgruppen in einem Zeitraum von sechs Semestern stetig weiter entwickelt. Darüber hinaus wurden mehrere Abschlussarbeiten diesbezüglich verfasst.
Die im Jahr 2010 von Patrick Spengler und Christoph Gammeter verfasste Arbeit mit dem Titel *Modeling of 1:43 scale race cars* [SG10], befasst sich mit der Entwicklung eines geeigneten Fahrzeugmodells und entsprechenden Kontrollalgorithmen. Inhaltlich knüpfen die Studenten an die Resultate einer Abschlussarbeit aus dem Vorjahr an, in welcher sich die Studenten Marc Osswald und Florian Engeler mit dem Entwurf und der Implementierung eines *autonomous real-time RC car racing systems* [OE09] beschäftigt haben. Obwohl der entwickelte PID-Regler nach genügend Feinabstimmung akzeptable

Ergebnisse lieferte, war eines der größten Probleme in der Entwicklung das Fehlen eines geeigneten Fahrzeugmodells.
Als Setup verwendeten Spengler und Gammeter zwei monochrome *Grasshopper 03K2C* Kameras, die jeweils mit einem Infrarot-LED-Array und einem Infrarotfilter ausgerüstet wurden. Dadurch konnte eine Auflösung von 5 mm erzielt werden. Mittels eines Kabels des Typs *FireWire 800* wurden die Daten an einen Computer übermittelt. Limitiert durch das FireWire-Kabel kann entweder eine Fläche von $2{,}0 \cdot 3{,}0$ m mit einer Bildübertragungsrate von 200 Hz oder $4{,}0 \cdot 3{,}0$ m mit 100 Hz observiert werden. Die Dächer der Karosserie der RC-Cars wurden mit individuellen Zuschnitten aus Reflektorfolie versehen. Der Aufbau orientiert sich an den Ergebnissen der Arbeit *Infrared based vision system for tracking 1:43 scale race cars*[Rut10] von Martin Rutschmann.

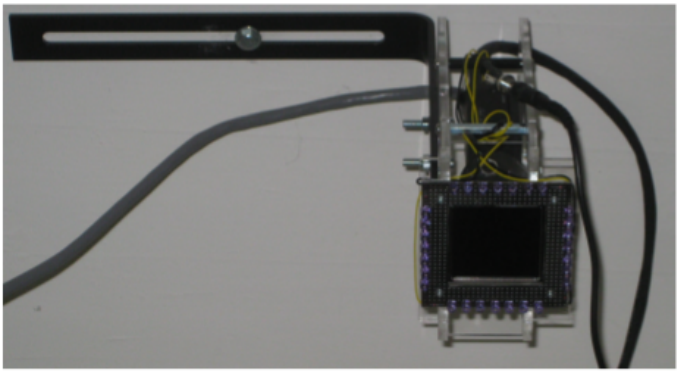


Abbildung 2.1: Von Spengler und Gammeter verwendetes Kamerasetup [SG10]

Das in C++ implementierte Interface wurde so simpel wie möglich gehalten und erlaubt dem Nutzer die Steuerung von bis zu neun Fahrzeugen der *dNaNo*-Plattform über jeweils eine Tastatur oder über einen Controller einer *Sony PlayStation*. Außerdem konnte das Interface Daten aufzeichnen und speichern. Die Fahrzeugmodelle wurden mit *Matlab Simulink* [Mat] entwickelt.
In dem Paper *Optimization-based autonomous racing of 1:43 scale RC cars* [LDM13] von Alexander Liniger, Alexander Domahidi und Manfred Morari aus dem Jahr 2013, welches sich inhaltlich mit der mathematischen Optimierung des Fahrverhaltens der RC-Cars befasst, geht hervor, dass sich die Bilderfassungskomponenten im Laufe der Entwicklung verändert haben. Anstelle der zwei oben erwähnten Kameras, wird eine einzige *Point Grey Flea 3* Kamera verwendet. Außerdem wurde die Steuerung der Fahrzeuge optimiert. Anstatt wie zuvor die RC-Cars mittels eines Digital-Analog-Konverters (DAC) über die Standardfernsteuerung zu bedienen, werden die Fahrzeuge nun über Bluetooth gesteuert. Da die originale Leiterplatte der *dNaNo*-Fahrzeuge keine Kommunikation per *Bluetooth* unterstützt, wurde diese durch eine eigens entwickelte Leiterplatte ersetzt. Die Leiterplatte verfügt über einen *Bluetooth* Chip, Spannungssensoren, H-Bridges für die

Motorsteuerung und einen *ARM Cortext M4* Mikrocontroller.
Nach dem aktuellen Stand ist das System soweit ausgereift, dass mehrere Fahrzeuge mit einer Geschwindigkeit von 3 m/s auf den Geraden autonom über die etwa 19 m lange Rennstrecke (ausgehend von der Mittellinie der Bahn) gesteuert werden können. Dabei können die Fahrzeuge Kollisionen mit stehenden und fahrenden Fahrzeugen vermeiden und in den Kurven driften [LDM13]. Außerdem kann die Rennstrecke auch erfolgreich rückwärts durchfahren werden.

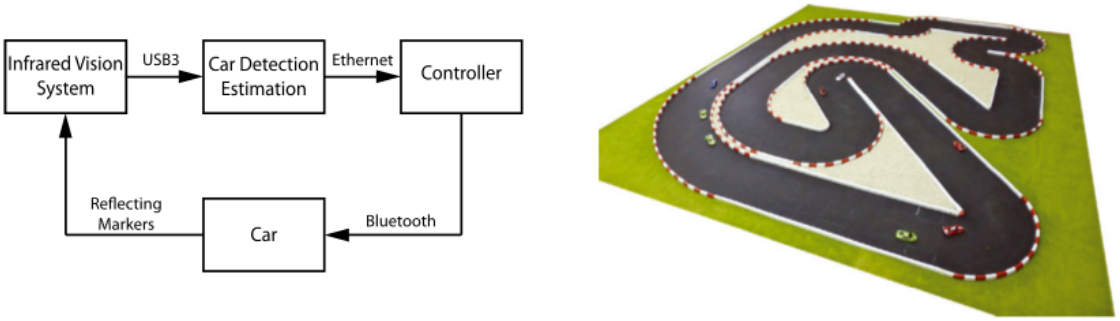
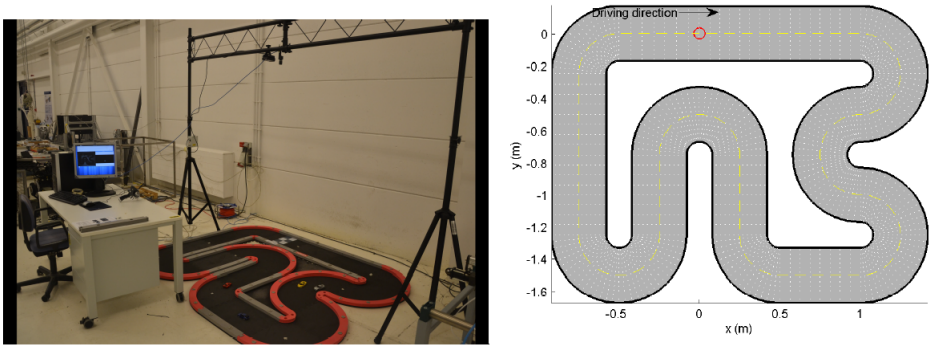


Abbildung 2.2: Signalfluss und Rennstreckenverlauf der ETH Zürich [LDM13]

2.1.2 Katholische Universität Leuven

Die KU Leuven präsentierte ihre Forschungsarbeiten im Bereich der modellbasierten RC-Cars unter anderem in einer Masterarbeit von Robin Verschueren [Ver14], aus welcher die folgenden Informationen gezogen wurden. Diese Arbeit entstand in einer Kooperation mit *LMS (Siemens)* und sollte dort später zu Demonstrationszwecken dienen. Das primäre angesetzte Ziel ist ein zeitoptimiertes Fahren mit RC-Cars auf einer Rennstrecke, unter Verwendung von Model Predictive Control (MPC).
Als Setup wurde eine Rennstrecke mit einer Minimum Bounding Box von 2,5 · 2,0 m verwendet, welche in Abbildung 2.3 dargestellt ist. Die Fahrzeuge wurden mit Reflektorfolie versehen und mit Hilfe einer Infrarotleuchte und einer 2 m über dem Boden befestigten *Point Grey Flea 3* Kamera detektiert. Diese Kamera arbeitet mit einer Frequenz von 100 Hz und bietet, laut Robin Verschueren, die beste Mischung aus Genauigkeit und Leistung. Die aufgenommenen Daten werden über eine USB 3.0 Schnittstelle an einen Rechner weitergeleitet.
Insgesamt wurden zwei Rechner genutzt. Der erste Rechner verarbeitet den Zustand der detektierten Fahrzeuge anhand der aufgenommenen Bilder (state estimation Rechner) und sendet diese Daten an den zweiten Rechner, den Kontrollrechner (Controller), welcher die Fahrzeuge im Anschluss ansteuert. Die Rechner (*Intel Core i3-3220 @ 3,3 GHz*) arbeiten mit einem *Debian Linux 7.0 (Wheezy)* Realzeitbetriebssystem (Kernel: *Linux 3.8.13 with*

Preempt RT Patch). Als Software wurde unter anderem *ACADO* [Aca] eingesetzt. Dies ist ein Open Source Optimierungstool, welches Schnittstellen zu *Matlab* bietet und ein Code Generation Tool besitzt. Neben *ACADO* wurde *Matlab* sowie eigen geschriebener C++ Code verwendet.



(a) The experimental setup at LMS.

(b) The test track used to experiment with the car control algorithm. The dimensions are approximately 2.5 m by 2 m

Abbildung 2.3: Rennstreckenverlauf der KU Leuven [Ver14]

Das entwickelte System wurde im Rahmen einer weiteren Masterarbeit am Polytechnikum Mailand weiter ausgebaut und optimiert [Fon14].

2.1.3 Universität Uppsala

Das Projekt *CARS* (Camera-based Autonomous Racing System) [CAR14] wurde an der Universität Uppsala entwickelt. Das Ziel war es eine Plattform zu schaffen, die Studenten die Möglichkeit bietet ihre theoretischen Kenntnisse im Bereich Systems and Control praktisch anzuwenden. Außerdem wird das Projekt genutzt, um aktuelle Forschungsergebnisse aus diesem Bereich der Öffentlichkeit zu präsentieren.
Seit Projektbeginn im Jahr 2014 hat das Projekt zwei Entwicklungsphasen durchlaufen, die sich über einen Zeitraum von je einem halben Jahr erstreckten. Während der ersten Phase hatte eine Gruppe von fünf Studenten zunächst die Aufgabe den Hardwareaufbau des Systems zu realisieren und eine erste einfache Regelungssoftware für die autonome Steuerung eines einzelnen Fahrzeugs zu implementieren. Zum Abschluss dieser Phase war das System bereits in der Lage, die beste Rundenzeit eines menschlichen Fahrers zu unterbieten. In der anschließenden zweiten Entwicklungsphase wurde das Projekt von vier anderen Studierenden übernommen. Diese hatten die Aufgabe, die Regelungsalgorithmen der Fahrzeugsteuerung zu optimieren, um eine schnellere und präzisiere autonome Steuerung zu erreichen. Im Weiteren sollten zwei Fahrzeuge gleichzeitig autonom gesteuert werden.

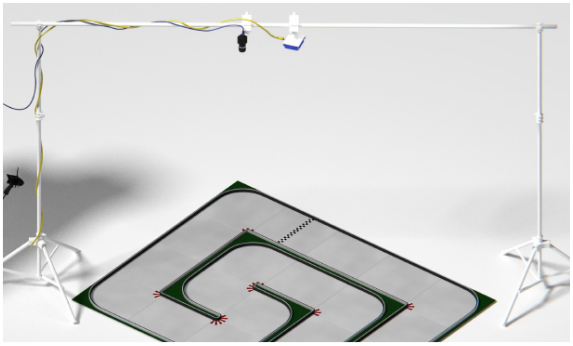


Abbildung 2.4: Aufbau der Rennstrecke und der Bildererfassungskomponenten im Projekt *CARS*. [CAR14]

Bei dem Systemaufbau der Universität Uppsala werden ferngesteuerte Fahrzeuge der *dNaNo*-Plattform [KYO] verwendet, deren Position mit Hilfe von Infrarotlicht und entsprechender Reflektorfolie erfasst wird. Bei der Bilderfassung kommt die *Point Grey Flea 3* Kamera zum Einsatz, welche mit einem Weitwinkelobjektiv und einem Infrarotlichtfilter ausgestattet ist. Als Infrarotlichtquelle wird die Lampe *Smart Vision Lights S75* genutzt. Die Kamera und IR-Leuchte wurden wiederum an einem Gerüst befestigt und 2,3 m über dem Zentrum der eigens konstruierten Rennstrecke positioniert (siehe Abbildung 2.4). Die Verarbeitung der Kamerabilder und die Berechnung der Regelungsparameter für die Fahrzeugsteuerung sowie die Darstellung einer grafischen Benutzeroberfläche werden gemeinsam auf einem Rechner in drei separaten Threads durchgeführt. Die Kommunikation zwischen Rechner und Fahrzeug wurde mit Hilfe einer DA-Wandlerkarte des Typs *NI-DAQ PCIe 6321* realisiert, deren analoge Ausgaben an eine modifizierte *dNaNo*-Fernbedienung weitergeleitet werden, welche die Signale wiederum an das Fahrzeug übermittelt. Als zusätzliche Visualisierungskomponente besitzt das System einen Projektor, der Systeminformationen direkt auf die Rennstrecke projektieren kann. Die Software wurde in C++ implementiert. Dabei wurde u. a. die Bibliothek *OpenCV* für die Bildverarbeitung und die *FlyCapture2* API der o. g. Kamera verwendet. Einzelne Hilfsfunktionen wurden mit *Matlab* implementiert.

2.1.4 Universität Göteborg

Unter dem Titel *Advanced vehicle control systems* [EGH+15] veröffentlichten sechs Studenten der University of Goethenburg im Jahre 2015 ihre Bachelorarbeit. Der Ursprung dieser Arbeit entstand im Frühjahr 2013 in Form eines Projektes zur Entwicklung einer Plattform zum Testen verschiedener Regelungsmethoden für Schwarmroboter. Im Herbst 2014 wurden diese gegen realistischere RC-Cars ersetzt. Die Aufgaben bestand in der Entwicklung einer verbesserten Plattform, die Erhöhung der Fahrzeuggeschwindigkeit

und einer modularen Struktur für die Software. Als Ziel hatte das Team eine Fahrzeuggeschwindigkeit von 2 m/s festgelegt. Die Systemkomponenten der Positionsbestimmung sollten dabei keine Auswirkungen auf die Leistung der Fahrzeuge haben und die Kommunikation zwischen dem Computer und dem Sender sollte zuverlässig sein. Der Quellcode sowie neue Fahrzeug-Algorithmen sollten einfach und gut kommentiert sein, so dass eine Weiterentwicklung möglich ist. Das System sollte außerdem fähig sein, Objekte zu erkennen, um sowohl Kollisionen zu vermeiden, als auch Überholmanöver durchführen zu können.
Als Fahrzeuge wurden *Kyosho dNaNo* RC-Cars im Maßstab 1:43 eingesetzt. Diese wurden mit Reflektorfolie versehen und mit einer eigens entwickelten Infrarotleuchte bestrahlt. Die IR-Leuchte setzt sich hierbei aus einem *ILS OSLON 4 IR Wide PowerStar IR-LED-Array* [IR] mit einer Lichtwellenlänge von 850 nm, einem passenden Kühlkörper und einem Netzteil zusammen. Eine *Point Grey Flea 3* Kamera mit *GOYO GM25414MCN* Objektiv und IR-Filter filmt die Fahrzeuge von oben und leitet die Daten über eine USB 3.0 Schnittstelle an einen Rechner weiter. Der Rechner führt die Bildverarbeitung und Berechnung der Fahrzeugregelung durch. Die Kommunikation zwischen dem Rechner und dem RC-Car wurde über eine eigens entwickelte Platine realisiert, welche einen *Texas Instruments TLC5620CN* Digital-Analog-Konverter (DAC) enthält und die Verdrahtung zweier *Kyosho Perfex KT-18* [Fer] Fernsteuerungen mit einem Entwicklerboard des Typs *Arduino Uno* ermöglicht. Diese Komponenten sind, wie in Abbildung 2.5 dargestellt, auf einer Kunststoffhalterung befestigt. Mit dieser Konstruktion ist die parallele Ansteuerung zweier RC-Cars möglich.

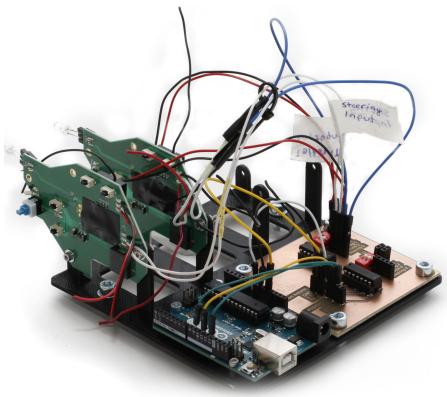


Abbildung 2.5: Kunststoffhalterung mit Arduino Uno, DAC-Platine und zwei *dNaNo*-Fernsteuerungen [EGH+15]

Zur Simulation der Regelung wurde *Matlab* mit der *Simulink* Toolbox verwendet. Die Programmierung erfolgte in der Programmiersprache C# unter Einsatz der Entwicklungsumgebung *Visual Studio* und in C mit der *Arduino IDE*. Für die Objekterkennung wurden

die Bibliotheken *FlyCapture SDK* und *AForge.NET* verwendet.
Das Team von sechs Personen erreichte eine präzise Kommunikation zwischen den Fahrzeugen und dem Computer und die als Ziel gesetzte Geschwindigkeit von 2 m/s. Des Weiteren sind Algorithmen zur Kollisionsvermeidung und zum Überholen (jedoch unzuverlässig bei beweglichen Hindernissen) und zum Platooning (ebenfalls noch unzuverlässig) entstanden. Schwierigkeiten ergaben sich bei der Objekterkennung, da die selbst entwickelte IR-Leuchte die Rennstrecke nicht gleichmäßig ausleuchten konnte. Dadurch traten Störungen bei der Erkennung der Fahrzeuge auf.

2.2 Long Term Vision

Langfristiges Ziel des Projekts *RCCARS* ist die Realisierung eines Testfelds für die Entwicklung und Analyse von Strategien im Bereich des autonomen Fahrens. Dies beinhaltet einerseits realzeitfähige Regelungsansätze für die Längs- und Querführung von Fahrzeugen, beispielsweise aus dem Bereich Model Predictive Control (MPC), und andererseits Konzepte aus den Bereichen der Spieltheorie sowie der Car2X-Kommunikation zur Umsetzung von (kooperativen) Fahrmanövern. Durch die Verwendung eines transportablen Setups ist es des Weiteren angedacht, das Testfeld als Demonstrator auf Messen und Projektreviews verwenden zu können.

Da das Projekt an der Carl von Ossietzky Universität zukünftig federführend durch studentische Projektgruppen sowie Abschlussarbeiten vorangetrieben werden soll, ergibt sich auf Grundlage des State of the Art (siehe Abschnitt 2.1) und den Erfahrungen der beteiligten Universitäten / Forschungseinrichtungen für die Durchführung des Projekts *RCCARS* folgender mehrstufiger Prozess (siehe Abbildung 2.6).

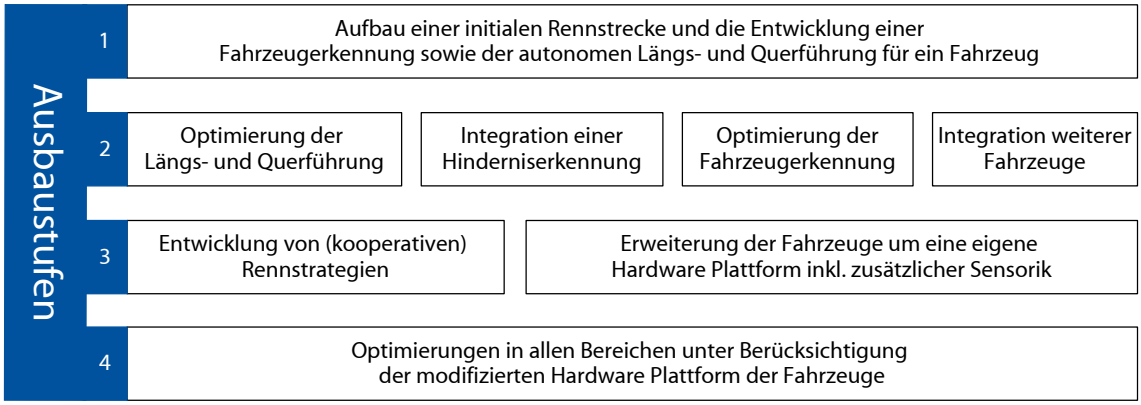


Abbildung 2.6: *RCCARS* Ausbaustufen

Initial ist ein minimales Setup zu entwerfen, welches die Grundlage für weitere Ausbau-

stufen bildet. Dazu gehören eine Rennstrecke, die Fahrzeugerkennung sowie die autonome Steuerung eines Fahrzeuges. Darauf aufbauend können in der zweiten Ausbaustufe Optimierungen und Erweiterungen vorgenommen werden. Nach der Integration von mehreren Fahrzeugen können in der dritten Ausbaustufe (kooperative) Rennstrategien entwickelt werden. Ein Austausch der Fahrzeugelektronik durch eine eigen entwickelte Hardwareplattform inklusive zusätzlicher Sensorik (vgl. 2.1.1) erlaubt die Entwicklung präziserer Regelungs- und Steuerungsalgorithmen. In der vierten Ausbaustufe gilt es anschließend alle Systemkomponenten im Hinblick auf kooperative Rennstrategien zu optimieren.

2.3 Aufgabenstellung der Projektgruppe RCCARS

Die Aufgabe der Projektgruppe besteht zunächst darin die erste Ausbaustufe (siehe Abbildung 2.6) zu realisieren. Dies beinhaltet neben der Anschaffung der notwendigen Hardware, dem Aufbau einer Rennstrecke, der Implementierung einer Fahrzeugerkennung und einer autonomen Fahrzeugsteuerung, insbesondere die Planung und den Entwurf einer modularen Systemarchitektur, welche die Basis der weiteren Ausbaustufen bildet.
Grundlage der Entwicklungsarbeit sind dabei die Erfahrungen, welche von anderen Universitäten und Forschungseinrichtungen bei der Realisierung ihrer Systeme gewonnen wurden. Um eine gewisse Standortübergreifende Vergleichbarkeit zu ermöglichen gibt es von Seiten der Betreuer die Vorgabe ebenfalls Rennfahrzeuge der Marke *Kyosho* vom Typ *dNaNo* [KYO] zu verwenden.
Wesentliche Bestandteile der Umsetzung sind somit ferngesteuerte Fahrzeuge, eine geschlossen Rennstrecke sowie Hard- und Softwarekomponenten für die lokale Positionsbestimmung bzw. für die autonome Steuerung der Fahrzeuge.
Bei der Entwicklung der Regelungssoftware, welche für die autonome Fahrzeugsteuerung benötigt wird, steht das Softwarewerkzeug *SCADE* [SCA] zur Verfügung. *SCADE* ermöglicht die modellbasierte Entwicklung sicherheitskritischer Software, inklusive Simulation und Codegenerierung für verschiedene Zielplattformen.
Im folgenden Abschnitt wird das Anwendungsszenario beschrieben, welches im Rahmen der ersten Ausbaustufe umgesetzt werden soll. Dieses ist als Minimalziel der Projektgruppe anzusehen.

2.4 Szenario: Unfallfreie Fahrt

Das Szenario unfallfreie Fahrt sieht vor, dass ein einzelnes Fahrzeug auf einer geschlossenen Rennstrecke autonom mit einer Durchschnittsgeschwindigkeit von 1,5 m/s mindes-

tens fünf Runden ohne eine Kollision mit der Fahrbahnbegrenzung fährt. Die in Abbil-

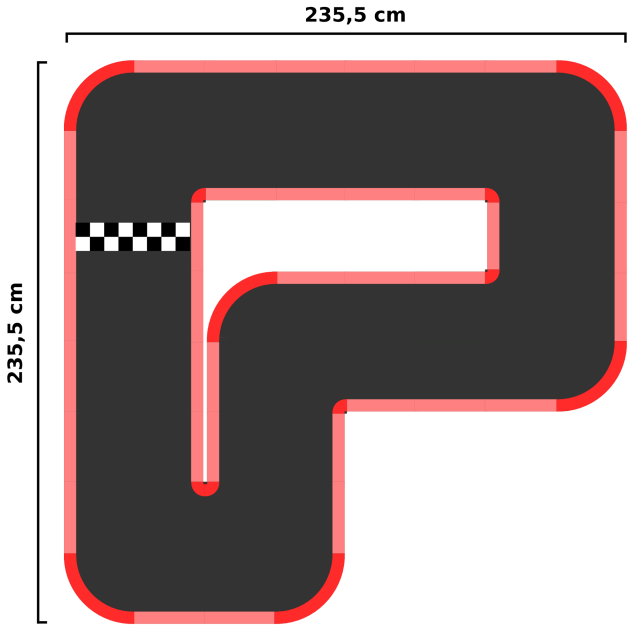


Abbildung 2.7: Schematische Darstellung der im Projekt *RCCARS* verwendeten Rennstrecke mit Maßangaben.

dung 2.7 illustrierte Rennstrecke befindet sich in einem geschlossenen Raum, welcher diese vor äußeren Umwelteinflüssen wie Wind und Wetter schützt. Dadurch wird gewährleistet, dass die Fahrbahn zu jedem Zeitpunkt eben und trocken ist. Des Weiteren werden Fahrbahn und Fahrzeug regelmäßig von Staub befreit, um dem Fahrzeug eine optimale Bodenhaftung zu bieten. Vor jedem Rennen wird daher von einem Administrator geprüft, ob das Fahrzeug auf der Fahrbahn starten kann. Ferner ist sicher zu stellen, dass das Fahrzeug unbeschädigt und die Verbindung mit der Fernsteuerung intakt ist.
Wie auch bei realen Autorennen werden die Zuschauer gebeten einen Mindestabstand zur Rennstrecke während einer Fahrt zu wahren, um weder die Sensorik des Systems, noch den Zustand der Fahrbahn zu beeinträchtigen.

Sind alle Sicherheitsvorschriften eingehalten, kann der Rennbetrieb freigegeben werden. Über eine externe Kontrolleinheit kann der Administrator den Startbefehl geben, so dass sich das Fahrzeug in Bewegung setzt. Hierbei sei angemerkt, dass das Fahrzeug ausschließlich von der autonomen Steuerungseinheit kontrolliert wird; ein manuelles Eingreifen durch Personen in das Fahrgeschehen ist, außerhalb eines sicherheitskritischen Fehlerfalls in dem das Rennen mittels eines Not-Aus abgebrochen werden muss, nicht vorgesehen.
Mangels GPS wird das Rennen durch eine Kamera oberhalb der Rennstrecke überwacht, welche in Realzeit Bilder vom Fahrzeug und der Rennstrecke an die Positionserkennung

übermittelt. Die Positionserkennung errechnet die Pose des Fahrzeugs und übermittelt diese an die Steuerungseinheit. Diese entscheidet auf Grundlage der vorliegenden Daten welches Fahrmanöver angemessen ist und sendet entsprechende Steuerbefehle an das Fahrzeug.

Das Fahrzeug wird durch die Steuerungseinheit mit einer Durchschnittsgeschwindigkeit von 1,5 m/s über die Rennstrecke gelenkt. Dies würde im Maßstab 1:1 einer Geschwindigkeit von ca. 230 km/h entsprechen. Das Szenario gilt als erfolgreich abgeschlossen, wenn das Fahrzeug die oben besagte Durchschnittsgeschwindigkeit erreicht, die Fahrbahn der Rennstrecke mindestens fünf mal unmittelbar hintereinander umrundet und dabei nicht mit der Fahrbahnbegrenzung kollidiert.

3 Projektmanagement

Dieses Kapitel befasst sich mit dem Projektmanagement dieser Projektgruppe. Dazu zählt zu Beginn die Vorstellung von einigen weit verbreiteten und häufig verwendeten Vorgehensmodellen worauf im Anschluss das in diesem Projekt verwendete Vorgehensmodell - welches den Namen *ScrVm* zugewiesen bekommen hat - aufgebaut wird. Im Weiteren wird auf das Kosten- sowie Risikomanagement eingegangen und die in diesem Projekt verwendeten Softwarewerkzeuge vorgestellt.

3.1 Vorgehensmodell

Bei der Entwicklung von Softwaresystemen steht zu Beginn jedes Projekts das Ziel, ein lauffähiges System - aus diversen zuvor erhobenen Anforderungen - umzusetzen. Um dieses Ziel erreichen zu können, muss stets ein gut organisierter Plan zur Verfügung stehen. Dieser Projektplan gliedert das Gesamtziel in kleinere Teilziele und soll so das Risiko eines Scheitern des Projekts minimieren beziehungsweise die Erfolgswahrscheinlichkeit deutlich erhöhen. Probleme die dieser Plan lösen muss sind beispielsweise die Abarbeitung großer Mengen an Anforderungen. Dadurch bedingt, dass Anforderungen zum Teil vom Auftraggeber als auch vom Auftragnehmer abgeleitet werden ist darauf zu achten, dass die festgehaltenen Anforderungen keine Unklarheiten enthalten.
Der zuvor genannte Plan lässt sich durch Vorgehensmodelle (VM) realisieren. Die Vorteile eines VM liegen neben den zuvor genannten Punkten unter anderem in einer projektbegleitenden Dokumentation, einem strukturierten und nachvollziehbaren Projektablauf, der Ressourcenplanung und in der frühzeitigen Fehlererkennung. Es gibt eine Vielzahl an vorgefertigten und ausgearbeiteten VM am Markt. Zwei aktuelle und häufig verwendete VM werden im folgenden Abschnitt aufgeführt woraus sich in dem daran anschließenden Abschnitt 3.1.2 das für dieses Projekt eigens konstruierte VM zusammensetzt - das sogenannte *ScrVm* Modell.

3.1.1 V-Modell und Scrum

V-Modell

Das V-Modell basiert auf dem am weitesten verbreiteten Wasserfallmodell [MB09] und erweitert dieses um jeweils nachstehende Testphasen (Verifikation und Validierung). Nach dem Buch *Das V-Modell XT* [Hö08] besteht die Zielsetzung des V-Modells hauptsächlich darin die Projektrisiken durch standardisierte Vorgehensweisen zu minimieren und so die Planbarkeit des Projekts zu verbessern und Planungsabweichungen sowie Risiken frühzeitig zu erkennen. Zusätzlich bietet das V-Modell den Vorteil, dass Produkte im Mittelpunkt stehen und so eine Ziel- und Ergebnisorientierte Entwicklung entsteht. Das V-Modell wird nebenher für alle Bereiche der Bundesbehörden (Abweichungen nur mit Begründung erlaubt) eingesetzt und wird von der Bundesverwaltung insbesondere für kleine bis mittelständige Unternehmen empfohlen [Inn16].
Ein besonderer projektbezogener Vorteil an diesem Modell ist das kontinuierliche entwerfen von Testfällen, welche das Modell am Ende der Entwicklungsphase bestehen muss. Dadurch zeichnet sich das Modell besonders für sicherheitskritische Projekte aus. Zuletzt sei auch noch der im Vordergrund stehende projektbezogene Nachteil aufgeführt. Das V-Modell sieht keine Iterationen vor und stellt so ein nicht-agiles Verfahren dar. Dies ist an dieser Stelle jedoch sehr wichtig, denn die Anforderungen des Projekt verändern/erweitern sich zur Laufzeit dynamisch.

Die Grafik 3.1 beschreibt das V-Modell genauer und basiert auf der Spezifikation und Zerlegung des Systems mit anschließender Realisierung und Integration der Lösungen. Zu Beginn werden bei diesem VM - wie auch bei dem Wasserfallmodell - die **Anforderungen** der Anwender und der Entwickler erhoben, die das System letztendlich erfüllen muss und durch gegebenenfalls passende Beschreibungen ergänzt, um die Anforderungsdefinition zu vervollständigen. Parallel dazu werden kontinuierlich Testfälle entwickelt, die später in der Realisierungs- und Integrationsphase validiert werden. Als nächstes folgt die Spezifikation des Systems - auch **Produktentwurf** genannt - wobei es um die Erstellung einer Softwarearchitektur geht. Parallel hierzu erfolgt die Spezifikation von Abnahmetests (Akzeptanztests/Systemtests), welche das System in der aufsteigenden Phase des V-Modells bestehen muss. Nach der Grafik 3.1 von Reinhard Höhn [Inn16] folgt darauf der Punkt „System entworfen“, welcher einem **Komponentenentwurf** entspricht. Hierbei wird das zu entwickelnde System in kleinere Teilkomponenten zerlegt (Hardware/Software) und entsprechende einzelne Komponententestfälle (Integrationstests) aufgebaut. Im Anschluss an den Komponentenentwurf folgt der **Feinentwurf**, auch Modulentwurf genannt. Dabei wird letztendlich die Software (und gegebenenfalls die Hardware) realisiert

und im Anschluss einzeln getestet. Nachdem die absteigende Phase des V-Modells beendet wurde folgt das Testen der einzelnen Entwicklungsschritte anhand der jeweiligen Tests, die zuvor aufgeführt wurden.

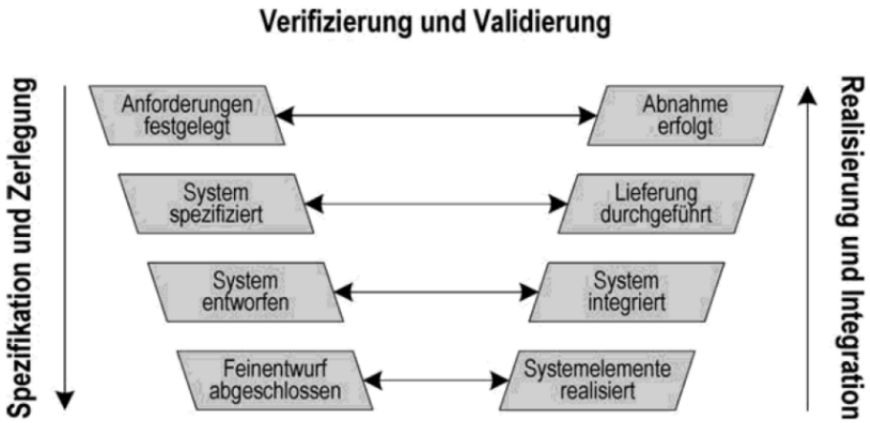


Abbildung 3.1: Das V-Modell nach [Hö08]

Scrum

Das Scrum-Modell [Scra] basiert auf drei grundlegenden Beteiligten. Zum Einen ist dies der Scrum-Master, der - ähnlich wie ein Projektleiter - für organisierende Aufgaben und das einwandfreie Arbeitsumfeld für das Team zuständig ist, der Auftraggeber sowie das Team. Wie in Abbildung 3.2 dargestellt, wird auch im Scrum-Modell - ähnlich wie im V-Modell - mit der Erhebung der Anforderungen begonnen. Die gesammelten Anforderungen werden in einem sogenannten **Product Backlog** erfasst. Sind alle zum aktuellen Zeitpunkt des Projekts bekannten Anforderungen erhoben, so wird vom Team anhand einer Aufwandsabschätzung eine Anzahl von Anforderung aus dem Product Backlog gezogen woraus sich wiederum der **Sprint Backlog** ergibt. Das Team verpflichtet sich diesen Sprint Backlog innerhalb eines **Sprints** zu realisieren. Üblicherweise dauert ein Sprint bis zu 30 Tagen [Scra]. Neben dem Sprint gibt es ein **Daily Scrum Meeting**, in dem der aktuelle Fortschritt besprochen wird. Am Ende des Sprints gibt es ein **Sprint Review** in dem gegebenenfalls aus dem bisherigen Prozess neu angefallene Anforderungen in das Product Backlog aufgenommen werden oder bereits enthaltene Anforderungen angepasst werden können. Nach jedem Sprint ergibt sich ein eigenes Produktinkrement, an dem nicht weiter gearbeitet werden muss. Aus einer Anzahl von Produktinkrementen entsteht im Endeffekt das Endprodukt.

Abbildung 3.2: Das Scrum-Modell nach [Scrb]

3.1.2 ScrVm

Nachdem die Vorzüge und Nachteile der einzelnen etablierten Vorgehensweisen evaluiert wurden, ist der Entschluss gefallen, dass weder Scrum, noch das V-Modell in Frage kommen können. Scrum hat den Vorteil, dass es ein leichtes Projektmanagement Framework mit wenigen starren Regeln ist, jedoch eignet es sich nur zur Entwicklung von Software. Das V-Modell hingeben ist für eine kleine Projektgruppe mit zu vielen administrativen Aufgaben verbunden und fällt daher in Reinform ebenfalls weg. Daher werden die am Besten geeigneten Eigenschaften beider Modell zu einem neuen Vorgehensmodell vereint: ***ScrVm***.
ScrVm vereinigt das agile und iterative Projektmanagement des Scrum-Modells mit den hohen Qualitätsstandards des V-Modells.
Wie auch bei Scrum wird zunächst ein Product Backlog erstellt und abhängig vom aktuellen Stand im Anschluss ein Sprint Backlog. In Form eines Sprints werden die geordneten Anforderungen in immer kleineren Schritten abgearbeitet. Nachdem die letzten Module erstellt wurden, werden Testfälle formuliert. Die untere Pyramide in ScrVm Modell verdeutlicht den Vorgang des Testens. Ausgehend von den kleinsten Einheiten innerhalb des Systems umfassen die Tests fortschreitend immer größere Komponenten. Insgesamt erinnert das Vorgehen an dieser Stelle stark an das V-Modell. Innerhalb des Sprints wird, im Gegensatz zum Original nicht täglich, sondern nur wöchentlich ein Treffen stattfinden. In diesen Treffen besprechen die Teammitgliedern den aktuellen Stand und geben sich gegenseitig Feedback um gegebenenfalls noch produktiver werden zu können.
Sobald der Sprint Backlog abgearbeitet ist, erfolgt ein Abnahmetest und eine Evaluierung. Dabei wird überprüft ob das Inkrement den im Sprint Backlog enthaltenen Anforderungen entspricht und Testfälle erfolgreich bestanden wurden. Außerdem bewertet die Projektgruppe den Verlauf und die Resultate der Iteration und äußert Kritik und Anregungen

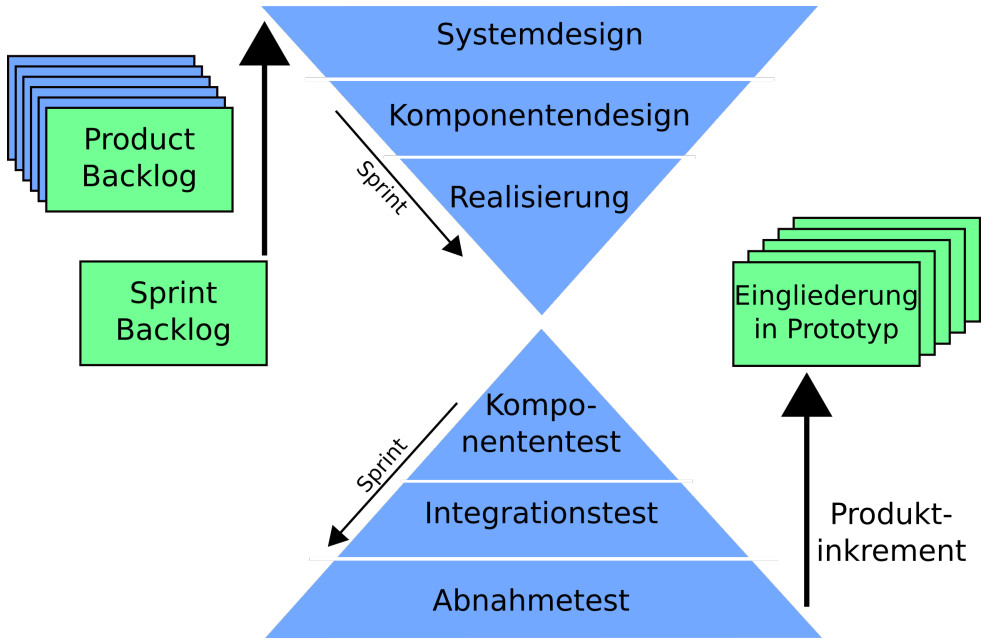


Abbildung 3.3: Das ScrVm-Vorgehensmodell

für den nächsten Sprint. Dadurch sollen die Arbeitsweisen und Methodiken langfristig verbessert werden um zukünftig noch effektiver als Team arbeiten zu können. U.a. soll so sichergestellt werden, dass im Falle einer versäumten Deadline zukünftige Termine eingehalten werden können.
Im letzten Schritt wird das fertige Inkrement in den Prototypen eingegliedert, welcher von Sprint zu Sprint iterativ verbessert und um Funktionen erweitern wird. Um sicherzustellen, dass der Prototyp nach einer Funktionserweiterung immer noch wie vorgesehen funktioniert, erfolgt nach jeder Eingliederung ein weiterer Abnahmetest. Damit diese Testfälle handhabbar bleiben, ist an dieser Stelle Testautomatisierung vorgesehen. Darüber hinaus ist geplant eine Testdatenbank zur Verwaltung der durchgeführten Tests zu verwenden.
Somit endet die Iteration und es wird mit einem neuen Sprint begonnen.

3.2 Kostenmanagement

Die Abteilungen *Hybride Systeme* und *Sicherheitskritische Eingebettete Systeme* der Carl von Ossietzky Universität Oldenburg finanzieren das Projekt *RCCARS* um das in Kapitelabschnitt 2.4 beschriebene Szenario zu realisieren. Dadurch bedingt, dass dieses Projekt zum November 2015 neu gestartet ist, musste ein Großteil der Hardware als Grundausstattung angeschafft werden. Folgeprojekte, welche auf dem Projekt *RCCARS* aufbauen, werden mit dieser Hardware als Grundlage für ihr Projekt starten und reduzieren somit die Folgekosten. Die Tabelle 3.1 zeigt die aktuellen Kosten, die sich durch die Beschaf-

fung diverser benötigter Teile zusammengesetzt haben sowie eine detaillierte Ansicht der genutzten Hardware. Die Preise sind inklusive der gesetzlichen Mehrwertsteuer und belaufen sich aktuell auf X.XXX,XX Euro (Stand 24. Oktober 2016).

Datum	Buchungstext	Betrag in Euro
13.08.15	Erstausstattung Fahrzeuge, Fernbedienung, Rennstrecke	...
14.12.15	Rechner Dell Optiplex 7020 MT	...
18.01.16	Kyosho Sender Perfex KT-18 2-Kanal	...
18.01.16	ILS, OSLON 4 IR Wide PowerStar IR-LED	...
01.05.16	ILS, OSLON 4 IR Wide PowerStar IR-LED	...
18.01.16	Intelligent LED Solutions Kühlkörper	...
01.05.16	Intelligent LED Solutions Kühlkörper	...
18.01.16	ILs Konstantstrom LED-Treiber	...
25.01.16	Flea3 FL3-U3-13S3M-C USB3.0 Kamera	...
25.01.16	USB 3.0 Kabel verschraubbar, 3m Länge	...
27.01.16	Lensagon Objektiv CMFA0420ND	...
27.01.16	MIDOPT FIL LP780/27 Tageslichtfilter	...
28.01.16	3M Reflexstreifen 8850-120	...
04.02.16	Walimex Pro Teleskop Hintergrundsystem	...
04.02.16	Tarion Studioklemme mit Gewinde $\frac{1}{4}$ & $\frac{1}{8}$ Gewinde	...
01.05.16	Tarion Studioklemme mit Gewinde $\frac{1}{4}$ & $\frac{1}{8}$ Gewinde	...
01.05.16	Tarion OS01529 Studioklemme mit Gewinde $\frac{1}{4}$ & $\frac{3}{8}$ Gewinde	...
08.02.16	JJC GM1414 Verbindungsstück	...
10.02.16	Kabel USB3.0 Repeater 5m	...
22.02.16	Einschnittgewindebohrer $\frac{1}{4} \cdot 20$	...
14.12.15	STM32F4 Discovery	...
	Verbrauchsmaterial	...
	Versandkosten insgesamt	...
		...

Tabelle 3.1: Kostenplan

3.3 Werkzeuge

In diesem Kapitelabschnitt werden die verwendeten Softwarewerkzeuge in diesem Projekt aufgezeigt. Im Weiteren wird die Verwendung dieser Programme kurz begründet und die Vorteile dargelegt.

Versionsverwaltung

Zur verteilten Versionsverwaltung wird das Softwaretool GIT [GIT] verwendet. Dieses Tool stellt die Möglichkeit des sogenannten ”branchen” und des ”mergen” bereit, wel-

che dem Erstellen eines Entwicklungszweiges zum parallelen Arbeiten entspricht, der anschließend wieder mit dem Master Zweig vereint wird. Diese Funktionalität wird auch als nicht-lineare Entwicklung bezeichnet und stellt für eine verteilte Entwicklung so eine grundlegende Eigenschaft zur Verfügung.
Im Weiteren bietet GIT den großen Vorteil, dass es Plattform unabhängig eingesetzt werden kann und so Windows wie auch Linux und Mac OS X Anwendern das Arbeiten problemlos ermöglicht. Zusätzlich ist das Tool für alle eben genannten Plattformen kostenfrei zu erwerben.

Projektmanagement

Um das Projekt zu managen wird das Softwaretool Microsoft Project 2016 [MSP16] verwendet. Dieses Tool steht nicht kostenfrei zur Verfügung, bietet dafür jedoch ausgeprägte Vorteile. MS Project ist eine Software die das Planen, Steuern und Überwachen von Terminen in Projekten ermöglicht. Im Weiteren bietet es Möglichkeiten der Netzplantechnik und lässt so das Verketten von bestimmten Aktionen zu. Ein weiterer ausschlaggebender Vorteil dieses Softwaretools ist das visuelle Darstellen der geplanten Aktionen inklusive der Möglichkeit kritische Vorgänge hervorzuheben. Zusätzlich können Meilensteine definiert werden und Aktionen feste Ressourcen - in diesem Fall Projektmitglieder - zugeordnet werden und stellt so insgesamt eine gute Übersichtsmöglichkeit über den Verlauf und die Planung des Projekts dar.
Microsoft Project ist lediglich für Windowsplattformen nutzbar. Es gibt jedoch die Möglichkeit die Planung unter anderem in das PDF Format zu exportieren und ist so beispielsweise auch für Linux und Mac OS X Nutzer einsehbar.
Die Software kann für Studenten im Rahmen von Dreamspark [Dre] kostenfrei für nicht kommerzielle Zwecke genutzt werden.

Softwareentwicklung

Zur modellgetriebenen Softwareentwicklung wird das Tool *SCADE* [SCA] verwendet. *SCADE* ist ein kommerzielles Produkt und wurde der Carl von Ossietzky Universität Oldenburg im Rahmen des *SCADE* Academic Programm [Tec16] kostenfrei zur Verfügung gestellt. Das Werkzeug eignet sich insbesondere für die Entwicklung von sicherheitskritischer Software. Esterel Technologies [Est] gibt an, dass es sich für den Designprozess von beispielsweise Flugkontrollsystemen, Flugzeugfahrwerkssystemen oder Autopiloten im Flugbereich eignet. Im Weiteren wird *SCADE* im Zugbereich in der Steuerung der Schranken und Ampelsignalen verwendet und in der Entwicklung von Fahrerassistenzsystemen im Automotiv Bereich angewandt.

Ein weiterer signifikanter Vorteil ist der integrierte KCG Code Generator, welcher unter der ISO 26262:2011 qualifiziert sowie unter IEC 61508 2010 und EN 50128:2011 zertifiziert ist.
SCADE ist ausschließlich auf Windowsplattformen nutzbar.

Zur weiteren Softwareentwicklung wurde die Entwicklungsumgebung Eclipse [Ecl] benutzt mit dem entsprechenden C-Plugin CDT. Die Systemsteuerungssoftware (siehe Kapitelabschnitt 10.4) wurde ebenfalls in Eclipse entwickelt, jedoch in der Programmiersprache Java (siehe Kapitelabschnitt 10.4). Eclipse ist plattformunabhängig kostenlos nutzbar.

Die CarControl (siehe Kapitelabschnitt 10.3.2) wurde mit dem Softwarewerkzeug CooCox [Coo] entwickelt, da Bibliotheken benötigt wurden, welche ausschließlich in CooCox zur Verfügung standen. CooCox ist kostenfrei nutzbar.

Zur Dokumententation des Quellcodes wurde das Dokumentationstool Doxygen [Dox] verwenden. Doxygen ist ein häufig genutztes Tool in der Softwareentwicklung um entwickelten Quellcode zu dokumentieren und bietet die Möglichkeit einen umfangreichen Dokumentationsbericht zu generieren.
Um den entwickelten Quellcode zu testen wurde das Softwarewerkzeug CUTE [LF15] verwendet, welches sich in das genutzte Entwicklungstool Eclipse integrieren lässt. Sowohl Doxygen als auch CUTE sind Open-Source Programme und somit kostenfrei benutzbar.

Zusätzlich wurde das Terminal-Programm HTERM [Ham] zum Auslesen (virtueller) serieller Schnittstellen verwendet. Das Programm ist kostenfrei und läuft unter Windows sowie Linux.

Bildbearbeitung

Die Erstellung sowie Bearbeitung von Grafiken werden mit InkScape [Ink] vorgenommen. InkScape ist ein Open-Source-Vektorgrafikprogramm, welches kostenfrei zur Verfügung steht und beispielsweise Adobe Illustrator ähnelt. Die Software bietet den Vorteil, dass sie flexibel und mit vielen Dateitypen kompatibel ist. Das Tool ist anwendbar auf Windows, Mac OS X sowie auf Linux Systemen.
Neben InkScape wurden zudem einige Grafiken, wie beispielsweise Flussdiagramme, mit Microsoft Visio [Vis] entworfen. Visio ist ausschließlich auf Windowsplattformen nutzbar und kann für Studenten im Rahmen von Dreamspark [Dre] kostenfrei für nicht kommerzielle Zwecke genutzt werden.

3.4 Risikomanagement

Bei der Betrachtung des Risikomanagements muss das Augenmerk auf zwei verschiedene Abschnitte des Entwicklungsprozesses in dem Projekt *RCCARS* gelegt werden. Zum Einen auf die Einarbeitungs- und Anforderungsdefinitionsphase in derer nicht das Risikomanagement des Vorgehensmodells aus Kapitelabschnitt 3.1.2 greift. Diese wird im Folgenden zuerst betrachtet. Zum Anderen auf die Sprintphasen. Innerhalb dieser Phase greift das Risikomanagement des *ScrVm* Modells.

Einarbeitungs- und Anforderungsdefinitionsphase

Diese Phase muss gesondert betrachtet werden da, wie bereits erwähnt, hier ein spezielles Risikomanagement eingeführt werden muss. In der Einarbeitungs- und Anforderungsdefinitionsphase beschäftigt sich das Team mit dem Einarbeiten in die Materie. Dazu zählt unter anderem das Erarbeiten des aktuellen Stands der Technik (Kapitelabschnitt 2.1). Im Weiteren wird in dieser Phase mit dem Füllen des Product Backlogs (siehe Kapitelabschnitt 3.1.2) des Vorgehensmodells begonnen, Expertenwissen aufgebaut und zugehörig ein Prototyp erstellt. Der Stand des Prototyps wird in Kapitelabschnitt 13.3 näher erläutert.
Für das Zeit- und Projektmanagement wurde wie in Kapitelabschnitt 3.3 vorgestellt Microsoft Project verwendet. Mit diesem Tool werden die geplanten Vorgänge mit Fristen und Meilensteinen verwaltet. Als Meilensteine zählen beispielsweise die Abgabe von Dokumenten, unter anderem das hier aktuell betrachtete, sowie zugehörige Reviews. In der Einarbeitungs- und Anforderungsdefinitionsphase wird auf Grund des Projektmanagements vor dem Beginn des ersten Sprints von *ScrVm* besonders auf die Vermeidung von kritischen Vorgängen geachtet. Ein Vorgang wird als ”kritisch” bezeichnet sobald der Verzug dieses Vorgangs zum Überschreiten der Frist eines Meilensteins führt. Dies wird durch die Einplanung von Pufferzeiten vor und nach jedem Vorgang umgesetzt. Das führt dazu, dass die Nichteinhaltung einer Deadline - und das kann durch diverse spontane Ereignisse wie zum Beispiel verzögerten Lieferterminen oder dem Verschätzen in der Zeitplanung eintreten - keine Auswirkung auf Folgevorgänge hat. Sollte ein Puffer nicht benötigt werden und ein Folgevorgang keine weiteren zum betrachteten Zeitpunkt hinderlichen Abhängigkeiten besitzt kann dieser bereits früher begonnen werden. Eine Abhängigkeit gilt als hinderlich, wenn diese den (vorzeitigen) Start eines Vorgangs verzögert, da ein oder mehrere andere abhängige Vorgänge noch nicht abgeschlossen sind.
, dass der Start eines Vorgangs von dem Ende eines oder mehrerer anderer Vorgänge abhängt, welche noch nicht abgeschlossen wurden und so einen frühzeitigen Start verhindern.

Im Weiteren muss der Projektplan vorausschauend geplant und immer aktuell und dynamisch angepasst werden. Wird eine Frist eines Vorgangs versäumt, inklusive eingeplanter Pufferzeit, so muss der Folgevorgang innerhalb dessen Pufferzeit, wenn möglich und vorhanden, verschoben werden.

Sprintphasen

Wie beim Beginn dieses Abschnitts beschrieben gibt es neben der Einarbeitungs- und Anforderungsdefinitionsphase eine weitere Phase - die Sprintphase. Die Sprintphase beginnt sobald das Product Backlog (siehe Kapitelabschnitt 3.1.1) in der ersten Version vollständig gefüllt wurde. Die vorerst vollständige Füllung des Product Backlogs ist mit der Abgabe der Anforderungsdefinition (siehe *Anforderungsdefinition vom 09. März 2016*) vorerst abgeschlossen. Zu diesem Zeitpunkt beginnt somit das Füllen des Sprint Backlogs und damit verbunden der erste *ScrVm* Sprints.
Mit dem Start der *ScrVm* Sprints können sich während und nach der Sprints (siehe Kapitelabschnitt 3.1.2) neue Anforderungen ergeben, welche dem Product Backlog hinzugefügt werden. Zusätzlich können nicht abgearbeitete Anforderungen in einem Sprint aus dem Sprint Backlog ebenfalls in das Product Backlog zurückgeführt werden. Dadurch entsteht das Risiko, dass das Product Backlog bedingt durch die Sprints nicht minimiert, sondern maximiert wird. Um diesem Risiko entgegen zu wirken, dürfen nachträglich nur Anforderungen in das Product Backlog aufgenommen werden, die zwingend erforderlich sind um das Zielszenario (siehe Kapitelabschnitt 2.4) zu erreichen. Zusätzlich dürfen nur Anforderungen aus dem Sprint Backlog in das Product Backlog zurückgeführt werden, wenn diese durch unvorhersehbare Ereignisse im aktuellen Sprint unerreichbar sind. Unvorhersehbare Ereignisse stellen beispielsweise plötzlicher vorübergehender Personalausfall (z.B. krankheitsbedingt), Hardwaredefekte oder Lieferverzug dar. Zusätzlich erhöht sich die Priorität zurückgeführter Anforderungen. Dadurch müssen diese im folgenden Sprint Backlog erneut mit aufgenommen werden.

3.5 ScrVm-Dokument

In der Phase nach dem ersten Review wurde ein Dokument angelegt, welches bei der Sprintplanung und der allgemeinen Anforderungsverwaltung helfen sollte - das ScrVm-Planungsdokument. Dieses Dokument hat das Ziel den Verlauf des *ScrVm* Prozesses verfolgen zu können. Im Weiteren soll es eine Übersicht über den Projektfortschritt geben und dabei helfen die Anforderungen an das System zu verwalten. Es unterstützt zudem - neben dem offiziellen Projektplan - die Projektmitglieder darin, den Überblick über die

Aufgaben und Ziele des jeweiligen Sprints zu behalten.
Das Dokument ist wie folgt aufgebaut: Zu Beginn wird die Handhabung des Dokuments beschrieben. Darauf aufbauend wird der Product Backlog aufgeführt, der alle aufgenommenen Softwareanforderungen beinhaltet. Das nächste Kapitel in dem Dokument beinhaltet alle höher priorisierten Anforderungen. Eine Anforderung wird höher priorisiert, wenn Sie innerhalb eines Sprints zurückgeschoben oder neu aufgenommen wird. Dazu zählt die Referenz zur Anforderung, der Grund warum diese Anforderung höher priorisiert ist, die Auflage zur Anforderung und den aktuellen Status. Dieses Kapitel unterstützt dabei alle zurückgeführten oder neu aufgenommenen Anforderungen zu verwalten.
Im Anschluss daran folgen die Sprints. Die Sprints werden beschrieben durch das ausführende Team, den Sprint Zeitraum, den Sprint Backlog und die zugehörigen Testfälle zu den Anforderungen. Außerdem werden nicht erfüllte Anforderungen sowie neu aufgenommene Anforderungen hier festgehalten (welche ebenfalls in das zuvor erwähnte ”höher priorisierte Anforderungen” Kapitel aufgenommen werden). Zuletzt beinhaltet das Dokument eine Übersicht über alle bereits erfüllte Anforderungen, welches am Ende dem Product Backlog entsprechen muss. Dieses Kapitel wurde eingeführt, da der Product Backlog aus der Anforderungsdefinition (siehe Kapitelabschnitt 5.2.3) inkludiert wird und es technisch nicht möglich ist diesen Product Backlog entsprechend der Sprintphasen zu verkleinern. Aus diesem Grund werden alle Referenzen zu den erfüllten Anforderungen am Ende des Dokuments festgehalten, um die Übersicht zu steigern.

3.6 Zeitmanagement

In diesem Abschnitt wird das Zeitmanagement der Projektgruppe *RCCARS* dargelegt. Zu Beginn wird der Zeitraum vom Start der Projektgruppe im Oktober 2015 bis zum ersten Review, welches am 09. März 2016 statt fand, betrachtet. Um diese Phase besser darstellen zu können, wird Grafik 3.4 herangezogen.
Diese Phase bezog sich auf das Einarbeiten in das Projektthema und das Spezifizieren der Anforderungsdefinition. In dieser Phase wurden Referenzprojekte studiert um Hardwareanforderungen festzulegen. Diese Hardwareanforderungen waren die Grundlage für den darauffolgenden Auswahlprozess der Hardware, welche bestellt werden sollte. Im Weiteren diente die Grundlagenrecherche dazu, das endgültige Projektziel (siehe Kapitelabschnitt 2.3) zu definieren. Zudem wurde in dieser Phase entsprechend zur Anforderungsdefinition der Product Backlog (siehe Kapitelabschnitt 3.1.2) gefüllt und ein erster Prototyp zur Demonstration entworfen. Abgeschlossen wurde diese Phase mit dem ersten Review.
Auf die Einarbeitungs- und Anforderungsdefinitionsphase folgte mit einem Zeitraum vom

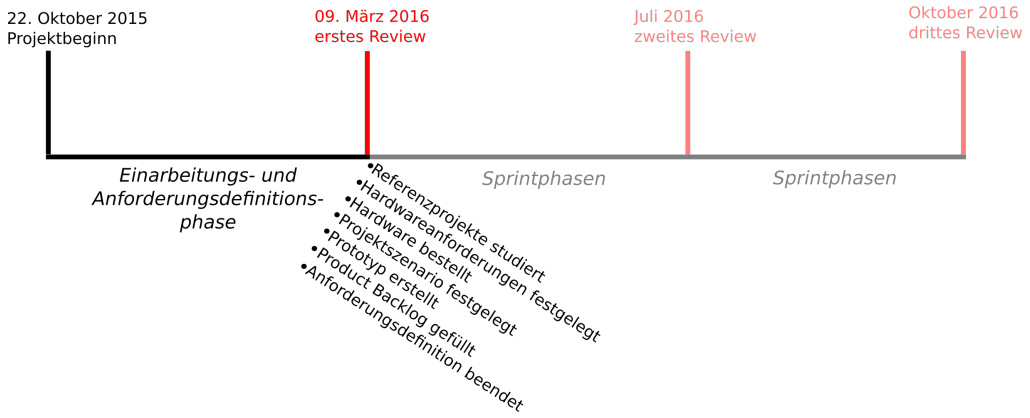


Abbildung 3.4: Zeitleiste, welche die Zeit vom Beginn der Projektgruppe bis zum ersten Review zeigt.

09. März 2016 bis zum 07 Juli 2016 die erste Sprintphase. Auch die Beschreibung dieses Zeitraums wird anhand einer Grafik (siehe Abbildung 3.6) beschrieben. Die erste Sprintphase hatte eine Laufzeit von etwa vier Monaten und beinhaltete zwei Sprints pro Arbeitsgruppe, woraus sich insgesamt vier Sprints zu je rund sechs Wochen ergaben (Grafik 3.5 zeigt den Verlauf der Sprints genauer).

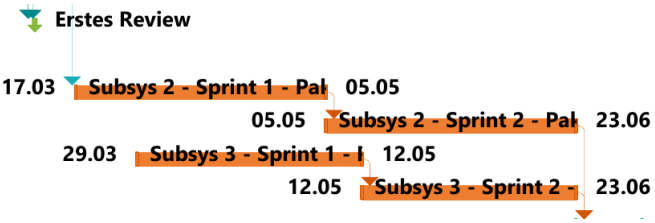


Abbildung 3.5: Sprintplanung vom ersten Review bis zum zweiten Review (Ausschnitt Projektplan).

Zu Beginn der ersten Sprintphase arbeiteten sich die zwei Arbeitsgruppen jeweils in ihr Themengebiet ein und starteten im Anschluss mit dem jeweiligen ersten Sprint und zugehörigem Arbeitspaket. Dabei begann die Arbeitsgruppe Control-Unit eine Woche später als Arbeitsgruppe Positionsbestimmung mit dem ersten Sprint. Dies hing mit dem höher geschätzten Einarbeitungsaufwand in das Softwaretool *SCADE* (siehe Kapitelabschnitt 3.3) zusammmen.
Die erste Sprintphase der Arbeitsgruppe Positionsbestimmung beinhaltete das zugehörige Arbeitspaket 1. Das Arbeitspaket 1 der Positionsbestimmungsgruppe zielte auf das korrekte Einlesen und Verarbeiten der Kamerabilder ab. Dazu gehörte das Detektieren der Fahrzeugmarker und Rennstreckenmarker. Das Arbeitspaket 2 der Gruppe Positionsbestimmung sollte aus den erkannten und verarbeiteten Bildern des Arbeitspaket 1 die korrekte Streckenposition und sowie die Fahrzeugpose berechnen. Die verarbeiteten Daten sollten im Anschluss an die Control-Unit gesendet werden. Daraus ergibt sich nach

Abschluss der beiden Sprintphasen eine fertiggestellte Positionsbestimmung (siehe Kapitelabschnitt 4.2), welches eines der drei Ziele für das zweite Review darstellte (siehe Abbildung 3.6).

Die erste Sprintphase der Arbeitsgruppe Control-Unit begann am 29. März 2016 mit dem zugehörigen Arbeitspaket 1. Das Arbeitspaket 1 zielte auf das korrekte Einlesen der Daten der Positionsbestimmung ab. Im Weiteren sollte die korrekte Lage der Rennstrecke erfasst und eine passende Zielspur, welche in der aktuellen Ausbaustufe die Mittelspur umfasst, (siehe Kapitelabschnitt 4.4.2) mit Richtung festgelegt werden. Das Arbeitspaket 2 der Control-Unit Gruppe erweiterte das Arbeitspaket 1 und sollte eine Plausibilitätsüberprüfung (siehe Kapitelabschnitt 4.3.3) mit den empfangen Daten der Positionsbestimmung durchführen. Im Weiteren sollten Funktionen implementiert werden, die eine Kollision eines Fahrzeugs mit der Fahrbahnbegrenzung sowie das Verlassen der Rennstrecke feststellen können. Zusätzlich sollte die Control-Unit in der Lage sein Steuerungsdaten zur CarControl zu senden. Dafür ist eine Sprintlaufzeit von jeweils sechs Wochen angesetzt worden. Aus den Sprints der Control-Unit Gruppe ergibt sich ein ”Grundgerüst” der Control-Unit, welches in der Lage sein sollte Daten anzunehmen, diese zu verarbeiten und Daten an die CarControl zu senden. Eine direkte Fahrzeugregelung zu implementieren war nicht Teil dieser Sprintphase, sondern der Sprintphase nach dem zweiten Review.

Anhand dieser Planung sollten beide Gruppen ihre letzten Sprints am 23. Juni 2016 beenden. Die Folgezeit bis zum zweiten Review sollte zusätzlich in den zugehörigen Bericht investiert werden. Aus den zuvor beschriebenen Sprints mit den zugehörigen Zielen ergab sich das dritte Ziel zum zweiten Review (siehe Abbildung 3.6): die funktionstüchtige Kommunikation vom Einlesen der Bilder von der Positionsbestimmung bis hin zur Annahme der Daten durch die Control-Unit und ein mögliches Senden von Daten zur CarControl.

Die für Scrum verhältnismäßig langen Laufzeiten der Sprints leiteten sich aus den mangelnden Ressourcen der Projektgruppe *RCCARS*, sowie schwierig abzuschätzenden Arbeitsaufwand ab.

Das Ziel zum dritten Review war die Umsetzung des ersten Szenarios (siehe Kapitelabschnitt 2.3). Dazu zählte die allgemeine Fertigstellung beziehungsweise die Optimierung, insbesondere der Performanceoptimierung, der Positionsbestimmung, da diese zum zweiten Review nicht vollständig fertiggestellt werden konnte. Zudem sollte die Systemsteuerungssoftware entworfen werden, die letztendlich die Subsysteme Positionsbestim-

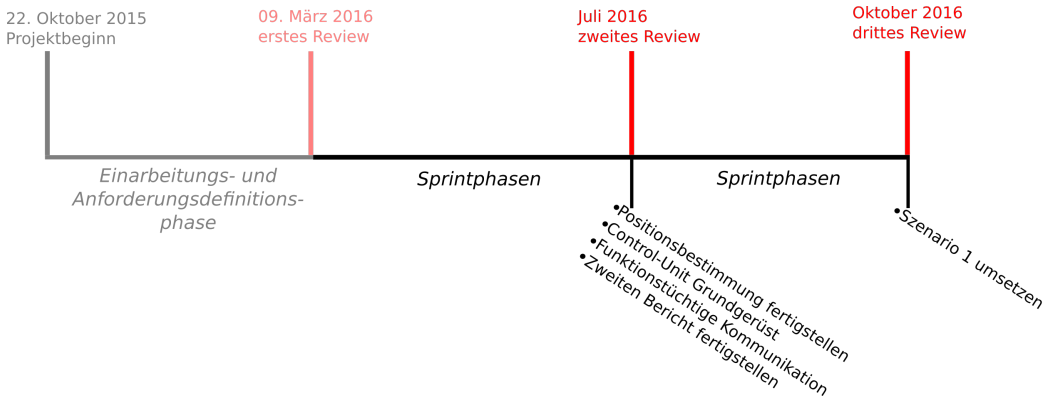


Abbildung 3.6: Zeitleiste, welche die Zeit vom ersten Review bis zum zweiten Review zeigt.

mung und Control-Unit (siehe Kapitelabschnitt 4.2 und 4.3) steuert und verwaltet. Eine genauere Beschreibung der Systemsteuerungssoftware ist in Kapitelabschnitt 10.4 zu finden. Parallel zu diesen Zielen wurde die Control-Unit weiter ausgebaut. Bis zum zweiten Review wurde das Grundgerüst der Control-Unit mit entsprechender Grundfunktionalität und Kommunikationsvermögen entwickelt. In der aktuell betrachteten Phase zwischen zweitem und dritten Review wurde das Hauptmerkmal auf die Implementierung der Regelungsalgorithmen gelagert. Im Weiteren war vorgesehen mehrfache Plausibilitätsüberprüfungen in der Positionsbestimmung sowie der Control-Unit anhand von vorangegangen Daten durchzuführen sowie eine detaillierte Störungserkennung in das Grundgerüst der Control-Unit zu implementieren.

Die vorangegangene Beschreibung und Planung der zwei Sprintphasen wird anhand von Grafik 3.7 noch einmal verdeutlicht. Zu sehen ist oberhalb der blau gestrichelten Linie die erste Sprintphase, welche vom ersten Review (09. März 2016) bis zum zweiten Review (07. Juli 2016) verlief. Dort sind zwei Teams mit je zwei Sprints dargestellt. Nach dem zweiten Review - zu sehen unterhalb der blauen Linie - wurde das Team neu aufgeteilt. In dem ersten Sprint der zweiten Sprintphase (Sprint 3), welcher vom 07. Juli 2016 bis 26. August 2016 angesetzt ist, organisierte sich das Team neu, um die vorhanden Ressourcen optimal zu nutzen. Das Ziel war es die Positionsbestimmung in diesem Sprint fertigzustellen. Ebenfalls die Systemsteuerungssoftware, mit deren Entwicklung zuvor noch nicht begonnen wurde, sollte in diesem Sprint vollendet werden. Bei der Control-Unit ging es ab der zweiten Sprintphase hauptsächlich darum die Regelalgorithmen zu entwerfen und zu implementieren. Da dies höchste Priorität hatte, wurde auch kontinuierlich die größte zur Verfügung stehende Menge an Ressourcenpotential der Control-Unit zugeteilt.
Im Sprint vier sollte die Entwicklung der Positionsbestimmung sowie der Systemsteuerungssoftware abgeschlossen sein und das gesamte Team an der Weiterentwicklung der

Control-Unit arbeiten. Der Sprint vier endete im Oktober 2016. Die restliche Zeit bis zum dritten Review sollte zur Vervollständigung des Endberichts genutzt werden.

Die genaue Zeitplanung ist anhand des Projektplans im Anhang unter Kapitelabschnitt 14.2 vollständig einsehbar. Ein Fazit zum hier beschriebenen Zeitmanagement ist in Kapitelabschnitt 13.2.2 zu finden.

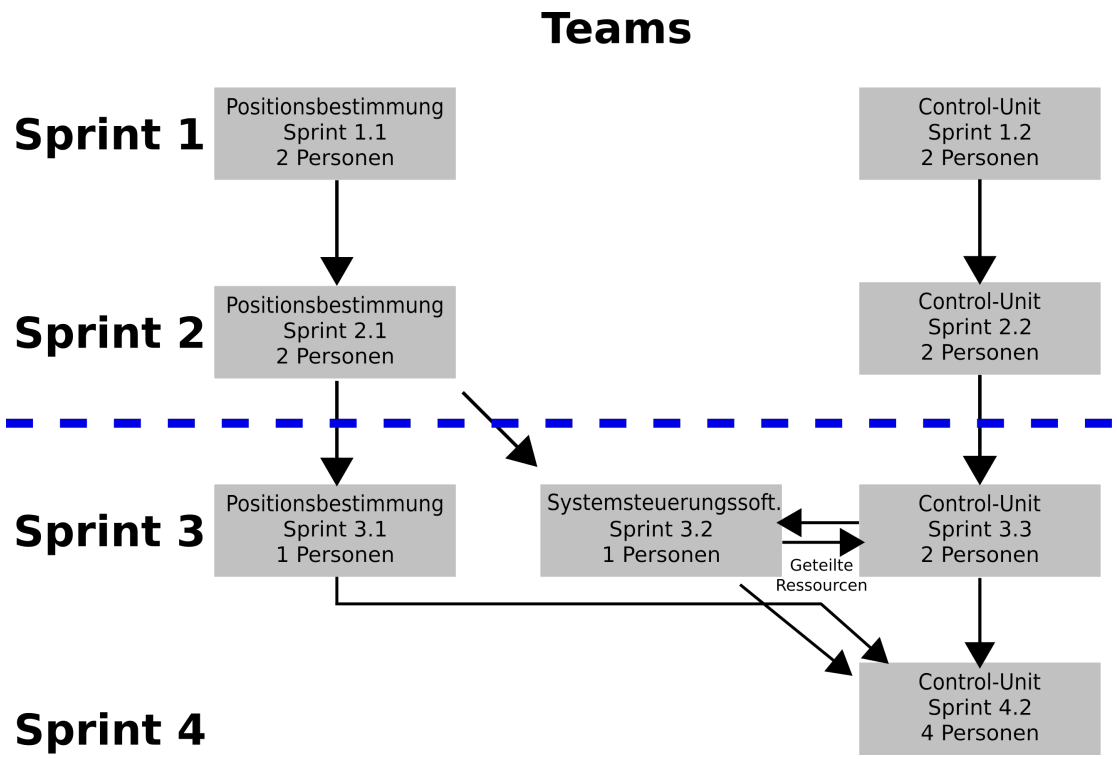


Abbildung 3.7: Sprintplanung vom ersten bis zum dritten Review. Oberhalb der blau gestrichelten Linie findet sich die erste Sprintphase, also die Phase nach dem ersten Review wieder, welche an die Einarbeitungs- und Anforderungsdefinitionsphase anschloss. Die Zeit unterhalb der Linie stellt die Zeit zwischen zweitem und dritten Review dar. Die blaue Linie selbst beschreibt den Zeitpunkt des zweiten Reviews (07. Juli 2016).

3.7 Meilensteine

In dieser Projektgruppe gab es drei offizielle Meilensteine. Dazu zählen die Reviews, welche am 09. März 2016, 07. Juli 2016 sowie am 10. November 2016 stattfanden. Diese Reviews setzen sich zum Einen aus der Vorstellung des aktuellen Entwicklungsstands inklusive des bisherigen Verlauf des Projekts zusammen. Zum Anderen ging jedes Review einher mit der Abgabe eines ausgiebigen Berichts, welcher den gesamten Projektfortschritt, sowie -verlauf beschreibt.

Im Folgenden wird genauer auf die Meilensteine eingegangen.

1. Review

Bei dem ersten Review, welches am 09. März 2016 statt fand, ging es in erster Linie darum die bisher getriebene Grundlagenforschung vorzustellen. Dabei wurde präsentiert, welchen Fortschritt andere Universitäten im selben Forschungsbereich erzielt haben und welche Hard- und Softwareanforderungen bei diesem Projekten galten. Anhand dessen wurde auch die letztendliche Aufgabenstellung der Projektgruppe *RCCARS* (siehe Kapitelabschnitt 2.4) abgegrenzt und eine entsprechende Long Term Vision (siehe Kapitelabschnitt 2.2) vorgestellt.
Im Weiteren wurde die Systembeschreiben mit der Darstellung der einzelnen modularen Subsysteme des Projekts aufgezeigt. Ein zusätzlicher wichtiger Punkt des Reviews war die Präsentation der Anforderungsspezifikation (siehe Kapitelabschnitt 5).

2. Review

Im zweiten Review, welches am 07. Juli 2016 statt fand, ging es vordergründig um das Vorstellen der Schnittstellen. Im Weiteren wurde ein Sicherheitskonzept aufgezeigt, die Hardware evaluiert und die bis zu dem Zeitpunkt entwickelte Software beschrieben.
Zusätzlich wurde der aktuelle Entwicklungsfortschritt anhand eines Demonstrators vorgestellt.

3. Review

Zum dritten Review, welches am 10. November 2016 statt fand, ging es um die Systemabnahme des Projekts. Es wurde der gesamte Entwicklungsfortschritt vorgestellt und überprüft, ob das entstandene System so von den Gutachtern „abgenommen wird“, also ein Systemabnahmetest durchgeführt. Zusätzlich wurde ein umfangreicher Endbericht geschrieben, der Teile der Berichte vom ersten Review (siehe *Anforderungsdefinition vom 09. März 2016*) sowie vom zweiten Review (siehe *Architektur und Schnittstellenspezifikation vom 07. Juni 2016*) beinhaltet, welcher Ihnen hiermit vorliegt.

3.8 Öffentlichkeitsarbeit

Dieses Kapitel beinhaltet Informationen zur Öffentlichkeitsarbeit, die neben der eigenständig geführten Website (`www.uni-oldenburg.de/rccars`) im Laufe der Projektlaufzeit entstanden sind. Dazu gehört bisher zum Einen der Schülerinformationstag und zum

Anderen die Re-Akkreditierung der Informatikstudiengänge. Im Weiteren wurde an der SCADE Adacemic Community Conference 2016 teilgenommen und die Projektgruppe präsentierte auf der Erstsemesterbegrüßung 2016 ihren Demonstrator.

Schülerinformationstag

Am Schülerinformationstag, welcher am 20. Januar 2016 statt fand, wurde die zum derzeitigen Zeitpunkt gesamte Hardware in die Carl von Ossietzky Universität gefahren und dort aufgebaut. Dazu zählte die Rennstrecke, zwei Fahrzeuge und zwei entsprechende Fernbedienungen. Im Weiteren wurde eine eigenständig entwickelte Lichtschranke in die Rennstrecke eingebaut, die Rundenzeiten sowie die gefahrene Rundenanzahl festhalten konnte.
Die gesammelten Daten ergaben, dass an diesem Tag von Schülern etwa 5000 Runden auf der Rennstrecke zurückgelegt wurde. Die gesammelten Daten gaben Aufschluss darüber, wie lange in etwa ein Akku eines Fahrzeugs hält und wie die durchschnittliche Rundenzeit eines ungeübten menschlichen Fahrers ist. Hierbei ist jedoch zu beachten, dass der dort aufgebaute Rennstreckenverlauf von dem aktuell verwendeten Verlauf (siehe Kapitelabschnitt 4.4.2) abweicht. Im Weiteren halfen die Daten, die genutzte Hardware unter Betrachtung der Hardwareanforderungen (siehe Kapitel 5) zu evaluieren. Weitere Informationen zu der Evaluation des Fahrzeugs sind in Kapitelabschnitt 7.1.1 beschrieben.

Re-Akkreditierung

Im April 2016 wurde der aktuelle Stand der Projektgruppe auf der Akkreditierung der Informatik Studiengänge präsentiert. Das vorrangige Ziel war bei den Gutachtern ein positives Bild der Oldenburger Informatik zu hinterlassen, in dem einige interessante Projektgruppen ihre Exponate vorstellten - darunter die Projektgruppe *RCCARS*.
Die gesamte Hardware des Prototyps wurde deshalb in die Carl von Ossietzky Universität Oldenburg überführt und dort aufgebaut. Der Rechner wurde an ein TV Endgerät angeschlossen und der Fortschritt bei der Positionsbestimmung präsentiert. Darunter spielte vor allem die Detektierung des Fahrzeugs eine Rolle. Die Detektierung wurde auf dem TV Endgerät angezeigt, während ein Projektgruppenmitglied mit einer Fernbedienung ein Fahrzeug über die Rennstrecke steuerte. Ein anderes Projektgruppenmitglied erklärte wie die Gesamthardware funktioniert und zusammenarbeitet und wie genau die Detektierung des Fahrzeugs funktioniert, insbesondere das, was letztendlich auf dem Fernsehgerät zu sehen war.
Die Gutachter zeigten in vertiefenden Rückfragen starkes Interesse an dem Thema der Projektgruppe *RCCARS*. Eine Rückmeldung der Organisatorin bestätigte die positive Resonanz ebenfalls.

SCADE Adacemic Community Conference 2016

Am Donnerstag, dem 08. September 2016 fand die SCADE Academic Community Conference 2016 (kurz: SACC) im ZAL Tech Center in Hamburg statt.
Neben Vorträgen zu und über die Software SCADE haben die Betreuer der Projektgruppe *RCCARS*, Günter Ehmen und Stefan Puch, die Carl von Ossietzky Universität Oldenburg, den Projektgruppenbetrieb, die Projektgruppe *RCCARS* und den aktuellen Stand dieser vorgestellt. Gespannt verfolgten die Besucher aktiv den Vortrag. Während der Pausen konnten alle Besucher den aktuellen Stand der Projektgruppe anhand des Demonstrators testen. Besonders der aktuelle Stand des Projektes in Anbetracht der geringen Anzahl an Projektgruppenmitgliedern traf auf Erstaunen und resultierte in Lob.

Erstsemesterbegrüßung

Am Mittwoch, dem 12. Oktober 2016 (während der Orientierungswoche), waren circa 130 Studierende aus dem ersten Semester im Rahmen der Orientierungswoche im OFFIS.
Nach der Begrüßung und einigen Vorträgen wurden Exponate präsentiert, welche im OFFIS bearbeitet werden. Darunter auch der *RCCARS* Demonstrator. Innerhalb von jeweils 15 Minuten wurden fünf Gruppen aufgezeigt, wie sich ein Master-Projekt zusammensetzt, was die Projektgruppe *RCCARS* in den letzten fast zwölf Monaten geschafft hat und wie das System aufgebaut ist. Das autonom fahrende Fahrzeug stieß auch hier auf Begeisterung. Die Zuschauer nahmen durch zahlreiche Fragen aktiv an der Vorstellung teil und zeigten so Interesse am Projekt und den Abteilungen.

Website

Die Projektgruppe *RCCARS* führte während der gesamten Projektgruppenlaufzeit aktiv eine Website. Auf der Website wird das Thema *RCCARS*, das Team sowie die Projektziele vorgestellt. Im Weiteren gab es einen aktiven Blog, der wöchentlich mit neuen Informationen gefüllt wurde. Zudem wurden alle Veranstaltungen an denen die Projektgruppe teilgenommen hat dargestellt. Außerdem ist es auf der Website möglich alle Berichte der Projektgruppe herunterzuladen.

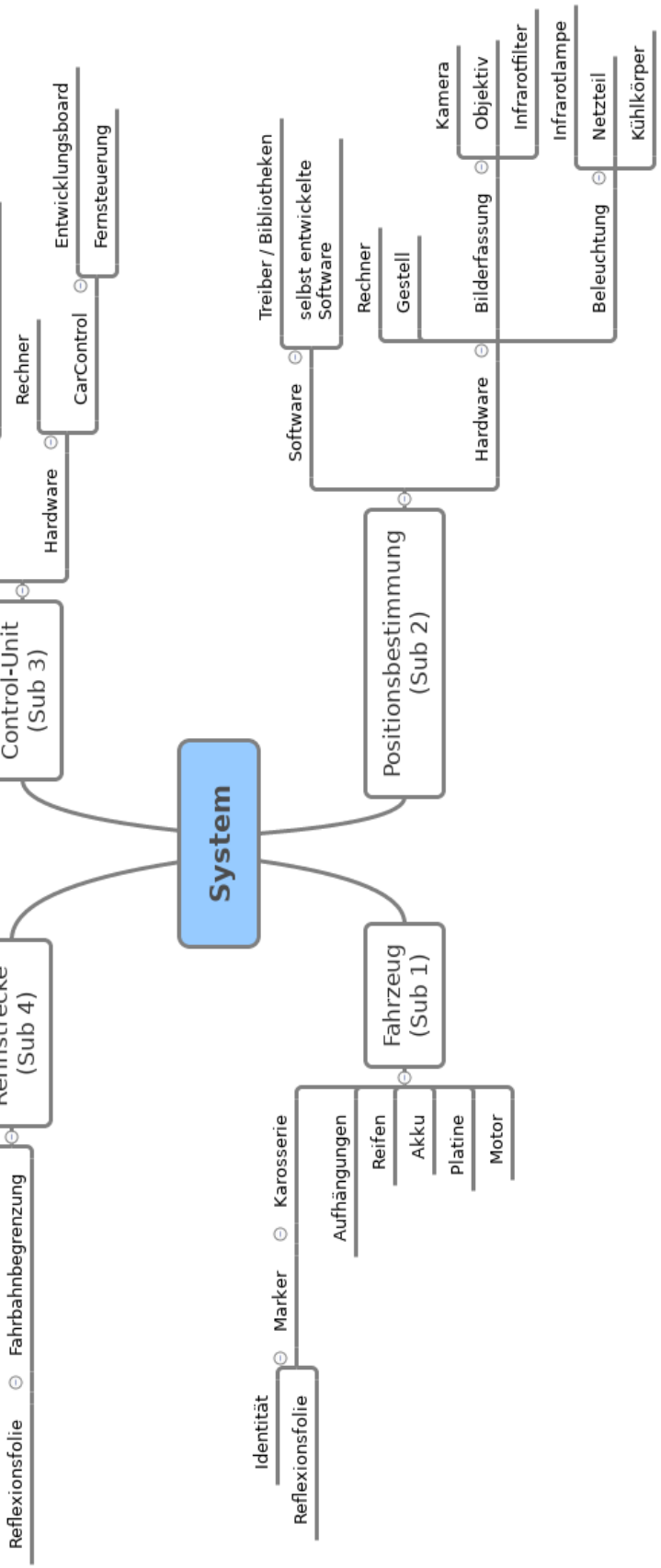


Abbildung 3.8: *Mind-Map* zur Aufteilung des Systems in Module und Komponenten

4 Systembeschreibung

Ausgehend von der Aufgabenstellung und den Ergebnissen der in Kapitelabschnitt 2.1 beschriebenen Referenzprojekte, werden in diesem Kapitel der Aufbau und die Komponenten der ersten Ausbaustufe des zu entwickelnden Systems beschrieben. Damit dieses im Rahmen zukünftiger Ausbaustufen erweitert werden kann, ohne dass grundlegende Änderungen des gesamten Systems vorgenommen werden müssen, wurde beim Entwurf darauf geachtet, dass das System modular aufgebaut ist. Abbildung 3.8 zeigt eine *Mind-Map*, welche die Struktur des Systems darstellt.
Das System setzt sich aus vier Subsystemen zusammen. Hierbei handelt es sich um das Fahrzeug (Subsystem 1), die Positionsbestimmung (Subsystem 2), die Control-Unit (Subsystem 3) sowie die Rennstrecke (Subsystem 4). Außerdem verfügt das System über eine Systemsteuerungssoftware zur Steuerung und Überwachung des Systembetriebs. Im Folgenden werden zunächst die einzelnen Subsysteme näher erläutert. In Abschnitt 4.8 wird schließlich erklärt wie die Komponenten der Subsysteme untereinander in Verbindung stehen, um die Regelung der autonomen Fahrzeugsteuerung zu realisieren.

4.1 Subsystem 1: Fahrzeug

Das Subsystem Fahrzeug besteht aus einem ferngesteuerten Fahrzeug in einem Maßstab von 1:43. Das Fahrzeug wird von der Positionsbestimmung erfasst und mit Hilfe der Control-Unit autonom über die Rennstrecke gesteuert.

4.1.1 Fahrzeug

Das im Projekt *RCCARS* verwendete Fahrzeug ist ein RC-Car der Marke *Kyosho* des Modelltyps *dNaNo FX-101* [KYO]. Das Fahrzeug weißt eine Länge von ca. 10,5 cm und eine Breite von ca. 4,6 cm auf. Es besteht aus einer Karosserie, einem Motor, den Aufhängungen, einem Reifensatz, einem Akku und einer Platine. Das Fahrzeug bietet die Möglichkeit die Aufhängungen sowie die Reifen und den Akku auszutauschen. Insbesondere die Veränderung der ersten beiden Komponenten hat großen Einfluss auf die Fahrdynamik und Performanz [SSW+14]. Außerdem handelt es sich um Verschleißteile, welche ggf. ersetzt werden müssen.

Die Karosserie ist mit Markierungen aus reflektierender Folie (siehe 4.1.2) ausgestattet. Durch diese Reflexionsmarker kann die Positionsbestimmung die Fahrzeugpose sowie dessen eindeutige Identität feststellen. Unterschiedliche Fahrzeuge können über individuelle Anordnungen von Reflexionsmarkern identifiziert werden. Die Codierung der Fahrzeuge ist in Abbildung 8.2 aus Kapitel 8.2.2 dargestellt und orientiert sich zum Teil am Referenzprojekt der ETH Zürich [Rut10]. Die eindeutige Identität wird für die geplante Erhöhung der Fahrzeuganzahl in Folgeszenarien benötigt.

4.1.2 Reflexionsfolie

Zur Erkennung der Fahrzeugpose und für die Identitätsermittlung der RC-Cars werden diese mit einer selbstklebenden Reflexionsfolie der Marke *3M* mit der Modellbezeichnung *Scotchlite Reflective Material 8850 Silver* [Fol] ausgestattet, die unter anderem das für die Positionsbestimmung wichtige Infrarotlicht reflektiert.
Eine Eigenschaft des verwendeten Reflexionsmaterials ist, dass das auf das Material treffende Licht nicht im selben Winkel reflektiert wird, in dem es einfällt. Stattdessen wirft das Material das Licht immer zurück in die Richtung der Lichtquelle. Dies ist relevant, um unabhängig von der Pose des Fahrzeugs, das Infrarotlicht in hoher Intensität in Richtung der Lichtquelle zurückzustrahlen (siehe auch Abschnitt 4.8 zum Signalfluss).

4.1.3 Sicherheit

Das Fahrzeug ist ein sicherheitskritischer Faktor im Projekt *RCCARS*. Der Zustand des Fahrzeugs ist essentiell für die Fahrdynamik. Um diese nicht negativ zu beeinflussen, müssen Verschleißteile regelmäßig überprüft werden. Verschleißteile sind vor allem die Reifen, aber auch der Akku und die Aufhängungen. Auch Verunreinigungen können das Fahrverhalten beeinflussen, daher müssen die Reifen und Aufhängungen regelmäßig von Staub und Reifenabrieb befreit werden.
Bezüglich der Reflexionsmarker ist zu beachten, dass diese groß genug sein müssen, um von der Positionsbestimmung zuverlässig erkannt zu werden. Laut Referenzprojekt der EHT Zürich sollten diese Marker eine Größe von $1\,\text{cm} \cdot 1\,\text{cm}$ haben, damit sie im Bild der Kamera jeweils eine Größe von mindestens zwei Pixeln haben [Rut10].

4.2 Subsystem 2: Positionsbestimmung

Das Subsystem Positionsbestimmung ist für die Bilderfassung und -verarbeitung zuständig. Dazu gehört das Einlesen der Bilder sowie die Ermittlung der aktuellen Pose eines Fahrzeugs. Das Subsystem unterteilt sich in Software und Hardware.

Zur Software gehören Bibliotheken und Treiber für das Einlesen der Bilder, die selbst entwickelte Software für die Bildverarbeitung und die Bestimmung der Pose des Fahrzeugs (siehe Abschnitt 4.2.7). Zur Hardware gehören ein Beleuchtungssystem, ein Kamerasystem und ein Rechner, auf welchem die selbst entwickelte Software ausgeführt wird. Das Beleuchtungssystem besteht aus Infrarotlampen mit Kühlkörper und Netzteil. Das Kamerasystem setzt sich aus einer Kamera mit Objektiv und Infrarotfilter zusammen. Für die Befestigung von Kamera- und Beleuchtungssystem ist ein Gestell erforderlich. In den folgenden Abschnitten 4.2.1 bis 4.2.6 werden diese Hardwarekomponenten näher beschreiben.

4.2.1 Infrarotlampe

Position und Ausrichtung des Fahrzeugs auf der Rennstrecke werden von der Kamera mit Hilfe von Reflexionsmarkern erkannt. Um die Identifizierung des Fahrzeugs zu erleichtern, wird die Rennstrecke mit Infrarotlicht bestrahlt. Die ETH Zürich hatte bei ihren Projekten vor 2010 ein Positionsbestimmungssystem verwendet, welches die Fahrzeuge bei sichtbaren Licht observiert hatte. Das Problem bestand jedoch darin, dass die erreichte Framerate zu gering war und es nicht möglich war, ähnlich aussehende Fahrzeuge anhand ihrer Farbe zu unterscheiden. Um die Bildwiederholungsrate zu verbessern, wurde zukünftig eine Positionsbestimmung mit Hilfe einer monochromen Kamera und Infrarotlicht entwickelt, da so im Endeffekt nur noch zwischen hellen und dunklen Pixeln unterschieden werden musste [Rut10].
Daher wird im Projekt *RCCARS* ebenfalls eine Infrarotlampe verwendet. Die in Kapitel 2.1 dokumentierten Rechercheuntersuchungen zu Infrarotlampen der Referenzprojekte zeigten, dass mehrheitlich die Lampe *SVL BRICK LIGHT S75-850* verwendet wird, welche jedoch sehr kostenintensiv ist. In diesem Projekt soll daher die auch von der Universität Göteborg verwendete, günstigere Variante des Typs *OSLON 4 IR Wide PowerStar* eingesetzt werden.
Die Lampe besteht aus vier Infrarot-LEDs, die auf einem Leiterplatten-Array angeordnet sind. Die Wellenlänge der LEDs beträgt 850 nm. Für die Stromversorgung und die Kühlung der LEDs wird ein Netzteil und Kühlkörper benötigt [IR]. Außerdem muss die Lichtquelle so dicht wie möglich an der Kamera befestigt werden, da das Reflexionsmaterial, wie in Kapitelabschnitt 4.1.2 beschrieben, das Licht in die Richtung zurück wirft, aus der es kommt [CAR14].
An der Universität Göteborg kam es zu Problemen bei der Bildverarbeitung, da die Infrarotlichtintensität zu den Rändern der Rennstrecke hin abfällt. Dieses Phänomen ließ sich auch bei der Entwicklung dieses Systems beobachten, sodass eine zweite Lampe zur Verbesserung der Lichtverhältnisse am Gerüst installiert wurde.

4.2.2 Infrarotfilter

Um das Infrarotlicht, welches von der Reflexionsfolie auf den Fahrzeugen reflektiert wird (siehe Reflexionsfolie in Abschnitt 4.1.2) vom sichtbaren Licht der Umgebung zu trennen, wird ein entsprechender Filter auf dem Objektiv vor der Kamera benötigt. Es wird der Infrarotfilter *MIDOPT FIL LP780/27* eingesetzt, welcher auch bei den in Kapitelabschnitt 2.1 genannten Referenzprojekten der Universitäten Uppsala und Göteborg verwendet wird. Dieser Filter lässt nur Strahlung mit einer Wellenlänge oberhalb von 780 nm passieren, sodass er für das verwendete Infrarotlicht mit 850 nm geeignet ist [Fil].

4.2.3 Objektiv

Bedingt durch die limitierte Raumhöhe ist ein Weitwinkelobjektiv notwendig, um die gesamte Rennstrecke mit der verwendeten Kamera (siehe Abschnitt 4.2.4) erfassen zu können. Es wird das Objektiv *Lensagon CMFA0420ND* [Obj] verwendet, welches auch im Referenzprojekt der Universität Uppsala genutzt wird. Das Objektiv hat einen horizontalen Erfassungswinkel von 75,14° und einen vertikalen von 59,96°. Dies ist für die Berechnung der Kamerahöhe relevant (siehe Abschnitt 4.2.5).

4.2.4 Kamera

Zur Erfassung der Pose des Fahrzeugs auf der Rennstrecke wird ein optischer Sensor benötigt. Es wird die Kamera *Point Grey Flea FL3-U3-13Y3M-C* (kurz *Flea 3*) [Kam] mit USB 3.0 Schnittstelle verwendet. Rechercheuntersuchungen zeigen, dass dieser Kameratyp in allen Referenzprojekten zum Einsatz kommt (siehe Kapitelabschnitt 2.1). Die Kamera zeichnet sich durch eine hohe Bildrate von maximal 150 Hz aus und besitzt eine Auflösung von $1280 \cdot 1024$ Pixeln. Beide Parameter sind wichtig, um eine präzise Regelung der Fahrzeuglenkung und Geschwindigkeit zu ermöglichen. Eine weitere Eigenschaft der Kamera ist, dass diese durch den verbauten *CMOS*-Sensor eine besonders gute Infrarotlichtempfindlichkeit besitzt [EGH+15].
Das Objektiv (siehe Kapitelabschnitt 4.2.3) und der Infrarotfilter (siehe Abschnitt 4.2.2) werden an der Kamera montiert.

4.2.5 Gestell

Die Kamera mit Objektiv und Infrarotfilter sowie die Infrarotlampe müssen zentral über der Rennstrecke angebracht werden. Aus diesem Grund wird ein Gestell benötigt.
Da die Seitenlänge von 237 cm der quadratischen Minimum Bounding Box der Rennstrecke sowie der Erfassungswinkel des Objektives bekannt sind, lässt sich daraus die Höhe,

in welcher die Kamera zu befestigen ist, berechnen. Abbildung 4.1 zeigt eine schematische Darstellung zur Berechnung der benötigten Kamerahöhe.

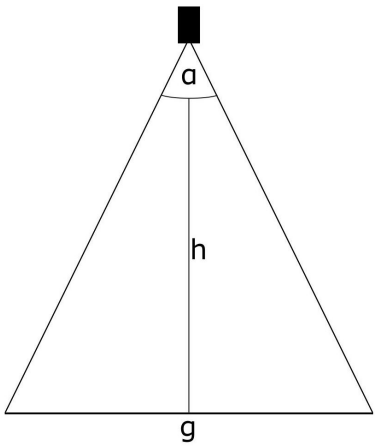


Abbildung 4.1: Berechnung der Höhe für die Kamera

Aus der dargestellten Seitenansicht ergibt sich ein gleichschenkliges Dreieck. Der vertikale Erfassungswinkel von 59,96° der als Rechteck dargestellten Kamera wird hierbei als Winkel α bezeichnet. Die Seitenlänge von 237 cm der Minimum Bounding Box wird als g und die zu ermittelnde Höhe der Kamera als h bezeichnet. Es wurde für die Berechnung bewusst der vertikale Erfassungswinkel genutzt, da bei der Verwendung des horizontalen Winkels Teile der Rennstrecke nicht von der Kamera erfasst werden würden. Die Höhe der Kamera kann mit der Tangens Funktion ermittelt werden:

$$\tan\left(\frac{\alpha}{2}\right) = \frac{0,5 \cdot g}{h} \tag{4.1}$$

$$\tan(29,98°) = \frac{118,5\,cm}{h} \tag{4.2}$$

$$h = \frac{118,5\,cm}{\tan(29,98°)} \tag{4.3}$$

$$\approx 205,41\,cm \tag{4.4}$$

Die Kamera muss daher in einer Mindesthöhe von 2,05 m zentral über der Rennstrecke positioniert werden, um diese vollständig zu observieren.

4.2.6 Rechner

Es wird ein Rechner benötigt, damit die Kamerabilder ausgewertet werden können. Aus den gewonnenen Daten berechnet der Rechner die Pose des Fahrzeugs und leitet diese an die Control-Unit (Abschnitt 4.3) weiter, welche in der ersten Ausbaustufe der Projektes *RCCARS* auf dem selben Rechner ausgeführt wird (siehe Abschnitt 4.3.1).

Es wird der Rechner *OptiPlex 7020MT* des Herstellers *Dell* verwendet. Dieser ist ausgestattet mit einem *i7-4790* Prozessor von *Intel*, 16 GB DDR3-Arbeitsspeicher, einer 500 GB Festplatte und USB 3.0 Schnittstellen. Die mit 3,6 Ghz getaktete CPU ist besonders leistungsfähig, da sie Hyper-Threading unterstützt. Dies ist bei der CPU intensiven Bildverarbeitung und den Realzeitanforderungen, welche an das System gestellt werden, notwendig (vergl. nächster Abschnitt 4.2.7).

4.2.7 Software

Die Software der Positionsbestimmung muss in der Lage sein, die aufgenommen Kamerabilder in einem von der Bildwiederholungsfrequenz der Kamera abhängigen Zeitrahmen zu empfangen, zu verarbeiten, auszuwerten und die ermittelten Daten weiterzuleiten. Ausgehend von der maximalen Bildwiederholungsfrequenz von 150 Bildern pro Sekunde der Kamera (vergl. 4.2.4), müsste die Software der Positionsbestimmung die oben genannten Schritte vom Empfang des Kamerabildes bis zur Weiterleitung der Pose in höchstens 6,7 ms durchführen, da nach diesem Zeitfenster bereits ein neues Kamerabild aufgenommen wird. Um die Bilder von der Kamera zu empfangen und ggf. die Kameraeinstellungen zu optimieren, wird die *Point Grey FlyCapture* Library [Fly] verwendet. Außerdem wird *OpenCV* [Ope] eingebunden, da die Bibliothek bereits zahlreiche Algorithmen zur Bildverarbeitung und speziell zur Objekterkennung bietet. Da beide Bibliotheken über C++ Interfaces verfügen, wird die selbst entwickelte Positionsbestimmungssoftware ebenfalls in C++ implementiert.

Die Bilder der Kamera müssen von der Bildverarbeitungskomponente der Positionsbestimmungssoftware zunächst vorverarbeitet werden, um die Objekterkennung zu vereinfachen, welche bei der Bestimmung der Fahrzeugpose zu Einsatz kommt. Dabei müssen u. a. Verzerrungen des Bildes entfernt werden, welche vor allem durch das Weitwinkelobjektiv der Kamera verursacht werden [CAR14]. Durch das Herausfiltern von Störeinflüssen kann die Objekterkennung weiter optimiert werden.
Die mit Hilfe der Objekterkennung ermittelte Fahrzeugpose beinhaltet die zweidimensionalen Koordinaten des Fahrzeugs auf der Rennstrecke sowie dessen Ausrichtung, welche aus der Anordnung der Reflexionsmarker (siehe Abschnitt 4.1.2) hervorgeht. Außerdem muss die Bildverarbeitungssoftware fähig sein, aus der Anordnung der Reflexionsmarker die eindeutige Identität des Fahrzeugs zu ermitteln. Diese Funktionalität ist essentiell für die Erweiterbarkeit des Systems, da in zukünftigen Ausbaustufen mehrere Fahrzeuge gleichzeitig auf der Rennstrecke fahren sollen.

Neben den Fahrzeugen erfasst die Positionsbestimmung auch die Lage der Rennstrecke. Da die Lage bekannt ist, werden die Koordinaten der Fahrzeugpose in relative Streckenkoordinaten transformiert und in dieser Form an die Control-Unit übermittelt. Die Lage der Rennstrecke wird sowohl während der Initialisierung als auch während des gesamten Rennbetriebs erfasst. So kann auch nach der Initialisierung des Systems festgestellt werden, ob die Rennstrecke verschoben wurde.

4.2.8 Sicherheit

Das Fahrzeug fährt mit einer vergleichsweise hohen Durchschnittsgeschwindigkeit von 1,5 m/s. Daher muss es von der Positionsbestimmung mit einer entsprechend hohen Frequenz erfasst werden. Damit eine zuverlässige und ausreichend präzise Bestimmung der Fahrzeugpose erreicht wird, soll eine Mindestbildwiederholfrequenz von 100 Hz realisiert werden. 100 Hz sind ein Kompromiss zwischen dem Rechenaufwand der Bilderfassung und der Präzision bei der Bestimmung der Fahrzeugpose [Ver14]. Daher dürfen Empfang und Verarbeitung des Bildes nicht länger als 10 ms dauern.
”Ausreichend präzise” bedeutet im Rahmen der ersten Ausbaustufe des Projekts *RCCARS* lediglich, dass das Fahrzeug, dem in Kapitelabschnitt 2.4 definierten Szenario entsprechend, die Rennstrecke mit der geforderten Durchschnittsgeschwindigkeit umrundet, ohne die Fahrbahnbegrenzungen zu berühren. Zukünftige Folgeprojekte sollten an dieser Stelle spezifizieren, wie groß die Abweichung der ermittelten Position des Fahrzeugs im Vergleich zur tatsächlichen Position auf der Strecke sein darf.
Um sicherzustellen, dass diese Anforderung jederzeit eingehalten wird, muss die Software zum Einen entsprechend laufzeiteffizient implementiert werden und zum Anderen auf einem leistungsstarken Rechner ausgeführt werden. Ein *Ubuntu 14.04 LTS* [Ubu] mit dem Realzeitkernel (*3.12.31-rt45l*) [rt], welches auf dem Rechner als Realzeitbetriebssystem eingerichtet wurde, soll die notwendigen Voraussetzungen schaffen.
Durch den Einsatz von Infrarotlicht in Kombination mit einem Infrarotfilter kann die Laufzeit der Bildverarbeitungssoftware verringert werden, da nur noch helle von dunkle Pixeln unterschieden werden müssen, um die Reflexionsmarker zu erkennen. [Rut10]

Die Positionsbestimmungssoftware ist auch dafür verantwortlich, die Lage der Rennstrecke zu erfassen. Sollte sie feststellen, dass die Rennstrecke nach der Initialisierung verschoben wurde, muss sie einen Fehler melden, so dass das das Fahrzeug gestoppt und System in einen Fehlerzustand überführt wird. Die Control-Unit kann die Verschiebung der Rennstrecke nicht erkennen, da die übermittelte Fahrzeugpose auf den relativen Koordinaten der Rennstrecke basiert (siehe Kapitelabschnitt 4.2.7).

4.3 Subsystem 3: Control-Unit

Die Control-Unit ist für die Berechnung der Regelungswerte aus der von der Positionsbestimmung ermittelten Pose und die Weiterleitung dieser an das Fahrzeug zuständig. Die berechneten Regelungswerte ermöglichen die Regelung der Geschwindigkeit und Lenkung des Fahrzeugs. Wie beim Subsystem der Positionsbestimmung (siehe Kapitel 4.2) findet auch hier eine Unterteilung in Software- und Hardwarekomponenten statt.

Hardwareseitig zerfällt das Subsystem in zwei Komponenten: Einen leistungsstarken Rechner und die CarControl. Diese Komponenten werden in den Kapitelabschnitten 4.3.1 und 8.4.4.3 beschrieben. Die Software verteilt sich auf diese zwei Komponenten. Die Regelungssoftware der autonomen Fahrzeugsteuerung wird auf dem Rechner ausgeführt. Die Software für die Signalverarbeitung wird auf dem Entwicklungsboard der CarControl ausgeführt (siehe Abschnitt 4.3.3).

4.3.1 Rechner

Es wird ein leistungsstarker Rechner benötigt, welcher die Regelungssoftware der autonomen Fahrzeugsteuerung ausführt und dabei die geforderten Realzeitbedingungen (siehe Kapitelabschnitt 4.3.4) erfüllt. In der ersten Ausbaustufe des Projekts *RCCARS* wird diese Software auf demselben Rechner ausgeführt, wie die Software der Positionsbestimmung (siehe Kapitelabschnitt 4.2.7). Das bedeutet, es wird der selbe Rechner verwendet, welcher bereits in Kapitelabschnitt 4.2.6 beschrieben wurde. Der Rechner sendet die digitalen Regelungswerte der Regelungssoftware über eine USB-Schnittstelle an die CarControl, welche diese auf einem Entwicklungsboard in analoge Signale (elektrische Spannungsstärken) umwandelt und mit Hilfe der Fernsteuerung an das Fahrzeug übermittelt.

4.3.2 CarControl

Die CarControl setzt sich aus einem Entwicklungsboard und der *Kyosho* Fernsteuerung des *dNaNo*-Fahrzeugs zusammen. In diesem Zusammenspiel ist die Steuerung des Fahrzeuges über den Rechner möglich.

Für die Kommunikation mit dem Fahrzeug wird die Fernsteuerung *Perfex KT-18* [Fer] von *Kyosho* verwendet. Diese Fernsteuerung wird mit vier Batterien des Typs AAA betrieben und arbeitet mit zwei Potentiometern. Jeweils ein Potentiometer wird für die Beschleunigung und die Steuerung eingesetzt. Beide Potentiometer arbeiten in einem Spannungsbereich von 0,00 bis 3,00 Volt. Mit Hilfe dieser Spannungsbereiche ermittelt die Fernsteue-

rung den Grad der aktuellen Beschleunigung und Lenkung und sendet diese Daten über das 2,4 GHz Frequenzband an das Fahrzeug.
Da die Fernsteuerung keine direkte Schnittstelle zum Rechner bietet, wurden im Referenzprojekt der Universität Göteborg (Kapitel 2.1.4) ein *Arduino Uno* Entwicklungsboard und ein DAC-Baustein verwendet. Die internen Verbindungen der Potentiometer der Fernsteuerung wurden durchtrennt und mit den analogen Ausgängen des DAC-Bausteins verbunden. Der DAC-Baustein wurde mit digitalen Ausgängen des Entwicklungsboards verbunden. Somit wurde die Kommunikation zwischen dem Rechner und dem Fahrzeug unidirektional ermöglicht.
Im Projekt *RCCARS* wird als Entwicklungsboard das *STM32F4 Discovery* [Ent] eingesetzt. Das Entwicklungsboard zeichnet sich durch seine zahlreichen Anschluss- und Konfigurationsmöglichkeiten aus. Es wird verwendet, um digitale Regelungswerte über eine integrierte USB-Schnittstelle vom Rechner zu empfangen. Da dieses Entwicklungsboard bereits einen integrierten DA-Wandler und somit analoge Ausgänge besitzt, werden diese lediglich mit den getrennten Verbindungen der Potentiometer der Fernsteuerung verbunden. Ein zusätzlicher DAC-Baustein wird nicht benötigt. Des Weiteren liefert das Entwicklungsboard einen Gesamtausgangsstrom von 25 mA. Damit ist es in der Lage, eine Fernsteuerung anzusteuern und diese zusätzlich mit der benötigten Betriebsspannung zu versorgen. Somit können die Batterien durch eine 5,00 Volt Ausgangsspannung des Entwicklungsboards ersetzt werden.
Ähnlich wie im Referenzprojekt der Universität Uppsala (Kapitel 2.1.3) wurden die Potentiometer der Fernsteuerung mit analogen Eingängen des Entwicklungsboard verbunden. Somit ist es möglich, die Potentiometer der Fernsteuerung auszulesen und die ausgelesenen Werte bei Bedarf über die analogen Ausgänge des Entwicklungsboards wieder der Fernsteuerung bereitzustellen. Somit ist die ursprüngliche Funktionalität der Fernsteuerung wiederhergestellt und um die Funktionalität der Steuerung über den Rechner erweitert. Um die beschriebene Funktionalität mit Hilfe des Entwicklungsboards herzustellen, wird eine selbst entwickelte Software (siehe Abschnitt 4.3.3) auf dem Entwicklungsboard ausgeführt.

Im Betrieb wurde festgestellt, dass die integrierte USB-Schnittstelle des Entwicklungsboards, welche eine softwarebasierte virtuelle serielle Schnittstelle ist, instabil arbeitet. Aus diesem Grund wurde diese durch einen *FTDI Basic-Breakout* ersetzt, welcher mit dem Entwicklungsboard verbunden und eine stabilere serielle Kommunikation ermöglicht.

Das Entwicklungsboard mit dem *FTDI Basic-Breakout* bildet daher - wie in Abbildung 4.2 schematisch dargestellt - die Kommunikationsschnittstelle zwischen dem Rechner und der Fernsteuerung und somit dem Fahrzeug.

]

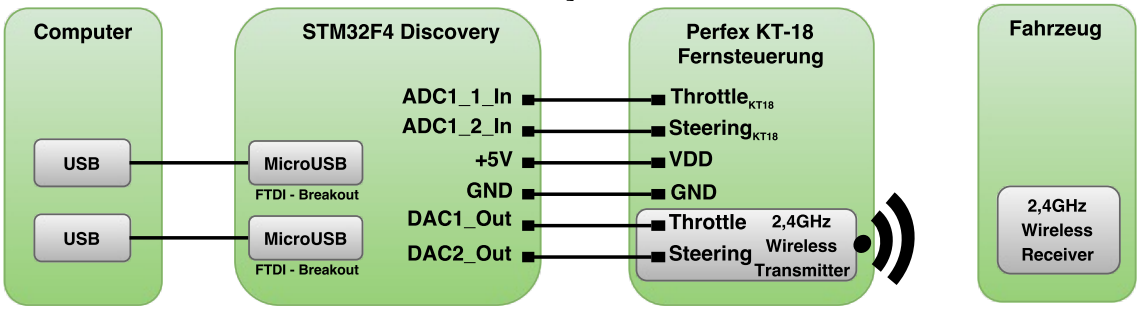


Abbildung 4.2: Schematische Darstellung der Kommunikation zwischen dem Rechner und der Fernsteuerung

4.3.3 Software

Zu den Softwarekomponenten gehört eine selbst entwickelte Software, welche auf dem Rechner ausgeführt wird. Diese wird in der Programmiersprache C implementiert. Für die Entwicklung der Regelungssoftware, steht das Werkzeug *SCADE* [SCA] zur Verfügung, welches die modellbasierte Entwicklung sicherheitskritischer Software ermöglicht.
Die Software erhält die aktuelle Pose des Fahrzeuges vom Subsystem der Positionsbestimmung (siehe Kapitelabschnitt 4.2). Da es bei der Bildererfassung möglicherweise zu Fehlern kommen kann, muss eine Plausibilitätsüberprüfung durchgeführt werden, um zu verhindern, dass Steuerungsbefehle an das Fahrzeug übermittelt werden, welche auf fehlerhaften Daten basieren. Sind die von der Positionsbestimmung erhaltenen Daten plausibel, kann durch die Auswertung von aufeinander folgender Fahrzeugposen, die Fahrtrichtung, Geschwindigkeit und Beschleunigung des Fahrzeugs berechnet werden. Anhand dieser Daten werden die Regelungsparameter für die autonome Steuerung des Fahrzeugs bestimmt.

Das Fahrzeug soll entlang einer vordefinierten Trajektorie über die Rennstrecke gesteuert werden. Diese Trajektorie ist der Control-Unit bekannt und auf genau eine spezifische Rennstrecke ausgelegt. In der ersten Ausbaustufe des Projektes *RCCARS* wird eine Trajektorie verwendet, welche das Fahrzeug in der Mitte der Fahrbahn halten soll (siehe Kapitelabschnitt 4.4.2), um die Gefahr einer Kollision mit der Fahrbahnbegrenzung zu minimieren. Trotzdem muss die Software der Control-Unit in der Lage sein, Unfälle und

Störungen zu identifizieren. So muss die Software erkennen, ob das Fahrzeug den befahrbaren Bereich der Rennstrecke verlässt, ob es mit der Streckenbegrenzung kollidiert oder ob es in die falsche Richtung fährt und ggf. einen Not-Halt des Fahrzeuges auslösen.

Darüber hinaus übermittelt die Software die berechneten Regelungsdaten an die CarControl. Diese wandelt die digitalen Regelungsdaten in analoge Signale in Form spezifischer Spannungsstärken um und leitet diese an die Fernsteuerung weiter. Zu diesem Zweck wird wiederum eine selbst entwickelte Software benötigt, welche auf dem Entwicklungsboard ausgeführt wird und diese Funktionalität bereitstellt. Diese Software wird, wie in der Mikrocontrollerprogrammierung üblich, in der Programmiersprache C implementiert. Um die Arbeit mit dem Entwicklungsboard zu erleichtern, werden fertige Bibliotheken für die Funktionalität der Umwandlung der digitalen Regelungssignale in Spannungen und ein Realzeitbetriebssystem eingesetzt.

Des Weiteren muss die Software der Control-Unit in der Lage sein, den Verlust des Sichtkontakts zum Fahrzug zu erkennen. Ein Verlust des Sichtkontakts liegt vor, sobald die Positionsbestimmung nach zwei Kamerabildern in Folge die Pose des Fahrzeugs nicht ermitteln konnte und entsprechende Fehlerdaten an die Control-Unit übermittelt. Dies kann durch einen Überschlag des Fahrzeugs verursacht werden, welcher zur Folge hat, dass das Fahrzeug auf dem Dach liegt und die Reflexionsmarker nicht sichtbar sind oder dadurch, dass das Fahrzeug das Sichtfeld der Kamera verlassen hat. Gleiches gilt für den Fall, dass die Erfassung des Fahrzeugs durch störende Objekte oder Hardwaredefekte beeinträchtigt wird. Außerdem muss die Software fähig sein, den Verbindungsverlust zur CarControl festzustellen.

4.3.4 Sicherheit

Um eine zuverlässige und ausreichend präzise autonome Fahrzeugsteuerung zu erreichen, soll die Regelung mit einer Wiederholrate von mindestens 100 Hz ausgeführt werden. Die Einhaltung dieser Realzeitanforderung soll durch eine laufzeiteffiziente Implementierung der Regelungssoftware und einen leistungsstarken Rechner mit Realzeitbetriebssystem erreicht werden (siehe auch Abschnitt 4.2.8).
”Ausreichend präzise” bedeutet hier wiederum, dass das Fahrzeug die Rennstrecke, wie in der Beschreibung in Kapitelabschnitt 2.4 festgelegt, umrundet, ohne die Fahrbahnbegrenzungen zu berühren. Sicherheitskritisch ist auch die Erkennung von Unfällen und Störungen, welche in Abschnitt 4.3.3 erläutert wurde.

Durch die Verwendung von *SCADE*, welches mit Hilfe eines Programmcodegenerators zertifizierbaren Programmcode erzeugen kann, soll sichergestellt werden, dass der aus den Modellen automatisch generierte Programmcode für die Regelungsalgorithmen korrekt ist. Außerdem kann die Funktionsweise der mit *SCADE* erzeugten Regelungsmodelle in der Entwicklungsumgebung simuliert werden.

Wie in Kapitelabschnitt 4.3.3 beschrieben, empfängt die Software der Control-Unit die Pose des Fahrzeugs von der Software der Positionsbestimmung. Kann die Positionsbestimmung die Pose des Fahrzeugs in zwei aufeinander folgenden Kamerabildern, was einer Zeit von maximal 20 ms entspricht, nicht feststellen, muss die Control-Unit den Rennbetrieb einstellen, da das Fahrzeug, ausgehend von einer Geschwindigkeit von 1,5 m/s, drei Zentimeter unkontrolliert zurück gelegt hat. Selbst wenn das Fahrzeug im darauf folgenden Kamerabild wieder erkannt wird, könnte eine Kollision mit der Streckenbegrenzung nicht mehr vermieden werden, falls sich das Fahrzeug in einer Kurve befindet. Daher muss das Fahrzeug spätestens nach 20 ms gestoppt werden, falls es nicht erfasst wird.

4.4 Subsystem 4: Rennstrecke

Da das im Projekt *RCCARS* entwickelte System erweiterbar sein soll, muss auch die Rennstrecke verändert oder ausgetauscht werden können. Daher ist die Rennstrecke ein eigenständiges Subsystem, auch wenn sie im Rahmen der ersten Ausbaustufe des Projekts unverändert bleiben soll. Das Subsystem Rennstrecke besteht aus der Fahrbahn und den Fahrbahnbegrenzungen. Die Fahrbahn definiert den befahrbaren Bereich, während die Fahrbahnbegrenzungen diesen von der Umgebung - dem nicht befahrbaren Bereich - abgrenzen.

4.4.1 Bauweise

Es wird die Rennstrecke *Mini-Z Grand Prix Circuit 30* der Firma *Kyosho* [KYO] verwendet. Dies ist eine Modellrennstrecke aus Schaumstoff, welche für die Nutzung von *dNaNo*-Fahrzeugen im Indoor-Bereich konzipiert wurde. Die Rennstrecke besteht aus 48 einzelnen Streckensegmenten, die nach belieben zusammengesetzt werden können. Diese modulare Bauweise ermöglicht es, unterschiedliche Rennstrecken mit denselben Bauteilen zu konstruieren.

4.4.2 Streckenführung

Die Abbildung 4.3 zeigt eine schematische Darstellung der Strecke mit zentralem Blick von oben. Dies entspricht der Sicht der Kamera (siehe Abschnitt 4.2.5). Der geschlossene Streckenverlauf besteht aus zwei 1,2 m langen Geraden und fünf Kurven. Es gibt vier 90°-Kurven und eine 180°-Haarnadelkurve, welche sich aus zwei direkt aufeinander folgenden 90°-Kurven zusammensetzt. Beim Fahren auf der Mittellinie ist eine 90°-Kurve etwa 47,1 cm lang. Hinzu kommen drei kurze Verbindungsgeraden zwischen den Kurven, mit Längen von 30 cm, 60 cm und wiederum 30 cm. Die Länge einer Runde, beim Fahren in der Mitte der Fahrbahn, beträgt bei zwölf Geradensegmenten und sechs 90°-Kurvensegmenten ca. 6,427 m (siehe Rechnung).
Die Rennstrecke soll gegen den Uhrzeigersinn umrundet werden.

$$r_{lm} = 12 \cdot 0,3\,m + 6 \cdot \frac{1}{4} \cdot 2\pi \cdot 0,3\,m \tag{4.5}$$

$$= 3,6\,m + \frac{9 \cdot \pi}{10}\,m \tag{4.6}$$

$$\approx 6,427\,m \tag{4.7}$$

4.4.3 Streckensegmente

Es gibt drei unterschiedliche Arten von Streckensegmenten (siehe Abbildung 4.3 rechts): Die Segmente der Geraden und die beiden Kurvensegmente (innen und außen). Auf der Außenseite aller Streckensegmente ist eine 1,5 cm hohe und 4,8 cm breite Schaumstoffschicht montiert, welche verhindern soll, dass das Fahrzeug bei einem Unfall oder einem Fehler der autonomen Fahrzeugsteuerung die Fahrbahn verlässt. Die Segmente der Geraden sind 32 cm lang. Im Hinblick auf die Gesamtlänge der Rennstrecke entfallen je 2 cm pro Segment aufgrund der 2 cm breiten Verbindungsbereiche.
Der modulare Aufbau der Rennstrecke aus einzelnen Streckensegmenten ist langfristig von Bedeutung, da mögliche Folgeprojekte gegebenenfalls eine variable Streckenführung unterstützen möchten. Im Rahmen dieser Projektgruppe soll jedoch der in Abbildung 4.3 gezeigte Aufbau beibehalten werden, da die wichtigste Aufgabe zunächst darin besteht, die grundlegenden Funktionen der Regelungssoftware zu implementieren und eine zuverlässige sowie ausreichend präzise autonome Fahrzeugsteuerung zu realisieren.

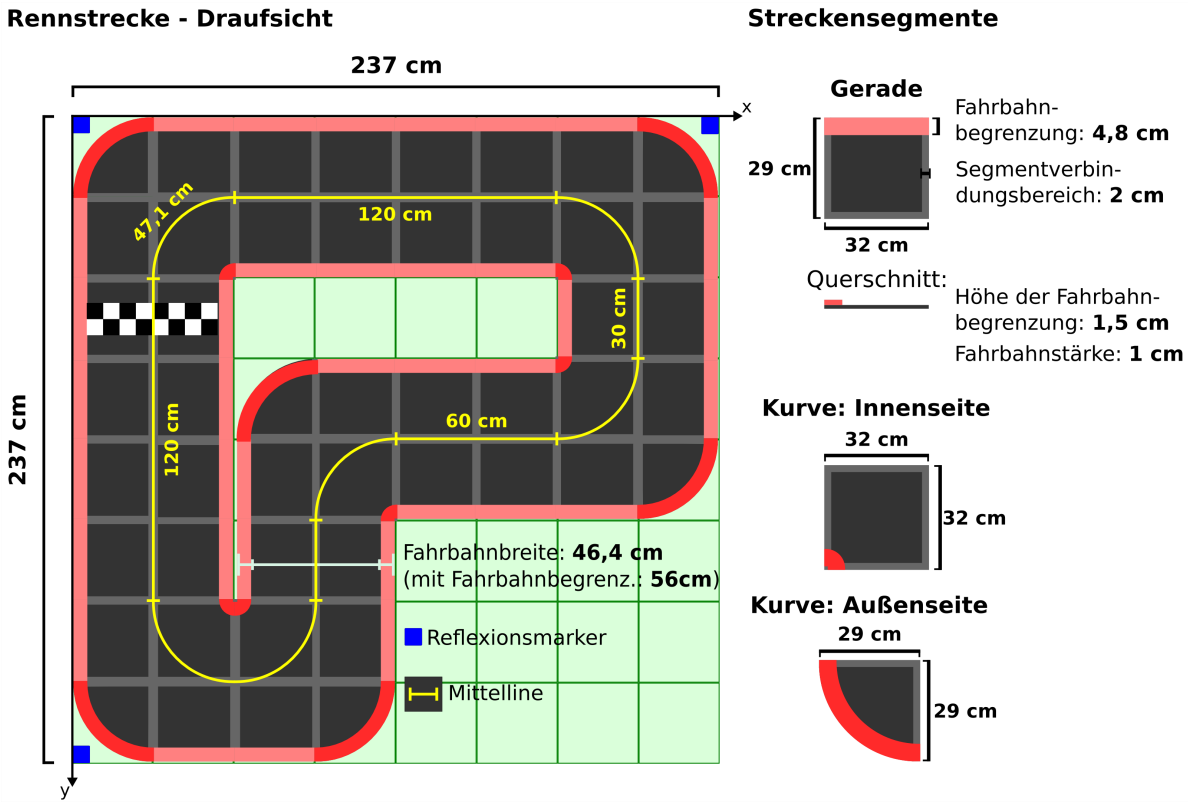


Abbildung 4.3: Schematische Darstellung der im Projekt *RCCARS* verwendeten Rennstrecke und der einzelnen Streckensegmente mit Maßangaben.

4.4.4 Erfassung der Rennstrecke

In der dargestellten Konfiguration besteht die Rennstrecke aus allen 48 Segmenten. Gemäß der in Kapitelabschnitt 4.4.2 genannten Maße der einzelnen Streckensegmente, müsste die gesamte Rennstrecke von einer Minimum Bounding Box mit einer Größe von 5,617 m^2 überdeckt werden. Aufgrund der Flexibilität des Schaumstoffmaterials ist jedoch zu beachten, dass es durch Stauchungen oder Streckungen zu leichten Abweichungen (< 1 cm) kommen kann. Die größten Abweichungen treten bei den langen Geraden auf, welche aus acht in Reihe liegenden Segmenten bestehen. Diese Varianzen sind bei der Implementierung der autonomen Fahrzeugsteuerung zu berücksichtigen (siehe Abschnit 4.4.5). Die Seitenlänge der quadratischen Minimum Bounding Box muss somit um 1 cm auf 2,37 m erweitert werden (grün hinterlegter Bereich in Abbildung 4.3), damit sichergestellt ist, dass die gesamte Strecke von der Kamera erfasst wird.

Dem System muss außerdem die genaue Lage der Rennstrecke bekannt sein, damit die Regelungssoftware das Fahrzeug möglichst präzise steuern kann. Zu diesem Zweck muss beim Systemstart eine Initialisierung vorgenommen werden, bei der die Lage der Strecke erfasst wird. Um diesen Vorgang zu automatisieren, sind an den äußeren drei Kur-

ven Reflexionsmarker (siehe Kapitelabschnitt 4.1.2) angebracht (blaue Quadrate in Abbildung 4.3).
Damit die von der Positionsbestimmungssoftware ermittelte Fahrzeugpose in relative Streckenkoordinaten transformiert werden kann, bevor diese an die Control-Unit übermittelt wird (siehe Kapitelabschnitt 4.2.7), besitzt die Rennstrecke ein eigenes Koordinatensystem. Der Koordinatenursprung befindet sich in Abbildung 4.3 links oben und die verwendete Maßeinheit ist Zentimeter.

4.4.5 Sicherheit

Vor Aufnahme des Rennbetriebs muss überprüft werden, ob die Rennstrecke selbst sowie die Umgebung der Strecke, welche von der Kamera aufgenommen wird, frei von Störungen und Hindernissen ist. Es dürfen sich keine störenden Objekte im Sichtfeld der Kamera befinden und Zuschauer müssen einen Sicherheitsabstand von 1 m zur Rennstrecke einhalten. Dieser Sicherheitsabstand verhindert, dass Zuschauer mit ihrem Körper die Fahrzeuge oder die Rennstrecke berühren oder das Sichtfeld der Kamera stören. Falls ein Zuschauer während des Rennbetriebs den Sicherheitsabstand verletzt oder das System mit Hilfe störender Objekte beeinträchtigt, muss der Rennbetrieb gestoppt werden.

Aufgrund der Flexibilität der Streckensegmente müssen auch die Fahrzeuge einen Sicherheitsabstand zur Fahrbahnbegrenzung einhalten. In der ersten Ausbaustufe des Projekts *RCCARS* ist dies jedoch noch nicht relevant, da das Fahrzeug zunächst in der Mitte der Fahrbahn gesteuert werden soll, um die Gefahr einer Kollision mit der Fahrbahnbegrenzung zu minimieren.

Ein weiterer sicherheitskritischer Aspekt dieses Subsystems sind die Fahrbahnbegrenzungen selbst, welche die gesamte Fahrbahn der Rennstrecke umschließen. Sie verhindern, dass ein Fahrzeug bei einem Unfall oder einem Fehler der autonomen Steuerung den befahrbaren Bereich verlässt.

4.5 Netzwerk

In der Long-Term-Vision der Projektgruppe RCCARS (siehe Kapitelabschnitt 2.2) ist vorgesehen, dass die Positionsbestimmung, die Systemsteuerung und sämtliche Control-Units auf eigenen Rechnern ausgeführt werden können. Für dieses Vorhaben muss eine Kommunikation zwischen den Rechnern bzw. den Subsystemen realisiert werden. Hierfür bietet sich das IEEE 802.3 Ethernet an, da alle Rechner bereits vom Werk aus mit einer

Netzwerkkarte ausgestattet sind und der mögliche Datendurchsatz sehr hoch ist.
Sämtliche Rechner werden über einen Switch und Netzwerkkabel miteinander verbunden. Somit ergibt sich ein wie in Abbildung 4.4 dargestellter Systemaufbau. Mit diesem Ansatz können sämtliche Control-Unit Rechner mit dem Rechner der Positionsbestimmung und dem Rechner der Systemsteuerung kommunizieren. Außerdem ist damit die Hardware-Grundlage für die Kommunikation der einzelnen Komponenten geschaffen.

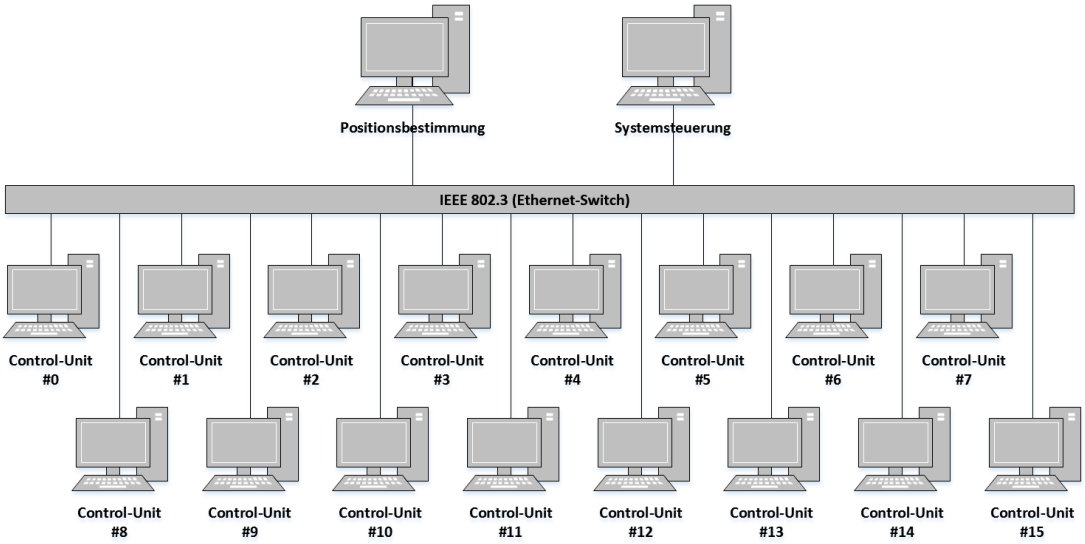


Abbildung 4.4: Schemeatische Darstellung der Netzwerkarchitektur.

Im ersten Ausbau der Projektgruppe RCCARS soll nur ein Fahrzeug betrieben werden. Unter dieser Voraussetzung wird vorerst nur eine Control-Unit benötigt. Daher wird die Control-Unit vorerst auf dem gleichen Rechner mit der Positionsbestimmung ausgeführt. Die Kommunikation soll im Hinblick auf die Erweiterbarkeit dennoch über das Netzwerk stattfinden. In Zukunft ist es denkbar, dass nicht vollwertige Desktop-Rechner für die Control-Unit eingesetzt werden, sondern ThinClients, Einplatinencomputer wie ein Raspberry Pi oder virtualisierte Rechner auf einem einzigen leistungsstarken Host-Rechner.

Die Kommunikation wird über UPD-Sockets realisiert. Hierzu werden ein UDP-Receiver für den Empfang und ein UDP-Sender zum Senden von Netzwerkdatenpaketen implementiert. Die Struktur des Netzwerkdatenpakets muss im Vorfeld festgelegt und sowohl dem Sender wie auch dem Empfänger bekannt sein. Daher werden sämtliche Variablen für die Kommunikation in einem Netzwerkdatenpaket zusammengefasst (siehe Abbildung 11.10). Anhand eines mitgeschickten Identifikators kann das Netzwerkdatenpaket als Fahrzeugdatenpaket (Identifikator = 1), Kontrolldatenpaket (Identifikator = 2), Fehlerdatenpaket (Identifikator = 3) oder Zeitdatenpaket (Identifikator = 4) identifiziert wer-

den. Zusätzlich zum Identifikator wird ein Zeitstempel mitgeschickt, welcher das Alter der jeweiligen Daten festhalten und als Fingerabdruck des Netzwerkdatenpakets dienen soll und eine Absender-ID. Je nach Identifikator bzw. Typ des Netzwerkdatenpakets werden nur bestimmte Variablen beschrieben und versendet bzw. empfangen und ausgelesen. Alle übrigen Variablen bleiben unbetrachtet. Mit Hilfe der Absender-ID ist es den einzelnen Komponenten möglich, nur Netzwerkdatenpakete von bestimmten Absendern zu betrachten. Um die Datenintegrität der Netzwerkdatenpakete zu gewährleisten, wird von den UDP-Sockets vor dem Versenden eine Prüfsumme der Daten gebildet und beim Empfangen abgeglichen. Der Vorteil an dieser Lösung ist, dass keine festen Verbindungen hergestellt werden oder gar IP-Adressen bekannt sein müssen, da Broadcasting genutzt wird. Dies spart einen größeren Organisationsaufwand, ist nachvollziehbarer und leicht erweiterbar.
Es sind keine IP-Adressen nötig, da Broadcasting genutzt wird.

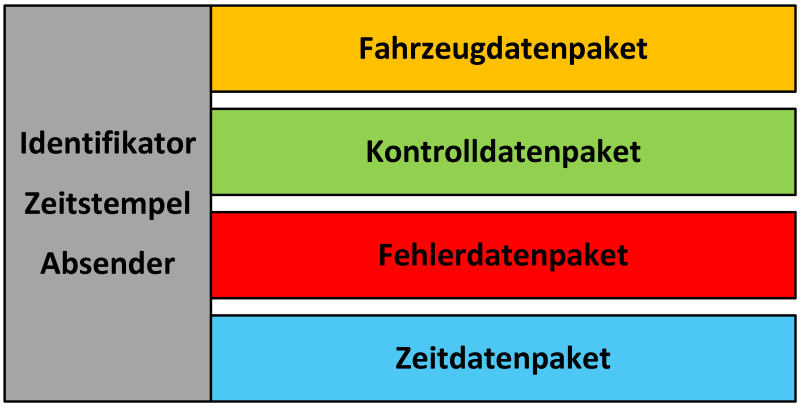


Abbildung 4.5: Schemeatische Darstellung des Netzwerkdatenpakets

Durch eine festgelegte Broadcast IP-Adresse des UDP-Senders kann entschieden werden, in welches Subnetz die Netzwerkdatenpakete gesendet werden sollen. Hinter der ”255.255.255.255” Broadcast IP-Adresse steht jedes Gerät im Netzwerk als Empfänger. Hinter der ”127.0.0.1” Broadcast IP-Adresse steht nur das Loopback-Interface bzw. der Localhost des Senders als Empfänger. Im letzteren Fall bedeutet dies, dass die Netzwerkdatenpakete die Netzwerkschnittstelle des Senders nicht verlassen. Da das Gesamtsystem unter anderem auch als Demonstrator genutzt werden soll, kann verhindert werden, dass fremde Rechner eines Netzwerks mit den Netzwerkdatenpaketen des Demonstrators geflutet werden.

4.6 Kontrollfluss

Um den Ablauf des Systembetriebs zu verdeutlichen, findet sich in Abbildung 4.6 ein Automat, welcher den Kontrollfluss während des Betriebs darstellt. Dieser deterministische endliche Automat unterteilt den Kontrollfluss in fest definierte Zustände.
Die ”Start”-Transition beschreibt das Starten der Systemsteuerungssoftware (siehe Kapitelabschnitt 4.7). Dabei wird ein Benutzerinterface gestartet und das System nimmt den Zustand ”Inspektion” an. Der Inspektionszustand kann ebenfalls aus dem Zustand ”Standby” (siehe Abbildung 4.6) betreten werden.

Unter ”Inspektion” wird das Kontrollieren des Fahrzeugs, der Rennstrecke und der Umgebung verstanden. Dazu zählt unter anderem die Sauberkeit, Trockenheit und Unversehrtheit aller Strecken- und Fahrzeugteile. Außerdem muss die Rennstrecke frei von Hindernissen jeglicher Art sein. Auch die Umgebung und der Verlauf der Rennstrecke werden überprüft. Dabei wird kontrolliert, ob alle Personen einen Sicherheitsabstand zur Rennstrecke einhalten und ob der Streckenverlauf geschlossen und eben ist. Es muss ebenfalls sichergestellt werden, das die Bilderfassung nicht durch Personen oder störende Objekte beeinträchtigt wird. Innerhalb dieses Zustands können Wartungsarbeiten, wie beispielsweise das Austauschen des Fahrzeugakkus, vorgenommen werden und die Softwarekomponenten der Positionsbestimmung und Control-Unit müssen ggf. gestartet und konfiguriert werden. Außerdem muss das Fahrzeug an der Start-Ziel-Linie in Fahrrichtung aufgestellt werden, damit das System korrekt initialisiert werden kann. Diese Tätigkeiten werden vom Administrator durchgeführt, welcher das System bedient und wartet. Um den Inspektionszustand zu verlassen, muss der Administrator über das Benutzerinterface bestätigen, dass die Inspektion durchgeführt wurde (im Automat als ”Inspektion OK” bezeichnet).

Als nächstes wird der Zustand ”Initialisierung” betreten. Die Initialisierung ist erfolgreich abgeschlossen, sobald alle Systemkomponenten für den Rennbetrieb bereit sind. Daraufhin wird zunächst der ”Standby”-Zustand erreicht und das System wartet auf das Startsignal des Administrators. Wird dieses Signal über das Benutzerinterface aktiviert, ist der Zustand ”Rennbetrieb” aktiv und das Fahrzeug setzt sich in Bewegung. Über das Stoppsignal kann der Administrator den Rennbetrieb beenden und das System wieder in den ”Standby”-Zustand versetzen.

Der Fehlerzustand wird betreten, sobald bei der Initialisierung im ”Standby”-Zustand oder im Rennbetrieb ein sicherheitskritischer Fehler auftritt. Dies ist der Fall, wenn das System selbständig einen entsprechenden Fehler erkennt oder wenn der Administrator ei-

nen manuellen Notstopp auslöst. Alle Systemzustände können jederzeit über das ”Aus”-Signal verlassen werden. Dadurch wird das System in den Endzustand versetzt und alle Softwarekomponenten des Systems werden ausgeschaltet. Befindet sich das System zuvor im Zustand ”Rennbetrieb”, müssen zunächst alle Fahrzeuge angehalten werden, bevor das System endgültig abgeschaltet wird.

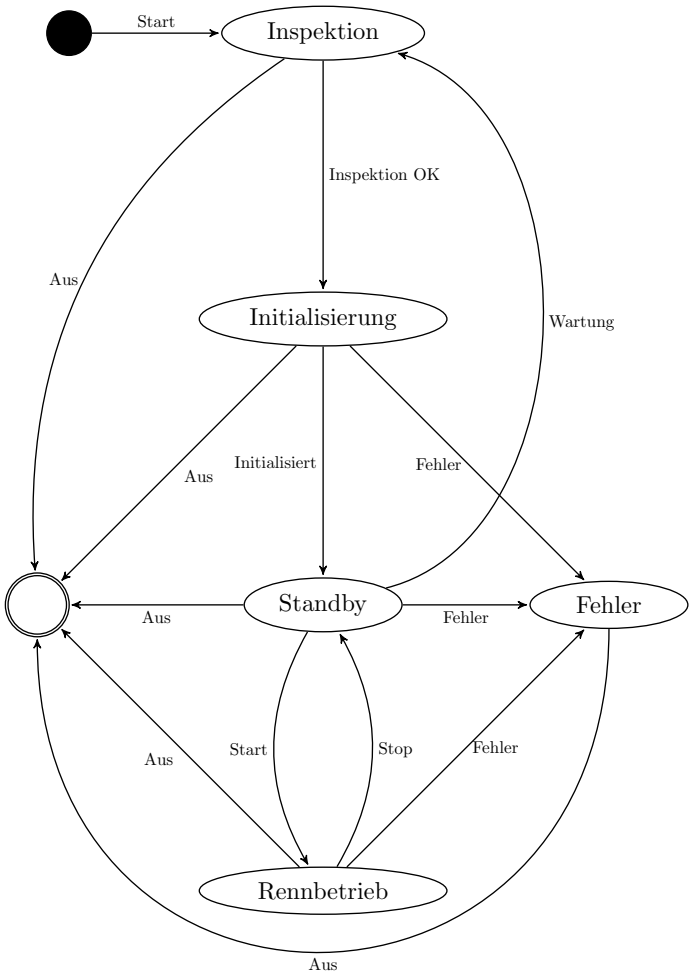


Abbildung 4.6: Die Abbildung zeigt einen deterministischen endlichen Automat, welcher den Kontrollfluss während des Systembetriebs beschreibt.

4.7 Systemsteuerungssoftware

Die Systemsteuerungssoftware stellt dem Administrator ein Interface zur Steuerung und Überwachung der Softwarekomponenten der Subsysteme Positionsbestimmung und Control-Unit zur Verfügung. Mit Hilfe der Kontrollelemente der Systemsteuerungssoftware induziert der Administrator die in Abschnitt 4.6 beschrieben Systemzustandswechsel.

Einzige Ausnahme ist das automatische Not-Aus, bei welchem das System ohne Benutzerinteraktion unmittelbar in der Fehlerzustand versetzt wird. Außerdem werden über das Benutzerinterface Systemparameter konfiguriert. Dazu gehören beispielsweise Fehlertoleranzen (siehe Anhang 14.6) und die erwartete Anzahl von Fahrzeugen, welche an einem Rennen teilnehmen sollen.

Die Systemsteuerungssoftware empfängt alle Netzwerkdatenpakete, welche von der Positionsbestimmungs und Regelungssoftware ausgegeben werden. Falls ein sicherheitskritischer Fehler gemeldet wird, versetzt sie das System unverzüglich in den Fehlerzustand. Außerdem kontrolliert sie durch die Auswertung bestimmter Netzwerkdaten, ob das Systems die Realzeitanforderungen einhält (siehe Abschnitt 4.2.8 und 4.3.4). Zu Optimierungs- und Fehlerbehandlungszwecken können Systeminformationen und Netzwerkdaten protokolliert werden. Fehlermeldungen werden generell protokolliert und gespeichert, während die Aufzeichnung von Netzwerkdatenpaketen und Informationen über den Systemzustand optional ist.

4.8 Signalfluss zwischen den Subsystemen

Das grundlegende Systemkonzept besteht darin, fortlaufend die Position des RC-Cars mittels einer Kamera zu erfassen und daraus die Steuerungsdaten für das Fahrzeug zu berechnen. Die Steuerungsdaten werden im Anschluss an das Fahrzeug übermittelt und durch dieses umgesetzt. Wenn sich die Pose des Fahrzeugs seit dem letzten Steuersignal verändert hat, wird die Position und Orientierung durch die Kamera neu erfasst. Dadurch wird ein Regelkreis gebildet, der mit seinen Komponenten und den jeweiligen Schnittstellen in Abbildung 4.7 dargestellt ist.

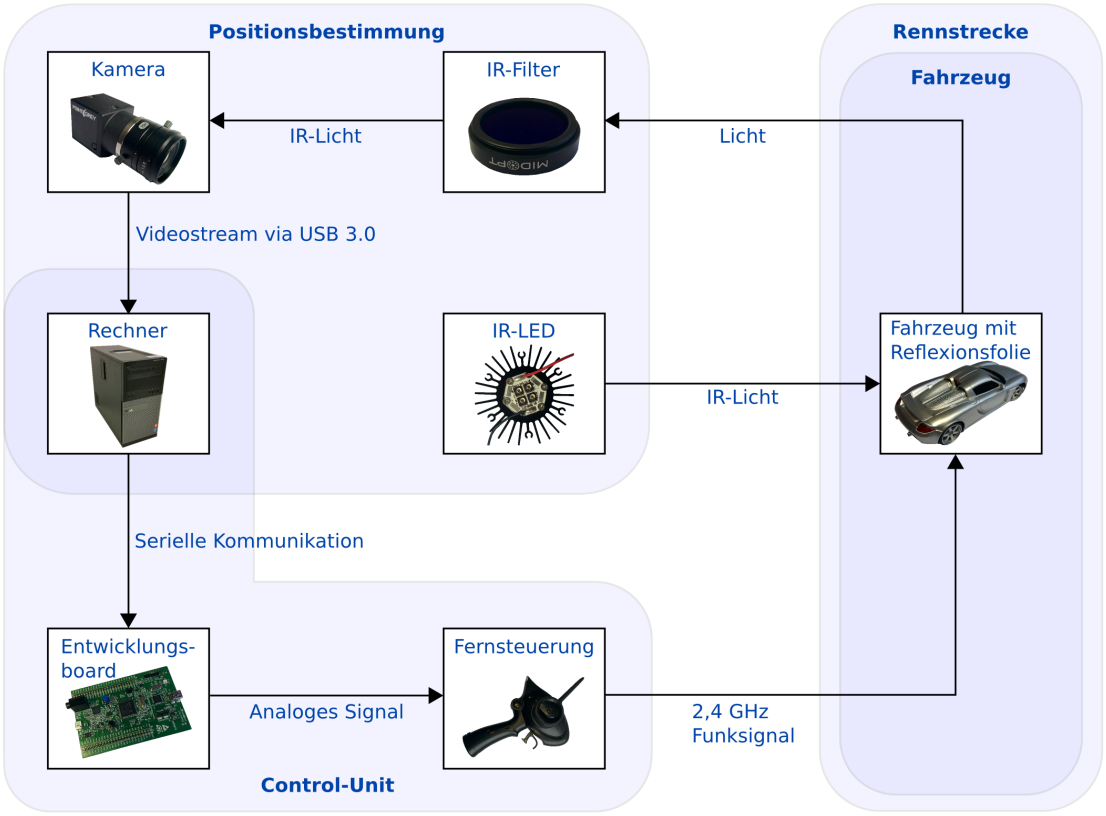


Abbildung 4.7: Schematische Darstellung des Signalflusses im Projekt *RCCARS*.

In der Abbildung sind ausgewählte Hardwarekomponenten der vier Subsysteme zu erkennen. Die Infrarotlichtquellen der Positionsbestimmung leuchten die gesamte Rennstrecke und das Fahrzeug, welches sich auf der Rennstrecke befindet, mit Infrarotlicht aus. Die Reflexionsmarker auf dem Fahrzeug reflektieren sowohl das sichtbare Licht, als auch das auftreffende Infrarotlicht. Durch den Infrarotfilter vor der Kamera wird das sichtbare Licht jedoch absorbiert, sodass hauptsächlich das Infrarotlicht auf den Kamerasensor trifft.
Die Kamera nimmt fortlaufend Bilder auf, welche über die USB 3.0 Schnittstelle an den Rechner der Positionsbestimmung übermittelt werden. Die Software des Subsystems liest die Bilder ein und bestimmt daraus die Pose des Fahrzeugs. Die ermittelte Pose wird an den Rechner der Control-Unit weitergeleitet, welcher mit diesen Daten die Regelungsparameter für die autonome Steuerung des Fahrzeugs ermittelt. Die Abbildung verdeutlicht, dass der Rechner in dieser Ausbaustufe des Projektes sowohl von der Positionsbestimmung, als auch von der Control-Unit genutzt wird.
Über eine serielle Schnittstelle übermittelt die Software der Control-Unit die an das Fahrzeug gerichteten Steuerungsdaten an das Entwicklungsboard. Die Software auf dem Mikrocontroller des Boardes wandelt die digitalen Steuerungssignale in analoge Signale um und überträgt diese wiederum an die Fernsteuerung.

Die Fernsteuerung sendet im Anschluss die entsprechenden Steuerungsbefehle per Funk auf einem 2,4 Ghz Frequenzband an das Fahrzeug, welches die Signale empfängt und diese entsprechend umsetzt.

Aus der Systembeschreibung werden in den Kapitelabschnitten 5.2.2 und 5.2.3 die spezifischen Soft- und Hardwareanforderungen abgeleitet.

5 Anforderungsspezifikation

In diesem Kapitel werden die Anforderungen spezifiziert, welche im Rahmen der ersten Ausbaustufe des Projektes *RCCARS* realisiert werden müssen. Diese Anforderungen leiten sich aus der Aufgabenstellung und dem Szenario (siehe Kapitel 2) sowie der Systembeschreibung (siehe Kapitel 4) ab. Die Anforderungen sind innerhalb dieses Kapitels ihrer Bedeutung entsprechend in Rahmenbedingungen, generelle Systemanforderungen, Hardware-, Software und nicht-funktionale Anforderungen unterteilt.
Die einzelnen Anforderungen werden mit Hilfe von Anforderungssätzen spezifiziert. Die Formulierung dieser Anforderungssätze ist so gewählt, dass eine möglichst präzise, einheitliche und einfach verständliche Definition der einzelnen Anforderungen entsteht. Die Anforderungssätze stehen in hierarchischen Beziehungen zueinander. Dies bedeutet, dass zunächst übergeordnete Basisanforderungen erhoben werden, die in Unterforderungen zerfallen, welche die übergeordnete Anforderung konkretisieren. Diese Unteranforderungen werden ihrerseits ggf. durch weitere Anforderungen präzisiert. Die Anforderungssätze sind durchnummeriert und ihrer Herkunft entsprechend kategorisiert und indiziert, so dass eindeutige Bezeichner entstehen, welche die Rückverfolgbarkeit gewährleisten und eine direkte Referenzierung einzelner Anforderungssätze ermöglichen. Des Weiteren müssen die Anforderungen konsistent und vollständig sein. Vollständig bedeutet hier, dass alle bekannten Anforderungen durch Anforderungssätze spezifiziert werden müssen. Falls während der Systementwicklung weitere Anforderungen identifiziert werden, sind diese in die Anforderungsspezifikation einzupflegen.
Die Notwendigkeit und Motivation der Anforderungen wird durch Referenzen auf die jeweiligen Abschnitte der o. g. Kapitel gestützt. Falls erforderlich, werden einzelne Anforderungssätze durch einen kurzen Text erläutert. Im Rahmen dieses Projektes werden ausschließlich Pflichtanforderungen dokumentiert (Schlüsselwort: ”muss”).

5.1 Rahmenbedingungen

Das im Projekt *RCCARS* zu entwickelnde System muss in der Lage sein, Modellfahrzeuge auf einer Rennstrecke zu erkennen und diese autonom und unfallfrei über die Strecke

zu steuern. Im Folgenden werden die Rahmenbedingungen definiert, welche dabei an das System und seine Umgebung gestellt werden. Dabei werden auch die sicherheitskritischen Voraussetzungen und Einschränkungen erfasst, deren Einhaltung für die erfolgreiche Durchführung des in Kapitelabschnitt 2.4 beschriebenen Szenarios "unfallfreie Fahrt" essentiell sind.

Vor jedem Rennen muss der Administrator des Systems eine Inspektion vornehmen, bei dem überprüft wird, ob alle Systemkomponenten für den Betrieb bereit und die im folgenden spezifizierten Rahmenbedingungen Rah1, Rah2 und Rah3 erfüllt sind. Wurde die Inspektion erfolgreich durchgeführt, darf mit der Initialisierung des Systems begonnen werden (siehe auch Kapitelabschnitt 4.6). Außerdem muss der Administrator nach der Initialisierung darauf achten, dass die Rahmenbedingungen Rah1 bis Rah4 während des Systembetriebs eingehalten werden. Falls es zu einer Verletzung der Rahmenbedingungen kommt, muss der Administrator das System stoppen.

Rah1 **Das Rennen muss unter kontrollierten Umweltbedingungen durchgeführt werden.**

Kontrollierte Umweltbedingungen bedeuten, dass das Rennen im Indoor-Bereich stattfindet. Dies ermöglicht, dass die Fahrbahn und die Fahrzeugreifen jederzeit in einem geeigneten Zustand sind, sodass eine konstante und möglichst hohe Bodenhaftung erreicht wird. Des Weiteren muss die direkte Sonneneinstrahlung auf die Minimum Bounding Box der Rennstrecke verhindert werden, da das Sonnenlicht relativ viel Infrarotlicht enthält und daher die Positionsbestimmung der Fahrzeuge stören kann.

- Rah1.1 Die Reifen der Fahrzeuge müssen sauber sein.
- Rah1.2 Die Reifen der Fahrzeuge müssen trocken sein.
- Rah1.3 Die Minimum Bounding Box der Rennstrecke darf während des Systembetriebs nicht direkter Sonneneinstrahlung ausgesetzt sein.
- Rah1.4 Die Aufhängungen der Fahrzeuge müssen sauber sein.
- Rah1.5 Die Fahrbahn der Rennstrecke muss sauber sein.
- Rah1.6 Die Fahrbahn der Rennstrecke muss trocken sein.
- Rah1.7 Die Rennstrecke muss frei von störenden Objekten sein.

Rah2 **Der Systembetrieb darf nicht durch Zuschauer gestört werden.**

Siehe Kapitelabschnitt 4.4.5 zur Sicherheit im Subsystem Rennstrecke.

Rah2.1 Alle Zuschauer müssen einen Mindestabstand von 1 m zur Minimum Bounding Box der Rennstrecke einhalten.

Rah2.2 Die Bilderfassung darf nicht durch Personen oder störende Objekte beeinträchtigt werden.

Rah3 **Der Verlauf der Rennstrecke muss unverändert bleiben.**
Die Fahrzeuge bewegen sich entlang eines vorgegebenen Kurses über die Rennstrecke. Im Rahmen der Initialisierung des Systems, bekommt die Control-Unit den Verlauf der Rennstrecke und die von den Fahrzeugen zu fahrende Route als Eingabeparameter übergeben. Diese werden erst bei einer erneuten Initialisierung aktualisiert.

Rah3.1 Der Streckenaufbau muss den Eingabeparametern für den Rennstreckenverlauf und die von den Fahrzeugen zu fahrende Route entsprechen, welche an die Control-Unit übergeben werden.

In der ersten Ausbaustufe des Systems muss der Streckenaufbau, dem in Abbildung 4.3 dargestellten Schema entsprechen (siehe Kapitelabschnitt 4.4.2 zur Streckenführung).

Rah3.2 Der Streckenaufbau darf nach der Systeminitialisierung nicht mehr verändert werden.

Rah3.3 Der Streckenverlauf muss eine geschlossene Runde bilden.

Rah3.4 Die Rennstrecke muss Streckenbegrenzungen auf beiden Fahrbahnseiten besitzen.

Dadurch wird sichergestellt, dass Fahrzeuge die Rennstrecke nicht verlassen können.

Rah3.5 Die Rennstrecke muss eben sein.

Es muss eine ebene Rennstrecke verwendet werden, da durch die zweidimensionale Bildererfassung mit einer einzelnen Kamera Höhenunterschiede im Streckenverlauf nicht zuverlässig erfasst werden können.

Rah3.6 Die Rennstrecke muss unbeschädigt sein.

Alle Segmente der Rennstrecke müssen vorhanden und eben anschließend miteinander verbunden sein.

Rah3.7 Die Rennstrecke muss eine Start-Ziel-Linie besitzen.

Im Rahmen der Inspektion müssen die Fahrzeuge vom Administrator so auf der Rennstrecke positioniert werden, dass das System korrekt initialisiert werden kann. Die Start-Ziel-Linie wird dabei als Orientierungshilfe benötigt.

Rah4 **Die Lage der Rennstrecke und die Pose der Fahrzeuge dürfen nach der Initialisierung des Systems nicht mehr manuell verändert werden.**
Das System muss bei der Initialisierung die Lage der Rennstrecke und die Pose des Fahrzeugs erfassen, da dem System die Lage der Rennstrecke bekannt sein muss, damit die Regelungssoftware das Fahrzeug wie vorgesehen entlang des vorgegebenen Streckenverlaufs steuern kann (siehe Kapitelabschnitt 4.4).

Rah4.1 Die Rennstrecke darf nach der Initialisierung nicht mehr bewegt werden.

Rah4.2 Die Hardwarekomponenten der Positionsbestimmung dürfen nach der Initialisierung nicht mehr bewegt werden.

Rah4.3 Die Fahrzeuge dürfen nach der Initialisierung nicht mehr manuell bewegt werden.

5.2 Funktionale Anforderungen

Im Folgenden werden die Funktionalitäten, die das System aufweisen muss, benannt und erläutert. Diese werden unterteilt in die Kategorien ”generelle Systemanforderungen”, ”Softwareanforderungen” und ”Hardwareanforderungen”.

5.2.1 Generelle Systemanforderungen

Die generellen Systemanforderungen umfassen allgemeine Anforderungen an das System, welche sich direkt aus der Aufgabenstellung (siehe Kapitelabschnitt 2.3) und dem Szenario der ersten Ausbaustufe (siehe Kapitelabschnitt 2.4) ergeben. Aus diesen Anforderungen leiten sich die spezifischen Hardware- bzw. Softwareanforderungen (siehe Kapitelabschnitt 5.2.2 und 5.2.3) ab.

Sys1 **Das System muss fähig sein, mindestens ein Fahrzeug autonom über eine vorgegebene Rennstrecke zu steuern.**

Dies ist die grundlegende Basisanforderung für das gesamte System der ersten Ausbaustufe. Die folgenden Unteranforderungen entsprechen der Einteilung des Systems in vier Subsysteme, so wie diese in Kapitel 4

beschrieben wurden. Da der Verlauf der Rennstrecke und die von den Fahrzeugen zu fahrende Route jeweils bei der Systeminitialisierung festgelegt wird (siehe Rahmenbedingung Rah3), wird von einer vorgegebenen Rennstrecke gesprochen.

Sys1.1 Das System muss mindestens ein ferngesteuertes Fahrzeug besitzen.

Es werden ferngesteuerte Modellrennwagen der *dNaNo*-Plattform [KYO] verwendet, welche einen Maßstab von 1:43 haben. Aus dieser Anforderung leitet sich die Hardwarebasisanforderung HW1 ab.

Sys1.2 Das System muss Komponenten besitzen, welche fähig sind, die Pose der Fahrzeuge zu bestimmten.

Relevante Parameter für die Pose sind die lokalen Koordinaten und die Ausrichtung der Fahrzeuge. Für die Erfassung der Pose wird eine Kamera in Kombination mit Infrarotlicht verwendet. Aus dieser Anforderung leitet sich die Hardwarebasisanforderung HW2 und die Softwarebasisanforderung SW1 ab.

Sys1.3 Das System muss Komponenten besitzen, welche fähig sind, mindestens ein Fahrzeug autonom über eine vorgegebenen Rennstrecke zu steuern.

Die autonome Fahrzeugsteuerung wird ausschließlich auf Grundlage der vom System selbstständig ermittelten Fahrzeugparameter durchgeführt. Aus dieser Anforderung leitet sich die Hardwarebasisanforderung HW3 sowie die Softwarebasisanforderung SW2 ab. Außerdem ergeben sich an dieser Stelle die folgenden zwei Unteranforderungen.

Sys1.3.1 Das System muss die autonome Fahrzeugsteuerung auf Grundlage der selbstständig ermittelten Fahrzeugparameter vornehmen.

Sys1.3.2 Eine manuelle Korrektur der Pose der Fahrzeuge sowie der Fahrzeugsteuerung darf nicht vorgenommen werden.

Sys1.4 Das System muss eine Rennstrecke besitzen.

Die Rennstrecke muss die Rahmenbedingung Rah3 inkl. aller Unteranforderungen erfüllen. Aus dieser Anforderung leitet sich die Hardwarebasisanforderung HW4 ab.

Sys2 **Das System muss durch einen Administrator bedient werden können.**

Aus dem Szenario der ersten Ausbaustufe (Kapitelabschnitt 2.4) ergibt sich, dass das System einen Administrator benötigt, welcher das System

bedient. Diese Basisanforderung zerfällt in die folgenden Unteranforderungen. Außerdem leitet sich aus dieser Anforderung die Softwarebasisanforderung SW3 und die Hardwareanforderung HW5.4 ab.

Sys2.1 Das System muss einem Administrator die Möglichkeit bieten, die Inspektion des Systems zu bestätigen.

Sys2.2 Das System muss einem Administrator die Möglichkeit bieten, die Initialisierung des Systems zu starten.

Sys2.3 Das System muss durch einen Administrator jederzeit ausgeschaltet werden können.

Sys2.4 Das System muss einem Administrator die Möglichkeit bieten, den Rennbetrieb zu starten.

Sys2.5 Das System muss einem Administrator die Möglichkeit bieten, den Rennbetrieb jederzeit zu unterbrechen.

Sys2.6 Das System muss einem Administrator die Möglichkeit bieten, den Systembetrieb manuell zu stoppen.

Diese Notstoppfunktion soll einem Administrator ermöglichen, das System nach eigenem Ermessen in den Fehlerzustand zu versetzen, falls sicherheitskritische Problemsituationen nicht automatisch erkannt werden.

Sys2.7 Das System muss einem Administrator ein Benutzerinterface bereitstellen.

Sys3 **Das System muss fähig sein, die Fahrzeuge mit einer Durchschnittsgeschwindigkeit von mindestens 1,5 m/s über eine vorgegebenen Rennstrecke zu steuern.**

Nach Vergleich mit den in Kapitelabschnitt 2.1 genannten Referenzprojekten und eigenen empirischen Erhebungen wurde festgestellt, dass eine Geschwindigkeit zwischen einem und zwei Metern pro Sekunde realistisch für das erste Szenario (siehe Kapitelabschnitt 2.4) ist. Die 1,5 m/s leiten sich daraus ab, dass eine Geschwindigkeit von 1 m/s für ein autonomes Rennszenario als zu langsam erachtet wurde, da diese Geschwindigkeit auch durch eine manuelle Steuerung problemlos erreicht werden kann. Eine Durchschnittsgeschwindigkeit von 2 m/s hingegen wird für das erste Szenario als zu riskant angesehen, um ein Fahrzeug unfallfrei um die engen Kurven der Rennstrecke zu lenken. Diese Beurteilung basiert auf den Daten des Referenzprojektes der Universität Uppsala [SSW+14]. Dort wurde als Ergebnis der ersten Ausbaustufe zwar

eine Durchschnittsgeschwindigkeit von etwa 2 m/s erreicht, jedoch verlor das Fahrzeug in Kurven die Bodenhaftung und konnte somit nicht sicher gesteuert werden. Aus dieser Anforderung leiten sich die Softwareanforderungen SW2.1.6 und SW2.1.7 ab.

Sys4 **Das System muss fähig sein, die Fahrzeuge unfallfrei über eine vorgegebene Rennstrecke zu steuern.**

Da es sich bei dem Projekt *RCCARS* um ein sicherheitskritisches System handelt, fordert das Szenario aus Kapitelabschnitt 2.4, dass ein Fahrzeug zum Abschluss der ersten Ausbaustufe autonom gefahren werden muss, ohne dass es zu Unfällen kommt. Daraus ergeben sich die folgenden Anforderungen.

Sys4.1 Das System muss fähig sein, die Fahrzeuge ohne Berührung der Fahrbahnbegrenzung über die Rennstrecke zu steuern.

Aus dieser Anforderung leitet sich die Softwareanforderung SW2.1.10.2 ab.

Sys4.2 Das System muss fähig sein, die Fahrzeuge so zu steuern, dass es die Rennstrecke nicht verlässt.

Aus dieser Anforderung leitet sich die Softwareanforderung SW2.1.10.1 ab.

Sys4.3 Das System muss fähig sein, die Fahrzeuge so zu steuern, dass die Räder nicht den Kontakt zur Fahrbahn verlieren.

Dies bedeutet, dass es ebenfalls nicht zu Unfällen kommen darf, bei dem sich die Fahrzeuge überschlagen oder auf die Seite kippen. Aus dieser Anforderung leitet sich die Softwareanforderung SW2.1.10.4 ab.

Sys4.4 Das System muss fähig sein, die Fahrzeuge mindestens fünf Runden lang fahren zu lassen.

Da auch die vorangegangen Anforderungen erfüllt sein müssen, bedeutet dies, dass das System fähig sein muss, die Fahrzeuge autonom und unfallfrei mit der geforderten Mindestdurchschnittsgeschwindigkeit mindestens fünf Runden lang über die vorgegebene Rennstrecke zu steuern. Aus dieser Anforderung leitet sich die Softwareanforderung SW2.1.8 ab.

Sys4.5 Falls das System einen Unfall autonom erkennt, muss es den Rennbetrieb stoppen.

Aus dieser Anforderung leiten sich die Softwareanforderungen SW1.6 und SW2.1.10 ab.

Sys4.6 Das System darf von der Aufnahme eines Kamerabildes bis zur Übermittlung der autonom berechneten Steuerungsbefehle maximal 30 ms benötigen.

Dies ist die Gesamtrealzeitanforderung an das System, welche benötigt wird, da das System fähig sein muss möglichst schnell und mit einer möglichst hohen Frequenz zu arbeiten, um das Fahrzeug sicher autonom steuern zu können. Der Wert von 30 ms wurde gewählt, da bei einer Geschwindigkeit von 1,5 m/s nach 30 ms bereits eine Distanz von 4,5 cm zurückgelegt wurde. Bedingt durch die Ausmaße der verwendeten Rennstrecke (siehe Kapitelabschnitt 4.4.2) könnten noch größere Verzögerungen zu Kollisionen mit der Fahrbahnbegrenzung führen.
Referenzprojekte [Ver14, EGH+15, SSW+14] haben gezeigt, dass für die Erfassung der Fahrzeugpose eine Frequenz von 100 Hz empfehlenswert ist. Ein einzelner Positionsbestimmungsschritt benötigt somit 10 ms. Da das System auch eine Störungstoleranz besitzen soll, muss es fähig sein, den Verlust einzelner Bilder bzw. Fahrzeugposen zu bewältigen, so dass zusätzliche 10 ms hinzukommen. Weitere 10 ms werden für die autonome Berechnung der Steuerungsbefehle und mögliche Verzögerungen bei der Datenübermittlung zwischen den Subsystemen veranschlagt.
Aus dieser Anforderung werden die Softwareanforderungen SW1.4 und SW2.3 an die Positionsbestimmungssoftware und die Softwarekomponenten der Control-Unit abgeleitet sowie die Hardwareanforderung HW5.1 an den Rechner.

Sys5 **Das System muss erweiterbar sein.**
Die Erweiterbarkeit des Systems soll durch einen modularen Aufbau erreicht werden. Dies ist essentiell für die iterative Weiterentwicklung des Systems (siehe Kapitelabschnitt 2.2), welche ggf. auch den Austausch und die Ergänzung von Systemkomponenten erfordert. Ohne diese Anforderung an die Erweiterbarkeit würde möglicherweise ein System entwickelt werden, welches nach Abschluss der Projektgruppe größtenteils wertlos wäre. Daher handelt es sich bei dieser Basisanforderung und den folgenden Unteranforderungen um funktionale Anforderungen.

Sys5.1 Das System muss Voraussetzungen schaffen, um mehrere Fahrzeuge autonom zu steuern.

Sys5.2 Das System muss fähig sein, unterschiedliche Fahrzeuge zu identifizieren.

Die Fahrzeuge werden mit Hilfe von Reflexionsmarkern durch die Positionserkennung erfasst. Damit mehrere unterschiedliche Fahrzeuge eindeutig identifiziert werden können (siehe Hardwareanforderung HW1.4.2), müssen sie verschiedene Muster aus Reflexionsmarkern besitzen.

Sys5.3 Das System muss durch einen modularen Aufbau die Voraussetzungen schaffen, dass die autonome Steuerung der Fahrzeuge zu einem späteren Zeitpunkt optimiert werden kann.

Durch den modularen Aufbau des Systems soll gewährleistet werden, dass nicht nur Hardwarekomponenten ersetzt werden können, sondern auch Softwarekomponenten. Dadurch sollen beispielsweise Voraussetzungen geschaffen werden, um die verwendete Rennstrategie durch eine andere zu ersetzen oder Regelungsparameter zu optimieren. Dies kann z. B. durch die Bereitstellung von Interfaces für die Rennstrategie erreicht werden, sodass die Implementierung einer kooperativen Rennstrategie, welche von der Long Term Vision gefordert wird, in das System integriert werden kann, ohne dass grundlegende Änderungen an der Software vorgenommen werden müssen.

5.2.2 Hardwareanforderungen

Die in diesem Kapitelabschnitt aufgeführten Anforderungssätze spezifizieren die Anforderungen, welche an die Hardware des Systems gestellt werden. Diese Hardwareanforderungen leiten sich aus den entsprechenden Abschnitten der Systembeschreibung (Kapitel 4) ab. Die Basisanforderungen sind den Subsystemen entsprechend organisiert (Anforderungen HW1 bis HW4). Dazu ergänzen sich generelle Hardwareanforderungen, welche sich auf das gesamte System beziehen (Anforderung HW5).

HW1 **Die Hardware des Subsystems 1 muss aus mindestens einem ferngesteuerten Fahrzeug bestehen.**

Diese Basisanforderung ergibt sich direkt aus der Systemanforderung Sys1.1. Die folgenden Unteranforderungen spezifizieren die geforderten Eigenschaften und Fähigkeiten der Fahrzeuge, welche sich aus Kapitelabschnitt 4.1 der Systembeschreibung ableiten.

HW1.1 Die Fahrzeuge müssen aus austauschbaren Komponenten bestehen.

Die Reifen, der Akku und die Aufhängungen müssen austauschbar sein, da es sich um Verschleißteile handelt. Außerdem bietet ein modularer Aufbau aus austauschbaren Komponenten die Möglichkeit, die Rennperformance der Fahrzeuge zu optimieren.

HW1.2 Die ferngesteuerte Fahrzeuge müssen vom Typ *dNaNo* sein.

Die Verwendung von RC-Cars der *dNaNo*-Plattform des Herstellers *Kyosho* [KYO] ist eine grundlegende Vorgabe im Projekt *RCCARS*. Die in Kapitelabschnitt 2.1 beschriebenen Referenzprojekte haben gezeigt, das sich der Einsatz dieser Fahrzeuge bewährt hat. Wichtig ist dabei auch die Vergleichbarkeit des zu entwickelnden Systems mit diesen Referenzprojekten.

HW1.3 Die Fahrzeuge müssen korrekt funktionieren.

Alle Komponenten der Fahrzeuge müssen korrekt funktionieren, da die Fahrzeuge sonst nicht wie vorgesehen gesteuert werden können. Falls vor dem Rennen ein Defekt festgestellt wird, darf der Rennbetrieb nicht gestartet werden (siehe auch Sicherheitscheck in Abschnitt 5.1). Falls während des Rennens ein Defekt auftritt, muss das Rennen abgebrochen werden. Relevant für das korrekte Funktionieren sind auch die Akkuleistung und die Vollständigkeit der Fahrzeuge.

HW1.3.1 Der Akku der Fahrzeuge muss fähig sein, die Fahrzeuge mindestens fünf Runden lang zu steuern, ohne dass ein für die geforderte Mindestdurchschnittsgeschwindigkeit relevanter Leistungsverlust auftritt.

Abgeleitet aus den Anforderungen Sys3 und Sys4.4 zur Durchschnittsgeschwindigkeit und Unfallfreiheit.

HW1.3.2 Die Fahrzeuge müssen vollständig sein.

Alle Komponenten der Fahrzeuge müssen vollständig montiert sein. Falls z. B. ein Reifen fehlt, kann das entsprechende Fahrzeug nicht wie vorgesehen gesteuert werden und das Rennen darf nicht gestartet werden. Falls während des Rennens ein Teil eines Fahrzeugs verloren geht, muss das Rennen abgebrochen werden.

HW1.4 Die Fahrzeuge müssen mit Reflexionsmarkern ausgestattet sein.

Die Verwendung von Reflexionsmarkern in Kombination mit Infrarotlicht erleichtert die Erfassung und Identifizierung der Fahrzeu-

ge (siehe Kapitelabschnitt 4.2 zum Subsystem der Positionsbestimmung), da durch die Aufbereitung der Kamerabilder ein schwarzes Bild mit weißen Punkten (den Markern) erzeugt wird. Die Reflexionsmarker müssen die folgenden Anforderungen erfüllen, welche aus den Kapitelabschnitten 4.1, 4.1.2 und 4.2.1 (Fahrzeug, Reflexionsfolie und Infrarotlampe) der Systembeschreibung hervorgehen.

HW1.4.1 Die Reflexionsmarker müssen so angeordnet sein, dass die Pose der Fahrzeuge erkennbar ist.

Die Anordnung der Reflexionmarker muss so gewählt werden, dass die Front und das Heck des jeweiligen Fahrzeugs auf den aufbereiteten Bildern identifiziert werden kann, damit die Pose des Fahrzeug von der Positionsbestimmung erkannt werden kann.

HW1.4.2 Die Reflexionsmarker müssen so angeordnet sein, dass die Fahrzeuge eindeutig identifiziert werden können.

HW1.4.3 Die Reflexionsmarker müssen groß genug sein, um von der Positionsbestimmung zuverlässig erkannt zu werden.

Laut Referenzprojekt der ETH Zürich [Rut10] beträgt die Mindestgröße 1 cm · 1 cm (siehe Kapitelabschnitt 4.1.2).

HW1.4.4 Die Reflexionsfolie muss Infrarotlicht reflektieren können.

HW1.4.5 Die Reflexionsfolie muss das Licht unabhängig von der Pose der Fahrzeuge, ausreichend stark reflektieren.

Ausreichend stark bedeutet hier, dass die Positionsbestimmungssoftware fähig ist, die Reflexionsmarker zuverlässig zu erfassen.

HW1.4.6 Die Reflexionsfolie muss das Licht direkt in die Richtung der Lichtquelle zurück reflektieren.

HW2 **Die Hardware des Subsystems 2 muss alle erforderlichen Komponenten für die Bestimmung der Fahrzeugposen beinhalten.**

Diese Basisanforderung ergibt sich direkt aus der Systemanforderung Sys1.2. Die folgenden Unteranforderungen spezifizieren die geforderten Eigenschaften und Fähigkeiten der Hardwarekomponenten des Subsystems Positionsbestimmung, welches in Kapitelabschnitt 4.2 der Systembeschreibung erläutert wurde.

HW2.1 Das Subsystem 2 muss einen Rechner zur Ausführung der Software der Positionsbestimmung besitzen.

Die Anforderungen an den im Projekt *RCCARS* verwendeten Rechner und die Software der Positionsbestimmung sind unter HW5 und SW1 spezifiziert.

HW2.2 Das Subsystem 2 muss eine Beleuchtungseinheit besitzen.

Folgende Anforderungen gehen aus Kapitelabschnitt 4.2.1 der Systembeschreibung hervor.

HW2.2.1 Die Beleuchtungseinheit muss eine Infrarotlichtquelle besitzen.

HW2.2.2 Die Infrarotlichtquelle muss Licht mit einer Wellenlänge von mindestens 780 nm ausstrahlen.

HW2.2.3 Die Infrarotlichtquelle muss durch einen Kühlkörper gekühlt werden.

Ohne Kühlkörper kann die Infrarotlichtquelle durch die entstehende Hitze zerstört werden.

HW2.2.4 Die Infrarotlichtquelle muss ein Netzteil besitzen, welches diese mit Energie versorgt.

HW2.2.5 Die Infrarotlichtquelle muss fähig sein, die gesamte Rennstrecke mit Infrarotlicht auszuleuchten.

HW2.3 Das Subsystem 2 muss Hardwarekomponenten für die Bilderfassung besitzen.

Folgende Anforderungen gehen aus den Kapitelabschnitten 4.2.4, 4.2.3, 4.2.2 und 4.1.2 der Systembeschreibung hervor.

HW2.3.1 Das Subsystem 2 muss eine Kamera besitzen.

HW2.3.1.1 Die Kamera muss eine Framerate von mindestens 100 Hz unterstützen.

HW2.3.1.2 Die Kamera muss die Rennstrecke zentral von oben filmen.

Die Kamera muss oberhalb des Zentrums der Rennstrecke installiert werden, damit sich perspektivische Verzerrungen möglichst gering auswirken. Dadurch können auch die Reflexionsmarker möglichst großflächig erfasst und somit zuverlässiger erkannt werden.

HW2.3.1.3 Die Kamera muss die Minimum Bounding Box der Rennstrecke vollständig erfassen und dabei die auf dem Fahrzeug befindlichen Reflexionsmarker zuverlässig erkennen können.

Die Minimum Bounding Box der im Projekt *RCCARS* verwendeten Rennstrecke hat eine Größe von 5,617 m^2 (siehe Kapitelabschnitt 4.4.4). Die verwendete Kamera muss nicht nur die gesamte Minimum Bounding Box erfassen können, sondern auch eine ausreichend hohe Auflösung besitzen, um die Reflexionsmarker zuverlässig zu erfassen. Die in den Referenzprojekten verwendete Kamera *Flea 3* des Herstellers *Point Grey* [Kam] bietet - in Verbindung mit einem Weitwinkelobjektiv (Anforderung HW2.3.2) - diesbezüglich eine ausreichend hohe Auflösung und erfüllt auch Anforderung HW2.3.1.1.

HW2.3.1.4 Die Kamera muss über einen monochromen Sensor verfügen.

Die Verwendung einer monochromen Kamera in Verbindung mit Infrarotlicht und Reflexionsmarkern beschleunigt die Bildverarbeitung, da lediglich zwischen hellen und dunklen Bereichen unterschieden werden muss.

HW2.3.1.5 Die Kamera muss fähig sein, Infrarotlicht zu erfassen.

HW2.3.2 Das Subsystem 2 muss ein Weitwinkelobjektiv besitzen.

Aufgrund der begrenzten Raumhöhe, muss ein Weitwinkelobjektiv verwendet werden, um die gesamte Rennstrecke erfassen zu können.

HW2.3.2.1 Das Weitwinkelobjektiv muss kompatibel zur Kamera sein.

HW2.3.3 Das Subsystem 2 muss einen Infrarotfilter besitzen.

HW2.3.3.1 Der Infrarotfilter muss kompatibel zum Objektiv sein.

HW2.3.3.2 Der Infrarotfilter muss das sichtbare Umgebungslicht absorbieren.

HW2.3.3.3 Der Infrarotfilter muss das von der Infrarotlichtquelle ausgestrahlte Licht (850 nm) passieren lassen.

HW2.4 Das Subsystem 2 muss ein Gestell für die Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten besitzen.

Folgende Anforderungen gehen aus Kapitelabschnitt 4.2.5 der Systembeschreibung hervor.

HW2.4.1 Das Gestell muss die stabile Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten ermöglichen.

HW2.4.2 Das Gestell muss eine Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten zentral über der Rennstrecke ermöglichen.

HW2.4.3 Das Gestell muss eine Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten in einer Höhe von 2,05 m über der Rennstrecke ermöglichen.

Um Störungen zu vermeiden soll neben der Rennstrecke selbst, ein möglichst kleiner Teil der Umgebung von der Kamera erfasst werden. Bei Verwendung des Weitwinkelobjektivs *Lensagon CMFA0420ND* [Obj] und der Größe der Minimum Bounding Box von 5,617 m^2 ergibt sich, dass die Kamera in einer Höhe von 2,05 m installiert werden sollte.

HW2.4.4 Die Beleuchtungseinheit und die Bilderfassungskomponenten müssen so dicht wie möglich nebeneinander am Gestell befestigt werden.

Dies ist wichtig da die verwendete Reflexionsfolie, das Licht direkt in die Richtung der Lichtquelle reflektiert (siehe Kapitelabschnitt 4.1.2). Bei der Befestigung der genannten Hardwarekomponenten ist zu berücksichtigen, dass diese während des Betriebs Hitze erzeugen. Daher muss gegebenenfalls ein Mindestabstand eingehalten werden, um die Komponenten vor Überhitzung zu schützen.

HW3 **Die Hardware des Subsystems 3 muss alle erforderlichen Komponenten für die autonome Fahrzeugsteuerung beinhalten.**

Diese Basisanforderung ergibt sich direkt aus der Systemanforderung Sys1.3. Die folgenden Unteranforderungen spezifizieren die geforderten Eigenschaften und Fähigkeiten der Hardwarekomponenten des Subsystems Control-Unit, welches in Kapitelabschnitt 4.3 der Systembeschreibung erläutert wurden.

HW3.1 Das Subsystem 3 muss einen Rechner zur Ausführung der Regelungssoftware besitzen.

Die Anforderungen an den verwendeten Rechner und die Regelungssoftware sind unter HW5 und SW2.1 spezifiziert.

HW3.2 Das Subsystem 3 muss einen DA-Wandler besitzen.

Es wird ein DA-Wandler benötigt, da die digitalen Ausgaben der Regelungssoftware in analoge Spannungsstärken umgewandelt werden müssen. Der im Projekt *RCCARS* verwendete DA-Wandler wird mit Hilfe eines Entwicklungsboards realisiert (siehe Kapitelabschnitt 8.4.4.3 der Systembeschreibung). Daraus lassen sich die folgenden

Anforderungen ableiten. Im Folgenden wird der übergeordnete Begriff CarControl verwendet. Dieser umfasst sowohl das Entwicklungsboard, als auch die daran angeschlossene Fernsteuerung.

HW3.2.1 Die CarControl muss eine USB-Schnittstelle zum Rechner besitzen.

Siehe auch Hardwareanforderung HW5 zum Rechner.

HW3.2.2 Die CarControl muss zwei DAC besitzen.

HW3.2.3 Die CarControl muss die digitalen Regelungssignale des Rechners in Spannungen im Bereich von 0 bis 3 V umwandeln.

Siehe auch Kapitelabschnitt 8.4.4.3 zur Fernbedienung.

HW3.2.4 Die CarControl muss die in Spannungen übersetzten Regelungssignale an die Fernsteuerung weiterleiten.

HW3.2.5 Die CarControl muss fähig sein, mindestens 100 Steuerungsbefehle pro Sekunde zu verarbeiten.

Um eine präzise autonome Steuerung des Fahrzeugs mit einer Mindestwiederholrate von 100 Hz zu ermöglichen, muss die Regelungssoftware (siehe Softwareanforderung SW2.1) 100 Steuerungsbefehle pro Sekunde ausgeben. Die CarControl ist in der Lage 100 Steuerungsbefehle in durchschnittlich 30 ms zu verarbeiten (siehe Kapitelabschnitt 4.3.4).

HW3.3 Das Subsystem 3 muss eine Fernsteuerung besitzen.

Die Fernsteuerung, welches ein Teil der CarControl ist, übermittelt die Steuerungsbefehle an das Fahrzeug. Aus Kapitelabschnitt 8.4.4.3 der Systembeschreibung gehen die folgenden Anforderungen hervor.

HW3.3.1 Die Fernsteuerung muss das Fahrzeug steuern können.

HW3.3.2 Die Fernsteuerung muss mit dem Fahrzeug kompatibel sein.

Es wird die Fernsteuerung *Perfex KT-18* [Fer] von *Kyosho* verwendet, welche für die Steuerung von *dNaNo*-RC-Cars (siehe Anforderung HW1.2) entwickelt wurde.

HW3.3.3 Die Fernsteuerung muss eine Komponente der CarControl sein.

HW3.3.4 Die Potentiometer in der Fernsteuerung müssen deaktiviert werden.

HW4 **Die Hardware des Subsystems 4 muss aus einer Rennstrecke bestehen.**

Diese Basisanforderung ergibt sich direkt aus der Systemanforderung Sys1.4. Die folgenden Unteranforderungen spezifizieren die Eigenschaften des Subsystems Rennstrecke, welches in Kapitelabschnitt 4.4 der Systembeschreibung beschrieben wurde. Weitere Anforderungen an die Rennstrecke der ersten Ausbaustufe wurden bereits in den Rahmenbedingungen Rah3 und Rah4 definiert.

HW4.1 Die Rennstrecke muss eine Fahrbahnbegrenzung auf beiden Seiten besitzen.

HW4.2 Der Verlauf der Rennstrecke muss eine geschlossene Runde bilden.

HW4.3 Die Rennstrecke muss für *dNaNo* RC-Cars geeignet sein.

Die Breite, die Kurvenradien sowie die Oberflächenstruktur der Rennstrecke, müssen für *dNaNo* RC-Cars (Anforderung HW1.2) geeignet sein.

HW4.4 An den äußeren Ecken der Rennstrecke müssen Reflexionsmarker platziert sein.

Diese Reflexionsmarker sind erforderlich, damit die Positionsbestimmungssoftware die Lage der Rennstrecke bestimmen kann.

HW5 **Zur Systemhardware muss ein Rechner gehören.**

Diese Basisanforderung ergänzt einige generelle Hardwareanforderungen, welche das gesamte System betreffen. Diese beziehen sich alle auf den Rechner, da dieser die einzige Hardwarekomponente ist, die von mehreren Subsystemen und von der Systemsteuerungssoftware (siehe Softwarebasisanforderungen SW4 und SW5) gemeinsam genutzt wird. Der im Projekt *RCCARS* verwendete Rechner wird in Kapitelabschnitt 4.2.6 beschrieben.

HW5.1 Der Rechner muss einen leistungsfähigen Prozessor besitzen.

Die Prozessorleistung des Rechners ist von besonderer Wichtigkeit. Der Rechner muss die Positionsbestimmungs-, die Regelungs- und die Systemsteuerungssoftware parallel ausführen (siehe Softwareanforderungen SW1, SW2.1 und SW3) und dabei so schnell sein, dass diese Softwarekomponenten ihre Realzeitanforderungen erfüllen können. Ein Rechner, welcher dazu zu langsam ist, würde im System einen Verlust an Präzision bei der autonomen Fahrzeugsteuerung bedeuten und wäre somit ein Sicherheitsrisiko. Daher wurde ein

Prozessor gewählt, welcher zu Projektbeginn (November 2015), dem gehobenen Stand der Technik entsprach.

HW5.1.1 Der Rechner muss fähig sein, die Positionsbestimmungs-, die Regelungs- und die Systemsteuerungssoftware parallel auszuführen.

HW5.1.2 Der Rechner muss so schnell sein, dass die Positionsbestimmungs-, die Regelungs- und die Systemsteuerungssoftware ihre Realzeitanforderungen erfüllen können.

Dies gilt unter der Annahme, dass diese Softwarekomponenten fehlerfrei und laufzeiteffizient implementiert sind.

HW5.2 Der Rechner muss eine USB 3.0 Schnittstelle besitzen.

Dies ist notwendig, damit die in Kapitelabschnitt 4.2.4 beschriebene Kamera mindesten 100 Bilder pro Sekunde an den Rechner übertragen kann.

HW5.3 Der Rechner muss eine USB-Schnittstelle zur Kommunikation mit der Car-Control besitzen.

Siehe Kapitelabschnitt 8.4.4.3 der Systembeschreibung.

HW5.4 Der Rechner muss einem Administrator Kontrollelemente für die Bedienung des Systems bereitstellen.

Als Kontrollelemente gelten Tastatur, Maus und Monitor. Zusätzlich wird ein Not-Aus-Button in das System integriert.

5.2.3 Softwareanforderungen

In diesem Kapitelabschnitt werden die Anforderungen spezifiziert, welche an die Software des Systems gestellt werden. Sie leiten sich aus den entsprechenden softwarebezogenen Abschnitten der Systembeschreibung (Kapitel 4) ab. Die Basisanforderungen SW1 bis SW3 repräsentieren die Softwarekomponenten des Systems.

SW1 **Die Software des Subsystems 2 muss fähig sein, die Pose der Fahrzeuge zu bestimmen.**

Diese Basisanforderung ergibt sich direkt aus der Systemanforderung Sys1.2. Die folgenden Unteranforderungen spezifizieren die geforderten Eigenschaften und Fähigkeiten der Positionsbestimmungssoftware, welche in Kapitelabschnitt 4.2.7 der Systembeschreibung beschrieben wurden. Auch die in Kapitelabschnitt 4.2.8 erläuterten sicherheitsrelevanten Aspekte des Subsystems 2 werden berücksichtigt.

SW1.1 Die Positionsbestimmungssoftware muss fähig sein, Bilder von der Kamera zu empfangen.

SW1.2 Die Positionsbestimmungssoftware muss fähig sein, aus den Bilddaten der Kamera die Pose des Fahrzeugs zu erkennen.

SW1.2.1 Die Positionsbestimmungssoftware muss fähig sein, die Bilder der Kamera nachzubearbeiten.

Dazu gehört beispielsweise das Entzerren und das Entrauschen des Bildes.

SW1.2.2 Die Positionsbestimmungssoftware muss durch Erkennung der Reflexionsmarker die Fahrzeuge erfassen.

Angaben zu den Reflexionsmarkern sind in Kapitelabschnitt 4.1 der Systembeschreibung zu finden.

SW1.2.3 Die Positionsbestimmungssoftware muss die zweidimensionalen Fahrzeugkoordinaten feststellen.

SW1.2.4 Die Positionsbestimmungssoftware muss die Ausrichtung der Fahrzeuge feststellen.

SW1.2.5 Die Positionsbestimmungssoftware muss die Identität der Fahrzeuge anhand der Anordnung der Reflexionsmarker feststellen.

SW1.3 Die Positionsbestimmungssoftware muss die Daten der Fahrzeugposen an die Regelungssoftware weiterleiten.

SW1.4 Die Positionsbestimmungssoftware muss fähig sein, die Bilderfassung, die Erkennung der Fahrzeugpose und die Weiterleitung der ermittelten Daten an die Regelungssoftware in höchstens 10 ms durchzuführen.

SW1.5 Positionsbestimmungssoftware muss fähig sein, die Lage der Rennstrecke zu erfassen.

Sowohl im Rahmen der Initialisierung als auch während des Rennbetriebs muss die Positionsbestimmung auch die Lage der Rennstrecke erfassen, da die Fahrzeugposen, welche an die Control-Unit übermittelt werden, auf den relativen Koordinaten der Rennstrecke basiert (siehe Kapitelabschnitt 4.4.4).

SW1.6 Die Positionsbestimmungssoftware muss Störungen erkennen.

SW1.6.1 Die Positionsbestimmungssoftware muss erkennen, ob die Rennstrecke verschoben wurde.

Dazu gehört auch das Bewegen der Kamera und das Verschieben des Gestells, da dies für das System die selben Auswirkungen in Bezug auf die Erfassung der Fahrzeuge und der Rennstrecke hat, wie das Verschieben der Rennstrecke selbst. Leichte Erschütterungen aus der Umgebung der Rennstrecke oder Ungenauigkeiten bei der Bilderfassung müssen hier jedoch toleriert werden. Als leichte Erschütterungen ist beispielsweise das Auftreten von Personen, welche an der Rennstrecke vorbeigehen, zu verstehen. Diese Toleranzen sind festzulegen, sobald die Entwicklung der Positionsbestimmung soweit vorangeschritten ist, dass die Lage der Rennstrecke erfasst werden kann.

SW1.6.2 Falls die Positionsbestimmungssoftware eine Störung erkennt, muss sie dies der Systemsteuerungssoftware mitteilen.

Die Systemsteuerungssoftware (siehe Softwareanforderung SW3) muss in diesem Fall dafür sorgen, dass die Fahrzeuge angehalten werden und das System in den Fehlerzustand (siehe Kapitelabschnitt 4.6) versetzt wird.

SW2 **Die Software des Subsystems 3 muss fähig sein, ein Fahrzeug autonom über eine vorgegebene Rennstrecke zu steuern.**

Diese Basisanforderung ergibt sich direkt aus der Systemanforderung Sys1.3. Die folgenden Unteranforderungen definieren die geforderten Eigenschaften und Fähigkeiten der Regelungssoftware, welche in Kapitelabschnitt 4.3.3 der Systembeschreibung beschrieben wurden. Die in Kapitelabschnitt 4.3.4 erläuterten sicherheitsrelevanten Aspekte des Subsystems 3 werden ebenfalls berücksichtigt. Informationen zur Streckenführung sind Kapitelabschnitt 4.4.2 zu entnehmen.

SW2.1 Das Subsystem 3 muss eine Regelungssoftware für die autonome Steuerung des Fahrzeugs besitzen.

SW2.1.1 Die Regelungssoftware muss die Daten der Positionsbestimmung empfangen.

SW2.1.2 Die Regelungssoftware muss aus den Daten der Positionsbestimmung die Regelungsparameter für die autonome Steuerung des Fahrzeugs berechnen.

SW2.1.3 Die Regelungssoftware muss fähig sein, das Fahrzeug entlang einer vorgegeben Route zu steuern.

Wenn die Regelungssoftware gestartet wird, bekommt sie die vom Fahrzeug zu fahrende Route als Eingabeparameter übergeben. Im Rahmen der ersten Ausbaustufe des Projektes *RCCARS* soll das Fahrzeug in der Mitte der Fahrbahn fahren (siehe Kapitelabschnitt 4.4.2).

SW2.1.4 Die Regelungssoftware muss den Verlauf der Rennstrecke und die vom Fahrzeug zu fahrende Route als Eingabeparameter empfangen.

SW2.1.5 Die Regelungssoftware muss das Fahrzeug gegen den Uhrzeigersinn um die Rennstrecke steuern.

SW2.1.6 Die Regelungssoftware muss fähig sein, das Fahrzeug mit einer Durchschnittsgeschwindigkeit von mindestens 1,5 m/s über eine vorgegebene Rennstrecke zu steuern.

Die Anforderung leitet sich aus Systemanforderung Sys3 ab.

SW2.1.7 Das Fahrzeug darf die Kurvengeschwindigkeit von 1,2 m/s nicht überschreiten.

Damit das Fahrzeug sicher autonom gesteuert werden kann, muss es in der Kurve die Spur halten. Empirische Tests im Rahmen der Hardwareevaluation haben ergeben, dass die *dNaNo*-Fahrzeuge des verwendeten Typs *FX-101* fähig sind, bei einer Geschwindigkeit von 1,2 m/s auf der verwendeten Rennstrecke *Mini-Z Grand Prix Circuit 30* in der Kurve zuverlässig die Spur zu halten. Dies gilt sowohl für 90° als auch für 180° Kurven.

SW2.1.8 Die Regelungssoftware muss fähig sein, das Fahrzeug mindestens fünf Runden lang über die vorgegebene Rennstrecke zu steuern.

SW2.1.9 Die Regelungssoftware muss die Regelungsdaten für die Fahrzeugsteuerung an die CarControl übermitteln.

SW2.1.10 Die Regelungssoftware muss Unfälle und Störungen erkennen.

SW2.1.10.1 Die Regelungssoftware muss erkennen, ob das Fahrzeug den befahrbaren Bereich der Rennstrecke verlassen hat.

Der befahrbare Bereich der Rennstrecke wird Kapitelabschnitt 4.4.4 der Systembeschreibung definiert.

SW2.1.10.2 Die Regelungssoftware muss fähig sein, eine Kollision des Fahrzeugs mit der Streckenbegrenzung zu erkennen.

SW2.1.10.3 Die Regelungssoftware muss erkennen, ob das Fahrzeug in die richtige Richtung fährt.

Es ist vorgesehen, dass das Fahrzeug gegen den Uhrzeigersinn gesteuert wird. Sollte dies nicht der Fall sein, muss die Regelungssoftware dies feststellen (siehe Kapitelabschnitt 4.4.2 der Systembeschreibung).

SW2.1.10.4 Die Regelungssoftware muss den Verlust des Sichtkontakts zum Fahrzeug erkennen.

Dies tritt ein, wenn sich das Fahrzeug überschlägt, die Bilderfassung gestört wird oder das Fahrzeug das Sichtfeld der Kamera verlässt. In diesen Fall gibt die Positionsbestimmungssoftware statt der Fahrzeugpose Fehlerdaten aus, so dass die Regelungssoftware erkennen kann, dass das Fahrzeug nicht erfasst werden konnte.

SW2.1.10.5 Falls keine Daten von der Positionsbestimmung empfangen werden, muss die Regelungssoftware das Fahrzeug stoppen.

Sobald das Fahrzeug für 20 ms nicht erkannt wird, muss die Regelungssoftware das Fahrzeug stoppen. Diese 20 ms entsprechen der Aufnahmezeit von zwei Kamerabildern. Ausgehend von der Durchschnittsgeschwindigkeit von 1,5 m/s (siehe Kapitelabschnitt 2.4) kann das Fahrzeug in dieser Zeit 3 cm zurücklegen. Bei dieser Geschwindigkeit muss die Regelungssoftware, fähig sein das Fahrzeug anzuhalten bevor es die Fahrbahnbegrenzung berührt. Da diese Geschwindigkeit – beispielsweise auf den langen Geraden der Rennstrecke – auch überschritten werden kann, ist es möglich, dass sich das Fahrzeug der Fahrbahnbegrenzung so schnell nähert, dass eine Kollision mit dieser unter Umständen nicht mehr vermieden werden kann. Im diesem Fall muss die Regelungssoftware das Fahrzeug sofort abbremsen, so dass die Geschwindigkeit des Fahrzeugs bei der Kollision mit der Fahrbahnbegrenzung möglichst gering ist.

SW2.1.10.6 Die Regelungssoftware muss fähig sein, den Verlust der Verbindung zur CarControl zu erkennen.

Die Regelungssoftware sendet Befehle über die CarControl an das Fahrzeug (siehe Kapitelabschnitt 8.4.4.3). Ist diese Verbindung unterbrochen, so ist eine Kontrolle des Fahrzeugs nicht mehr möglich. Die Kommunikation zwischen der Regelungssoftware und der CarControl wird über eine virtuelle serielle Schnittstelle hergestellt. Bei einem Verbindungsverlust wird diese automatisch geschlossen. Das Senden eines Befehls an die CarControl ist demnach nicht mehr möglich und ein Fehler wird erzeugt.

SW2.1.10.7 Die Regelungssoftware muss bei der Auswertung der Fahrzeugpose eine Plausibilitätsüberprüfung durchführen.

Siehe Kapitelabschnitt 4.3.3.

SW2.1.10.8 Falls die Regelungssoftware eine Störung erkennt, muss sie das Fahrzeug stoppen.

SW2.1.10.9 Falls die Regelungssoftware eine Störung erkennt, muss sie dies der Systemsteuerungssoftware mitteilen.

Die Systemsteuerungssoftware (siehe Softwareanforderung SW3) muss in diesem Fall nur noch dafür sorgen, dass das System in den Fehlerzustand (siehe Kapitelabschnitt 4.6) versetzt wird, da bereits die Regelungssoftware sicherstellt, dass das Fahrzeug angehalten wird (siehe Softwareanforderung SW2.1.10.8).

SW2.1.11 Bei der Entwicklung der Regelungssoftware muss das Werkzeug *SCADE* [SCA] zum Einsatz kommen.

SW2.2 Das Subsystem 3 muss eine Software für die Umwandlung der digitalen Regelungsdaten in analoge Spannungen besitzen.

SW2.2.1 Die CarControlsoftware muss die Regelungsdaten empfangen können.

SW2.2.2 Die CarControlsoftware muss die digitalen Regelungsdaten in Spannungen im Bereich zwischen 0 und 3 V umwandeln.

SW2.2.3 Die CarControlsoftware muss die in Spannungen übersetzen Regelungsdaten an die Fernsteuerung übermitteln.

SW2.2.4 Falls keine Daten von der Regelungssoftware empfangen werden, muss die CarControlsoftware das Fahrzeug stoppen.

Siehe auch Softwareanforderung SW2.1.10.6.

SW2.3 Die Software der Control-Unit (Regelungssoftware und CarControlsoftware) muss fähig sein, die Berechnung und das Weiterleiten der Steuersignale in weniger als 10 ms durchzuführen.

Die Berechnung und das Weiterleiten darf nicht länger als 10 ms dauern, da nach Ablauf dieser Zeitspanne bereits neue Daten von der Positionsbestimmung vorliegen. Siehe hierzu auch Kapitelabschnitt *Communication between computer and transmitter* in [EGH+15].

SW3 **Die Systemsteuerungssoftware muss Funktionalitäten zur Bedienung und Überwachung des Systems bieten.**

Diese Basisanforderung ergibt sich aus der Systemanforderung Sys2. Die folgenden Unteranforderungen definieren die geforderten Eigenschaften und Fähigkeiten der Systemsteuerungssoftware, welche in Kapitelabschnitt 4.7 der Systembeschreibung beschrieben wurden. Auch der Kapitelabschnitt 4.6 zum Kontrollfluss des Systembetriebs wird an dieser Stelle berücksichtigt.

SW3.1 Die Systemsteuerungssoftware muss dem Administrator die Möglichkeit bieten, das System über ein Benutzerinterface zu bedienen.

SW3.1.1 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, zu bestätigen, dass die Inspektion erfolgreich durchgeführt wurde.

Die Anforderungen an die Inspektion werden Abschnitt 5.1 erläutert.

SW3.1.2 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, die Initialisierung des Systems starten.

Diese Anforderung leitet sich aus Systemanforderung Sys2.2 ab.

SW3.1.3 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, nach der Initialisierung den Rennbetrieb zu starten.

Diese Anforderung leitet sich aus Systemanforderung Sys2.4 ab.

SW3.1.4 Das Benutzerinterface muss dem Administrator jederzeit die Möglichkeit bieten, den Rennbetrieb zu stoppen.

Diese Anforderung leitet sich aus Systemanforderung Sys2.5 ab.

SW3.1.5 Das Benutzerinterface muss dem Administrator jederzeit die Möglichkeit bieten, das System auszuschalten.

Diese Anforderung leitet sich aus Systemanforderung Sys2.3 ab.

SW3.1.6 Das Benutzerinterface muss dem Administrator jederzeit die Möglichkeit bieten, das System manuell in den Fehlerzustand zu versetzen.

Diese Anforderung leitet sich aus Systemanforderung Sys2.6 ab.

SW3.1.7 Das Benutzerinterface muss den Administrator über den aktuellen Systemzustand informieren.

SW3.2 Falls ein sicherheitskritischer Fehler auftritt, muss die Systemsteuerungssoftware das System in einen sicheren Fehlerzustand überführen.

SW3.2.1 Die Systemsteuerungssoftware muss die Fehlermeldungen, der Positionsbestimmungssoftware und der Regelungssoftware empfangen.

Siehe Softwareanforderungen SW1.6.2 und SW2.1.10.9.

SW3.2.2 Falls ein sicherheitskritischer Fehler auftritt, muss die Systemsteuerungssoftware allen Instanzen der Regelungssoftware mitteilen, dass die Fahrzeuge sofort gestoppt werden müssen.

SW4 Die Systemsteuerungssoftware, die Positionsbestimmungssoftware und alle Instanzen der Regelungssoftware müssen parallel ausgeführt werden können.

Die Positionsbestimmungssoftware und die Regelungssoftware müssen im Rahmen der ersten Ausbaustufe des Projektes *RCCARS* auf dem selben Rechner ausgeführt werden. Um die Leistung des Systems zu erhöhen, müssen diese parallel ausgeführt werden. So kann bereits ein neues Kamerabild verarbeitet werden, während noch Regelungsdaten berechnet werden, welche auf dem vorherigen Kamerabild basieren.

SW5 Die Systemsteuerungssoftware, die Positionsbestimmungssoftware und die Regelungssoftware müssen mit der höchsten Priorität ausgeführt werden.

Damit die Systemsteuerungssoftware, Positionsbestimmungssoftware und die Regelungssoftware im Vergleich zu anderen Prozessen die höchste Priorität erhalten, muss ein Realzeitbetriebssystem als Ausführungsumgebung verwendet werden (siehe Kapitelabschnitt 4.2.8). Die Systemsteuerungssoftware muss dafür Sorgen, dass die genannten Softwarekomponenten die entsprechende Priorität erhalten. Daraus leitet sich die folgende Anforderung an das Betriebssystem ab.

SW5.1 Das auf dem Rechner installierte Betriebssystem muss realzeitfähig sein.

Falls im Rahmen einer späterer Ausbaustufe eine oder mehrere Control-Units auf einen oder mehreren anderen Rechnern betrieben werden als die Positionsbestimmung und die Systemsteuerungssoftware, gilt diese Anforderung auch für jeden dieser Rechner.

5.3 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen definieren Eigenschaften des Systems, welche nicht essentiell für die Funktionalität sind, jedoch die Qualität verbessern oder die Entwicklungskosten reduzieren. Diese Anforderungen werden wiederum nach ”generellen Systemanforderungen”, ”Softwareanforderungen” und ”Hardwareanforderungen” unterteilt.

5.3.1 Generelle Systemanforderungen

In der aktuellen Ausbaustufe des Projektes *RCCARS* liegen keine nicht-funktionalen Systemanforderungen vor. Auch die Systemanforderung Sys5 und deren Unteranforderungen zur Erweiterbarkeit des Systems wurden den funktionalen Anforderungen zugeordnet, da diese essentiell für die vorgesehene Weiterentwicklung des Systems sind (siehe auch 2.2).

5.3.2 Hardwareanforderungen

NHW1 Die Systementwickler müssen die CarControl selbst konstruieren.

Bei den Referenzprojekten der ETH Zürich [ORC14] und der Universität Uppsala [CAR14] (siehe Kapitelabschnitt 2.1) wurden DA-Wandlerkarten verwendet, welche sehr kostspielig sind. Das Projekt *RCCARS* orientiert sich daher am Referenzprojekt der KU Leuven, welche einen DA-Wandler selbst konstruiert hat.

5.3.3 Softwareanforderungen

NSW1 **Die Softwarekomponenten des Systems müssen fähig sein, Information über den Systembetrieb in Textform zu protokollieren.**

Damit die Vorgänge während des Systembetriebs nachvollzogen werden können und um eine Erkennung und Auswertung von Fehlern zu ermöglichen, müssen die von den einzelnen Softwarekomponenten erzeugten

Ausgaben gespeichert werden. Zu Optimierungszwecken ist es außerdem erforderlich zu erfahren, wie lange die Softwarekomponenten benötigen, um ihre Aufgaben auszuführen. Da diese Protokollfunktionen die Programmlaufzeit verlängern, sollen sie optional im Rahmen eines Debugging-Modus aktiviert werden können. Die Ausgabe von Fehlerberichten ist hingegen immer erforderlich. Daraus ergeben sich die folgenden Anforderungen an die jeweiligen Softwarekomponenten.

NSW1.1 Die Positionsbestimmungssoftware muss im Debugging-Modus die an die Regelungssoftware übergebenen Parameter protokollieren.

NSW1.2 Die Positionsbestimmungssoftware muss im Debugging-Modus die Laufzeit der Bildverarbeitung protokollieren.

NSW1.3 Die Regelungssoftware muss im Debugging-Modus die berechneten Regelungssignale protokollieren.

NSW1.4 Die Regelungssoftware muss im Debugging-Modus die Laufzeit der Regelungssignalberechnung protokollieren.

NSW1.5 Die Software auf der CarControl muss im Debugging-Modus die an die Fernsteuerung übergebenen Spannungsstärken protokollieren.

NSW1.6 Die Software auf der CarControl muss im Debugging-Modus die Laufzeit der Signalumwandlung protokollieren.

NSW1.7 Falls ein Fehler auftritt müssen die betroffenen Softwarekomponenten des Systems einen Fehlerbericht erstellen.

6 Anwendungsfälle

In diesem Kapitel werden grundlegende Anwendungsfälle definiert, welche sich direkt aus dem Szenario ”Unfallfreie Fahrt” (siehe Kapitelabschnitt 2.4) ableiten. Diese Anwendungsfälle werden durch UML-Use-Case-Diagramme und Anwendungsfalltabellen spezifiziert. Daraus werden im Kapitelabschnitt 12.7 die grundlegenden Testfälle abgeleitet, welche verwendet wurden, um die erfolgreiche Umsetzung des Szenarios zu verifizieren.

Das Kapitel ist in zwei Abschnitte untergliedert, in denen die Anwendungsfälle des Szenarios aus der Sicht zweier unterschiedlicher Typen von Akteuren definiert werden. Zunächst werden jene Anwendungsfälle spezifiziert, welche sich auf Personen beziehen, die mit dem System interagieren können. Anschließend werden weitere Anwendungsfälle definiert, welche sich aus der Sicht der vier Subsysteme (siehe Kapitel 4) ergeben. Dementsprechend wird das Use-Case-Diagramm des Szenarios in zwei Hälften unterteilt. Dadurch soll auch die Lesbarkeit und die Übersichtlichkeit verbessert werden.
Die Anwendungsfalltabellen sind alle nach dem folgenden Schema aufgebaut: Die eindeutige **ID** erleichtert die Referenzierung und entspricht dem Namen des Anwendungsfalls. Die **Beschreibung** verschafft dem Leser eine Übersicht über den jeweiligen Anwendungsfall. **Akteure** sind die am Anwendungsfall beteiligten Personen bzw. Subsysteme und der **Auslöser** beschreibt die Aktionen, welche nötig sind, um die Ausführung des Anwendungsfalls auszulösen. Die **Vorbedingung** gibt die Voraussetzungen an, unter denen der Anwendungsfall eintreten kann, während die **Nachbedingung** beschreibt, welche Aktion nach dem Anwendungsfall eintreten wird. Der Tabellenabschnitt **Ablauf** verdeutlicht die Abfolge von Ereignissen, die während der Durchführung des Use-Cases auftreten. Das Feld **Ergebnis** beschreibt die Situation, welche nach einer erfolgreichen Durchführung des Anwendungsfalls vorliegt. Durch diese Strukturierung bauen die zusammenhängenden Anwendungsfälle des Szenarios logisch aufeinander auf.
In der letzten Zeile werden die **Anforderungen** aus den Rahmenbedingungen (Kapitelabschnitt 5.1) und den generellen Systemanforderungen (Kapitelabschnitt 5.2.1) referenziert, welche für die Realisierung des jeweiligen Anwendungsfalls relevant sind. Falls die angegebenen Anforderungen bzw. Rahmenbedingungen in Unteranforderungen zerfallen, sind auch diese zu berücksichtigen. Die in der Anforderungsspezifikation aufgeführten

spezifischen Soft- und Hardwareanforderungen leiten sich aus den generellen Systemanforderungen ab und werden somit indirekt ebenfalls referenziert.

6.1 Systeminteraktion mit Personen

In diesem Kapitelabschnitt werden die Anwendungsfälle des Szenarios spezifiziert, in denen das System mit Personen interagiert (siehe Abbildung 6.1). Die einzige Person, die aktiv mit dem System interagiert, ist der **Administrator**. Allein der Administrator ist autorisiert, das System zu starten, zu bedienen und zu warten. Alle weiteren Personen werden als **Zuschauer** bezeichnet, welche das System lediglich aus sicherer Entfernung betrachten und nicht in den Systembetrieb eingreifen dürfen.

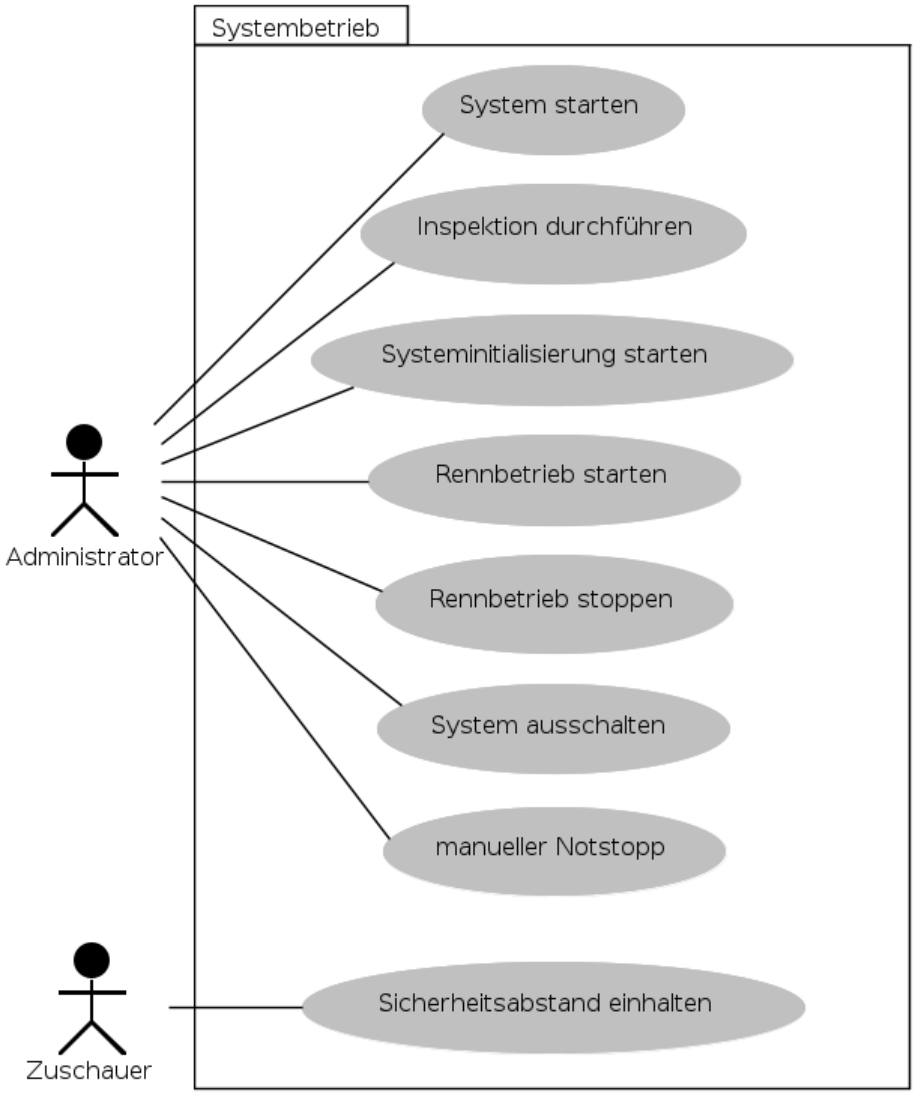


Abbildung 6.1: Erster Teil des Anwendungsfalldiagramms für das Szenario ”Unfallfreie Fahrt” mit Administrator und Zuschauern.

Die folgenden Anwendungsfalltabellen beschreiben die einzelnen Use-Cases des Szenarios mit den Akteuren Administrator und Zuschauer im Detail.

ID	A01
Beschreibung	Der Administrator startet das System.
Akteure	Administrator
Auslöser	Das System soll gestartet werden.
Vorbedingung	Alle Hardwarekomponenten des System sind voll funktionsfähig und korrekt verkabelt. Die benötigte Software ist installiert.
Nachbedingung	Es muss eine Inspektion durchgeführt werden.
Ablauf	1. Der Administrator schaltet die Hardware ein. 2. Der Administrator startet die Systemsteuerungssoftware.
Ergebnis	Die Hardwarekomponenten und die Systemsteuerungssoftware sind gestartet und das System ist bereit für die Inspektion.
Anforderungen	Sys2.7

Tabelle 6.1: Anwendungsfall System starten

ID	A02
Beschreibung	Der Administrator führt eine Inspektion der Hardwarekomponenten des Systems durch. Dazu gehören auch die Rennstrecke und das Fahrzeug sowie das Einhalten des Sicherheitsabstands zu den Hardwarekomponenten durch die Zuschauer. Außerdem startet der Administrator die Softwarekomponenten der Positionsbestimmung und der Control-Unit.
Akteure	Administrator
Auslöser	Die Inspektion soll durchgeführt werden.
Vorbedingung	Die Hardwarekomponenten des Systems und die Systemsteuerungssoftware wurden gestartet.
Nachbedingung	Der Administrator muss die erfolgreiche Inspektion bestätigen.
Ablauf	1. Der Administrator prüft, ob die Zuschauer den geforderten Sicherheitsabstand zur Rennstrecke einhalten. 2. Der Administrator prüft, ob die Rennstrecke intakt ist und befreit diese ggf. von Staub und störenden Objekten. 3. Der Administrator prüft den Zustand des Fahrzeugs und säubert dieses ggf. oder ersetzt Fahrzeugteile. 4. Der Administrator positioniert das Fahrzeug auf der Start-Ziel-Linie in der Mitte der Fahrbahn.

	5. Der Administrator prüft, ob die Minimum Bounding Box der Rennstrecke frei von direkter Sonneneinstrahlung ist. 6. Der Administrator startet die Softwarekomponenten der Positionsbestimmung und der Control-Unit und überprüft, ob diese funktionsfähig sind. 7. Der Administrator prüft, ob eine Verbindung zwischen Fahrzeug und Control-Unit besteht. 8. Der Administrator bestätigt ggf. die erfolgreiche Inspektion über das Benutzerinterface der Systemsteuerungssoftware.
Ergebnis	Die Hard- und Softwarekomponenten sind bereit für die Initialisierung.
Anforderungen	Rah1 bis Rah4, Sys2.1

Tabelle 6.2: Anwendungsfall Inspektion durchführen

ID	A03
Beschreibung	Der Administrator startet die Systeminitialisierung, bei welcher auch die Positionsbestimmungssoftware und die Regelungssoftware initialisiert werden. Dazu gehört ggf. auch die Konfiguration dieser Softwarekomponenten.
Akteure	Administrator
Auslöser	Der Administrator leitet mit Hilfe der Systemsteuerungssoftware die Systeminitialisierung ein.
Vorbedingung	Die Inspektion wurde vollständig durchgeführt und es liegen keine Fehler oder Störungen vor.
Nachbedingung	Das System befindet sich im Standby-Modus.
Ablauf	1. Der Administrator startet die Systeminitialisierung über das Benutzerinterface der Systemsteuerungssoftware. 2. Die Positionsbestimmungssoftware und die Regelungssoftware werden initialisiert und ggf. vom Administrator konfiguriert. 3. Die Positionsbestimmungssoftware und die Regelungssoftware teilen der Systemsteuerungssoftware ggf. mit, dass sie für den Rennbetrieb bereit sind.
Ergebnis	Das System und die dazugehörigen Hard- und Softwarekomponenten sind initialisiert.
Anforderungen	Sys2.2

Tabelle 6.3: Anwendungsfall Systeminitialisierung starten

ID	A04
Beschreibung	Der Administrator gibt den Rennbetrieb frei.
Akteure	Administrator
Auslöser	Der Rennbetrieb soll gestartet werden.
Vorbedingung	Das System ist initialisiert.
Nachbedingung	Das Fahrzeug setzt sich in Bewegung.
Ablauf	Der Administrator startet den Rennbetrieb mit Hilfe der Systemsteuerungssoftware.
Ergebnis	Das System befindet sich im Rennbetrieb.
Anforderungen	Sys2.4

Tabelle 6.4: Anwendungsfall Rennbetrieb starten

ID	A05
Beschreibung	Der Administrator stoppt den Rennbetrieb und das Fahrzeug kommt zum Stillstand.
Akteure	Administrator
Auslöser	Der Rennbetrieb soll beendet werden.
Vorbedingung	Das System befindet sich im Rennbetrieb.
Nachbedingung	Das System befindet sich im Standby-Modus.
Ablauf	1. Der Administrator stoppt den Rennbetrieb mit Hilfe der Systemsteuerungssoftware. 2. Das Fahrzeug erhält ein Stop-Signal und bleibt stehen. 3. Das System wechselt in den Standby-Modus.
Ergebnis	Der Rennbetrieb wurde beendet.
Anforderungen	Sys2.3, Sys2.5

Tabelle 6.5: Anwendungsfall Rennbetrieb stoppen

ID	A06
Beschreibung	Der Administrator beendet den Systembetrieb.
Akteure	Administrator
Auslöser	Das System soll ausgeschaltet werden.
Vorbedingung	Das System wurde eingeschaltet.

Nachbedingung	Alle Soft- und Hardwarekomponenten des Systems sind ausgeschaltet.
Ablauf	1. Der Administrator beendet den Systembetrieb über die Systemsteuerungssoftware. 2. Falls sich das System im Rennbetrieb befindet, wird das Fahrzeug sicher zum Stillstand gebracht. 3. Die Softwarekomponenten der Subsysteme werden geschlossen. 4. Die Systemsteuerungssoftware wird geschlossen. 5. Der Administrator schaltet alle Hardwarekomponenten des Systems aus.
Ergebnis	Das System ist ausgeschaltet.
Anforderungen	Sys2.3

Tabelle 6.6: Anwendungsfall System ausschalten

ID	A07
Beschreibung	Der Administrator muss den Systembetrieb aufgrund einer sicherheitskritischen Störung anhalten.
Akteure	Administrator
Auslöser	Eine sicherheitskritische Störung ist aufgetreten. Ein solcher Fall tritt beispielsweise ein, wenn ein Zuschauer die Rennstrecke betritt.
Vorbedingung	Die Inspektion wurde erfolgreich durchgeführt und das System befindet sich in der Initialisierung, im Standby-Modus oder im Rennbetrieb.
Nachbedingung	Das Fahrzeug steht still.
Ablauf	1. Der Administrator sendet ein Notstopp-Signal über die Systemsteuerungssoftware. 2. Das System wechselt in den Fehlerzustand. 3. Falls sich das System im Rennbetrieb befindet, wird das Fahrzeug sofort angehalten. 4. Der Administrator kann das Fehlerprotokoll einsehen.
Ergebnis	Das System befindet sich im Fehlerzustand.
Anforderungen	Sys2.6

Tabelle 6.7: Anwendungsfall Manueller Notstopp

ID	A08
Beschreibung	Um sowohl die Systemkomponenten, als auch die Zuschauer zu schützen, muss jeder Zuschauer einen Sicherheitsabstand zum System einhalten.
Akteure	Zuschauer
Auslöser	Zuschauer beobachten den Rennbetrieb.
Vorbedingung	Die Inspektion wird durch den Administrator durchgeführt.
Nachbedingung	Nach der Inspektion müssen alle Zuschauer den Sicherheitsabstand einhalten.
Ablauf	1. Der Administrator führt die Inspektion durch. 2. Der Administrator bittet die Zuschauer den Sicherheitsabstand zu halten.
Ergebnis	Die Zuschauer halten den geforderten Sicherheitsabstand ein.
Anforderungen	Rah2, Rah4

Tabelle 6.8: Anwendungsfall Sicherheitsabstand einhalten

6.2 Anwendungsfälle der Subsysteme

In diesem Kapitelabschnitt werden die Anwendungsfälle des Szenarios aus Sicht der Subsysteme spezifiziert. Der Akteur **Positionsbestimmung** umfasst die Hard- und Softwarekomponenten, welche zur Erfassung der Fahrzeugpose bis hin zur Weiterleitung dieser notwendig sind. Der Akteur **Control-Unit** erhält die von der Positionsbestimmung ermittelte Fahrzeugpose und nutzt diese, um das Fahrzeugs autonom über die Rennstrecke zu steuern. Die Akteure **Fahrzeug** und **Rennstrecke** nehmen eine untergeordnete Rolle ein.

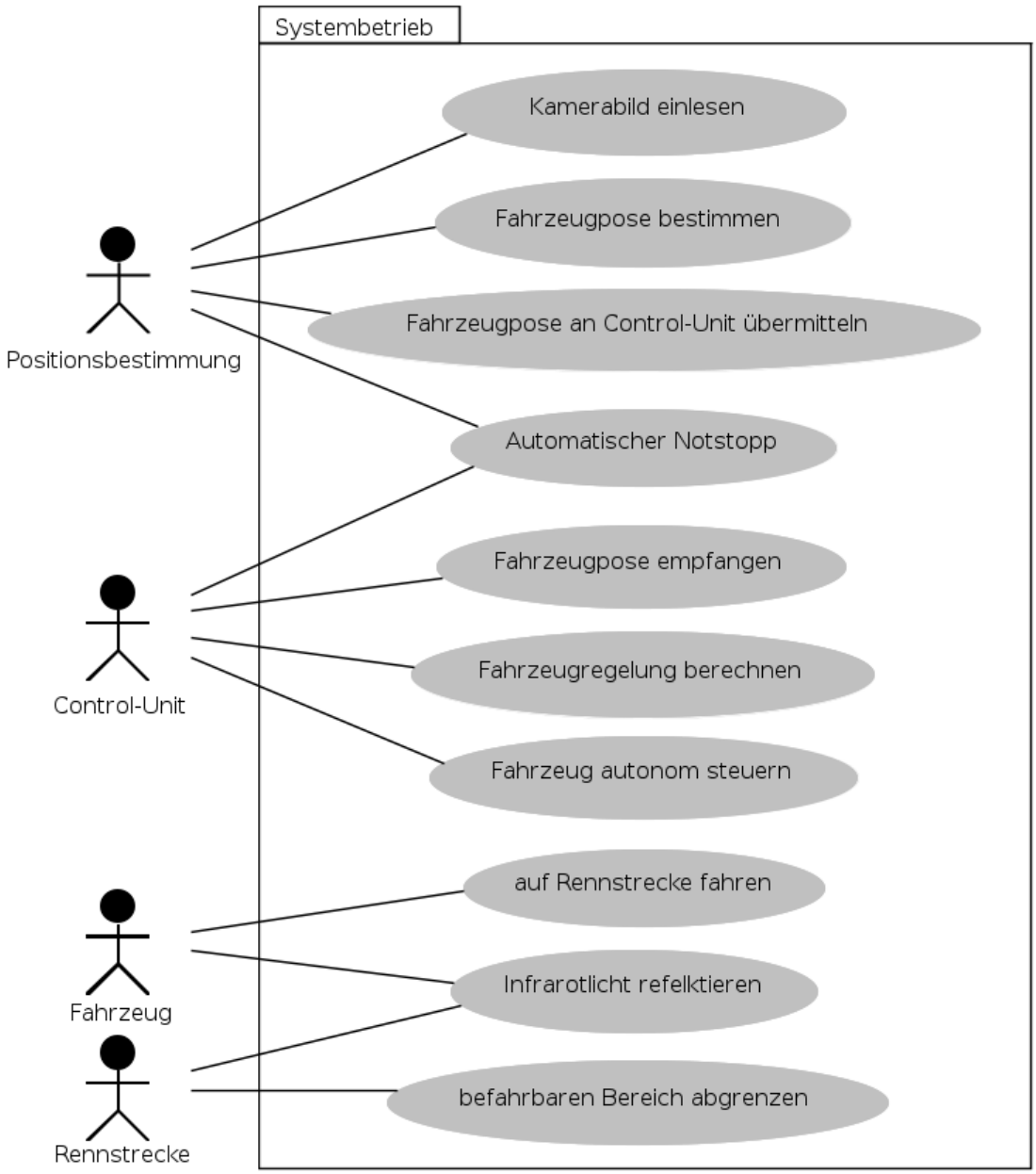


Abbildung 6.2: Zweiter Teil des Anwendungsfalldiagramms für das Szenario ”Unfallfreie Fahrt” mit den vier Subsystemen.

Die folgenden Anwendungsfalltabellen beschreiben die einzelnen Use-Cases des Szenarios mit den Akteuren Positionsbestimmung, Control-Unit, Fahrzeug und Rennstrecke im Detail.

ID	B01
Beschreibung	Die Positionsbestimmung des Systems empfängt ein Kamerabild der Rennstrecke und des Fahrzeugs.
Akteure	Positionsbestimmung
Auslöser	Die Positionsbestimmungssoftware ist oder wird initialisiert und empfängt ein Bild der Kamera.
Vorbedingung	Das System befindet sich in der Initialisierung oder hat diese erfolgreich abgeschlossen.
Nachbedingung	Das Kamerabild wird verarbeitet.
Ablauf	1. Die Kamera nimmt ein Bild auf. 2. Die Positionsbestimmungssoftware liest das Kamerabild ein.
Ergebnis	Die Positionsbestimmung hat ein Kamerabild empfangen.
Anforderungen	Sys1.2

Tabelle 6.9: Anwendungsfall Kamerabild einlesen

ID	B02
Beschreibung	Die Positionsbestimmung ermittelt aus dem eingelesenen Kamerabild die Pose des Fahrzeugs.
Akteure	Positionsbestimmung
Auslöser	Die Positionsbestimmung hat ein neues Kamerabild eingelesen.
Vorbedingung	Das System befindet sich in der Initialisierung oder hat diese erfolgreich abgeschlossen.
Nachbedingung	Die ermittelte Fahrzeugpose muss an die Control-Unit weiter geleitet werden.
Ablauf	Die Positionsbestimmungssoftware verarbeitet das Kamerabild, um die Fahrzeugpose zu ermitteln.
Ergebnis	Die Fahrzeugpose wurde bestimmt.
Anforderungen	Sys1.2

Tabelle 6.10: Anwendungsfall Fahrzeugpose bestimmen

ID	B03
Beschreibung	Die Positionsbestimmung leitet die ermittelte Fahrzeugpose an die Control-Unit weiter.
Akteure	Positionsbestimmung
Auslöser	Die Positionsbestimmung hat eine neue Fahrzeugpose bestimmt.
Vorbedingung	Das System befindet sich in der Initialisierung oder hat diese erfolgreich abgeschlossen.
Nachbedingung	Die Positionsbestimmung liest ein neues Kamerabild ein.
Ablauf	Die Positionsbestimmung leitet die Pose des Fahrzeugs an die Control-Unit weiter.
Ergebnis	Die Positionsbestimmung hat die Pose des Fahrzeugs an die Control-Unit übermittelt.
Anforderungen	Sys1.2

Tabelle 6.11: Anwendungsfall Fahrzeugpose übermitteln

ID	B04
Beschreibung	Die Positionsbestimmung oder die Control-Unit muss aufgrund eines sicherheitskritischen Fehlerfalls den Rennbetrieb automatisch stoppen.
Akteure	Positionsbestimmung oder Controll-Unit
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Solch ein Fall tritt beispielsweise ein, wenn eine Verbindungen zwischen den Hardwarekomponenten gestört ist.
Vorbedingung	Das System befindet sich in der Initialisierung, im Standby-Zustand oder im Rennbetrieb.
Nachbedingung	Das Fahrzeug steht still.
Ablauf	1. Die Positionsbestimmung oder die Control-Unit versendet ein Notstopp-Signal. 2. Das System wechselt in den Fehlerzustand 3. Das Fahrzeug erhält ein Stoppsignal und bleibt sofort stehen, falls es in Bewegung ist. 4. Der Administrator kann das Fehlerprotokoll einsehen.
Ergebnis	Das System befindet sich im Fehlerzustand.
Anforderungen	Sys4.5

Tabelle 6.12: Anwendungsfall Automatischer Notstopp

ID	B05
Beschreibung	Die Control-Unit empfängt eine Fahrzeugpose von der Positionsbestimmung.
Akteure	Control-Unit
Auslöser	Die Positionsbestimmung übermittelt eine Fahrzeugpose.
Vorbedingung	Das System wurde erfolgreich initialisiert.
Nachbedingung	Die Control-Unit berechnet aus der Fahrzeugpose die Regelungsparameter für die autonome Fahrzeugsteuerung.
Ablauf	Die Control-Unit empfängt eine Fahrzeugpose von der Positionsbestimmung.
Ergebnis	Die Control-Unit hat eine Fahrzeugpose empfangen.
Anforderungen	Sys1.3

Tabelle 6.13: Anwendungsfall Fahrzeugpose empfangen

ID	B06
Beschreibung	Die Control-Unit berechnet die Regelungsparameter für die autonome Fahrzeugsteuerung.
Akteure	Control-Unit
Auslöser	Die Control-Unit hat eine Fahrzeugpose von der Positionsbestimmung empfangen.
Vorbedingung	Das System befindet sich im Rennbetrieb.
Nachbedingung	Die Control-Unit übermittelt die Regelungsparameter an das Fahrzeug.
Ablauf	Die Control-Unit berechnet die Regelungsparameter für die autonome Fahrzeugsteuerung.
Ergebnis	Die Regelungsparameter wurden berechnet.
Anforderungen	Sys1.3

Tabelle 6.14: Anwendungsfall Fahrzeugregelung berechnen

ID	B07
Beschreibung	Die Control-Unit steuert das Fahrzeug autonom über den befahrbaren Bereich der Rennstrecke.

Akteure	Control-Unit
Auslöser	Die Regelungsparameter wurden berechnet.
Vorbedingung	Das System befindet sich im Rennbetrieb.
Nachbedingung	Die Control-Unit empfängt die nächste Fahrzeugpose von der Positionsbestimmung oder der Rennbetrieb wurde beendet.
Ablauf	Die Control-Unit übermittelt die berechneten Regelungsparameter an das Fahrzeug.
Ergebnis	Falls es nicht zu einem Fehler oder einer Störung kommt, steuert die Control-Unit das Fahrzeug so, dass es fünf Runden lang autonom mit einer Mindestdurchschnittsgeschwindigkeit von 1,5 m/s unfallfrei über den befahrbaren Bereich der Rennstrecke fährt.
Anforderungen	Sys1.3, Sys3, Sys4.1, Sys4.2, Sys4.3, Sys4.4, Sys4.6

Tabelle 6.15: Anwendungsfall Fahrzeug autonom steuern

ID	B08
Beschreibung	Das Fahrzeug fährt auf der Rennstrecke.
Akteure	Fahrzeug
Auslöser	Das Fahrzeug hat Steuerungsbefehle von der Control-Unit erhalten.
Vorbedingung	Das System befindet sich im Rennbetrieb.
Nachbedingung	Das Fahrzeug empfängt weitere Steuerungsdaten von der Control-Unit, falls sich das System weiterhin im Rennbetrieb befindet.
Ablauf	1. Das Fahrzeug empfängt Steuerungsdaten von der Control-Unit. 2. Das Fahrzeug setzt die Steuerungsdaten um und bewegt sich über die Rennstrecke.
Ergebnis	Das Fahrzeug befährt den befahrbaren Bereich der Rennstrecke.
Anforderungen	Sys1.1

Tabelle 6.16: Anwendungsfall Auf Rennstrecke fahren

ID	B09
Beschreibung	Das Fahrzeug befindet sich auf der Rennstrecke und dessen Reflexionsmarker reflektieren das Licht der Infrarotlampe.
Akteure	Fahrzeug
Auslöser	Die Reflexionsmarker des Fahrzeugs werden von der Infrarotlampe angestrahlt.
Vorbedingung	Das Fahrzeug befindet sich auf der Rennstrecke und die Infrarotlampe ist eingeschaltet.
Nachbedingung	Keine
Ablauf	Die Reflexionsmarker auf dem Fahrzeug reflektieren das Licht der Infrarotlampe.
Ergebnis	Das von den Reflexionsmarkern des Fahrzeugs reflektierte Infrarotlicht wird von der Kamera erfasst.
Anforderungen	Sys1.1

Tabelle 6.17: Anwendungsfall Infrarotlicht reflektieren (Fahrzeug)

ID	B10
Beschreibung	Die Fahrbahnbegrenzung der Rennstrecke grenzt den befahrbaren Bereich von der Umgebung ab.
Akteure	Rennstrecke
Auslöser	Die Rennstrecke wurde vollständig montiert.
Vorbedingung	Die Rennstrecke ist vollständig und jeder Streckenabschnitt besitzt eine Fahrbahnbegrenzung auf beiden Seiten.
Nachbedingung	Keine
Ablauf	Die Fahrbahnbegrenzung der Rennstrecke grenzt den befahrbaren Bereich von der Umgebung ab.
Ergebnis	Das Fahrzeug kann den befahrbaren Bereich der Rennstrecke während des Rennbetriebs nicht verlassen.
Anforderungen	Sys1.4

Tabelle 6.18: Anwendungsfall Befahrbaren Bereich abgrenzen

ID	B11
Beschreibung	Die an den äußeren Ecken der Rennstrecke platzierten Reflexionsmarker reflektieren das Licht der Infrarotlampe.
Akteure	Rennstrecke
Auslöser	Die Reflexionsmarker der Rennstrecke werden von der Infrarotlampe angestrahlt.
Vorbedingung	An den äußeren Ecken der Rennstrecke wurden Reflexionsmarker platziert und die Infrarotlampe ist eingeschaltet.
Nachbedingung	Keine
Ablauf	Die Reflexionsmarker der Rennstrecke reflektieren das Licht der Infrarotlampe.
Ergebnis	Das von den Reflexionsmarkern der Rennstrecke reflektierte Infrarotlicht wird von der Kamera erfasst.
Anforderungen	Sys1.4

Tabelle 6.19: Anwendungsfall Infrarotlicht reflektieren (Rennstrecke)

6.3 Abdeckung der Anforderungen

In diesem Kapitelabschnitt wird gezeigt, dass die zuvor definierten Anwendungsfälle die in Kapitel 5 spezifizierten Anforderungen an das Projekt *RCCARS* abdecken.

In Tabelle 6.20 ist zu erkennen, dass alle Rahmenbedingungen vollständig abgedeckt sind. Die direkte Referenzierung der übergeordneten Basisrahmenbedingungen impliziert, dass auch deren Unteranforderungen zu erfüllen sind.

Anwendungsfall \ **Rahmenbedingung**	**Rah1**	**Rah2**	**Rah3**	**Rah4**
A02	X	X	X	X
A08		X		X

Tabelle 6.20: Diese Matrix zeigt die Abdeckung der Rahmenbedingungen durch die entsprechenden Anwendungsfälle.

Tabelle 6.21 zeigt, dass alle generellen Systemanforderungen, bis auf Sys5, durch die Anwendungsfälle abgedeckt werden. Die Anforderung Sys5 und deren Unteranforderungen wurden durch keinen der Anwendungsfälle erfasst, da sie sich auf die Erweiterbarkeit und somit direkt auf die Entwicklung des Systems beziehen, während sich die erstellten

Anwendungsfälle am Szenario ”Unfallfreie Fahrt” orientieren. Gleiches gilt für alle nichtfunktionalen Anforderungen.
Durch den hierarchischen Aufbau der Anforderungsspezifikation lassen sich alle spezifischen Soft- und Hardwareanforderungen auf die generellen Systemanforderungen zurückführen. Daher wird auch deren Realisierung implizit gefordert.

Basierend auf den beschriebenen Anwendungsfällen werden in Kapitel 12 die grundlegenden Testfälle und Testszenarien definiert. Im folgenden Kapitel wird jedoch zunächst die Hardwareentwicklung und -Evaluation behandelt.

Anwendungsfall \ Systemanf.	1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	2.5	2.6	2.7	3	4.1	4.2	4.3	4.4	4.5	4.6
A01											X							
A02					X													
A03						X												
A04								X										
A05							X		X									
A06							X											
A07										X								
B01		X																
B02		X																
B03		X																
B04																	X	
B05			X															
B06			X															
B07			X									X	X	X	X	X		X
B08	X																	
B09	X																	
B10				X														
B11				X														

Tabelle 6.21: Diese Matrix zeigt die Abdeckung der Systemanforderungen durch die entsprechenden Anwendungsfälle. Der einzige in dieser Tabelle nicht aufgeführte Anwendungsfall ist A08, welcher bereits vollständig durch die Rahmenbedingungen abgedeckt ist (siehe Tabelle 6.20).

7 Entwicklung und Evaluation der Hardware

Dieses Kapitel befasst sich mit der Evaluation der im Projekt RCCARS verwendeten Hardwarekomponenten. Es wird gezeigt, dass die in Kapitel 4 beschriebene Hardware der vier Subsysteme die entsprechenden Hardwareanforderungen erfüllt bzw. welche Anpassungen ggf. vorgenommen wurden, um die Performanz oder die Zuverlässigkeit des Systems zu verbessern. Dadurch wird sichergestellt, dass die hardwareseitigen Voraussetzungen für die erfolgreiche Realisierung des in Kapitelabschnitt 2.4 beschriebenen Szenarios "Unfallfreie Fahrt" erfüllt sind und auch die Erweiterung des System im Rahmen der geplanten Folgeszenarios (siehe Kapitelabschnitt 2.2) möglich ist.

7.1 Subsystem 1: Fahrzeug

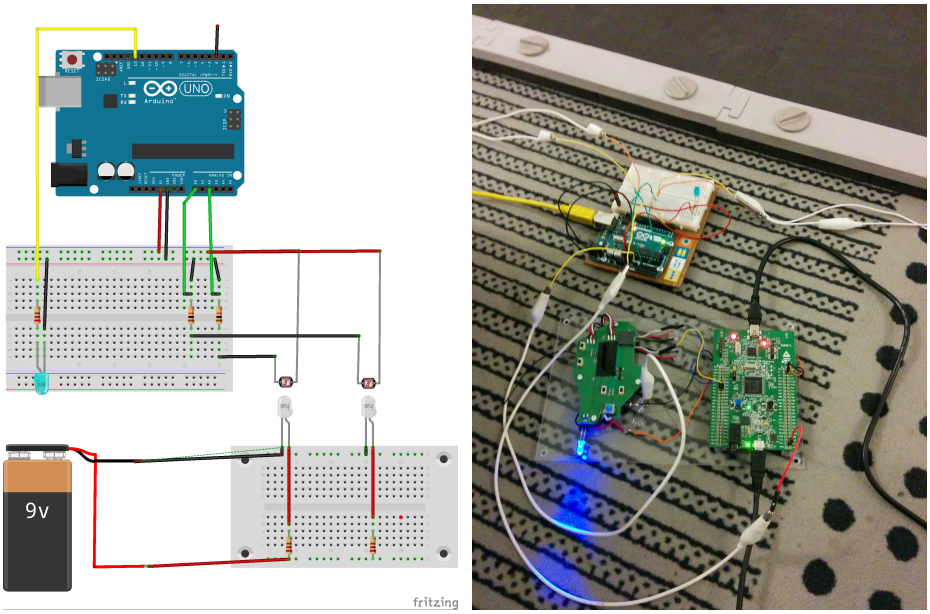
In diesem Kapitelabschnitt wird das Fahrzeug im Hinblick auf die gestellten Hardwareanforderungen evaluiert. Darüber hinaus werden die Resultate der Fahrdynamikuntersuchungen präsentiert.

7.1.1 Fahrzeug

Das verwendete Modellfahrzeug ist von der Marke *dNaNo* des Modelltyps *FX-101* [KYO]. Dies bestätigt die Aufschrift auf der Produktverpackung und die Beschriftungen auf der Unterseite des Fahrzeugs. Damit sind HW1.1 und HW1.2 erfüllt. Der Akku und die Reifen sollten wegen der hohen Abnutzung regelmäßig gewechselt werden. Zudem muss die Aufhängung so staubfrei wie möglich gehalten werden, da schon einzelne Staubkörner das Fahrverhalten negativ beeinflussen können. Untersuchungen mithilfe mehrerer Testszenarien, welche in Kapitelabschnitt 7.1.1.1 bis Kapitelabschnitt 7.1.1.3 beschrieben werden, haben ergeben, dass das Fahrzeug korrekt funktioniert, da es sich entsprechend der Regelungswerte verhält. Damit ist auch HW1.3 erfüllt. Zudem konnte durch Langzeittests die im Rahmen des Schülerinformationstages durchgeführt wurden (mehr als vierstündiger Betrieb nur unterbrochen durch Akkutausch) nachgewiesen werden, dass

ein voll geladener Akku das Fahrzeug für mindestens fünf Runden ohne einen feststellbaren Leistungsverlust mit Energie versorgen kann, womit die HW1.3.1 ebenfalls erfüllt ist.

Um im Rahmen der Testszenarien die Fahreigenschaften des Fahrzeugs zu ermitteln wurde ein Testaufbau errichtet. Der Testaufbau, siehe Abb. 7.2 besteht aus einer 3 m langen Geraden, welche durch eine Linkskurve abgeschlossen wird. Im Abstand von einem Meter sind zwei Lichtschranken errichtet, welche von einem Arduino Uno überwacht werden. Sowohl der Aufbau der Lichtschranke, welche jeweils aus einer weißen LED und einem Photoresistor besteht, als auch Teile des Quellcodes zur Zeitmessung mittels einer Lichtschranke sind bereits im Rahmen des Schülerinformationstages entstanden und wurden daher wiederverwendet. In Abbildung 7.1(a) ist der schematische Aufbau der Lichtschranke abgebildet. Über die GPIO Pins A0 und A3 des Arduino Uno werden Spannungsänderungen detektiert welche beim durchfahren der Lichtschranken verursacht werden. Löst das Fahrzeug die ersten Lichtschranke aus, so beginnt eine Zeitmessung und die an GPIO Pin 13 angeschlossene LED leuchtet auf. Durchquert das Fahrzeug nun die zweite Lichtschranke wird die Zeitmessung gestoppt und die LED deaktiviert. Des Weiteren wird die Geschwindigkeit des Fahrzeugs über die serielle Schnittstelle des Arduino Uno, welche mit einem PC verbunden ist, ausgegeben.



(a) Aufbau der Lichtschranke

(b) CarControl-Prototyp (unten) in Kombination mit der Lichtschranke (oben)

Abbildung 7.1: Aufbau der Evaluationsschaltung

In Abbildung 7.1(b) ist ein erweiterter Testaufbau mit dem Arduino Uno und ersten pro-

totypischen Version der CarControl dargestellt. Über den GPIO Pin 3 des Arduino Uno ist es möglich ein Kontrollsignal beim Durchfahren der Lichtschranken an die CarControl zu übermitteln. Sobald die CarControl das Kontrollsignal empfängt, wird das Fahrzeug entsprechend der jeweiligen Versuchsreihe voll gebremst oder eingelenkt. Hierdurch kann nicht nur festgestellt werden wie die Fahrzeuggeschwindigkeit mit dem digitalen Beschleunigungssignal zusammenhängt, sondern auch wie lang der Bremsweg des Fahrzeugs ist und mit welcher Geschwindigkeit eine Kurve noch sicher durchfahren werden kann.

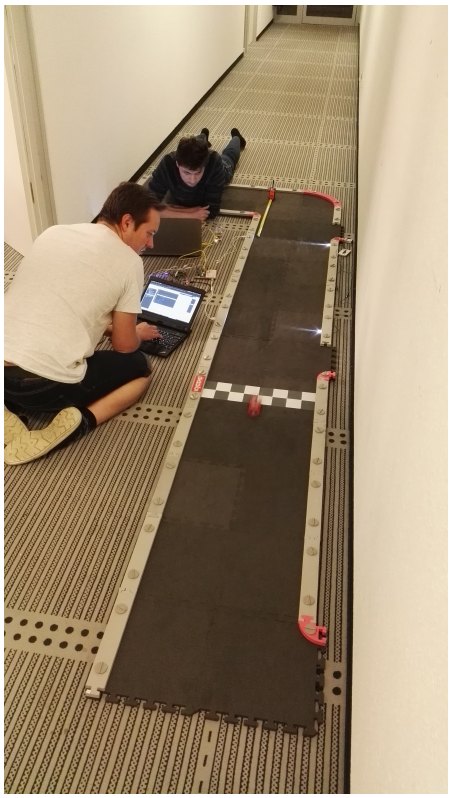


Abbildung 7.2: Studierende der PG RCCARS evaluieren die Fahrzeuggeschwindigkeit mithilfe des Testaufbaus.

7.1.1.1 Evaluation der Geschwindigkeit

Um die Abhängigkeit zwischen den Ansteuerungsbefehlen der CarControl und der resultierenden Geschwindigkeit des Fahrzeugs feststellen zu können, wurde aus der Zeit die das Fahrzeug benötigt um die Lichtschranken zu durchqueren und der Strecke zwischen den beiden Lichtschranken die jeweilige Geschwindigkeit ermittelt. Vor der ersten Lichtschranke sind zusätzlich zwei Meter Wegstrecke für die Fahrzeugbeschleunigung vorgesehen. Die Resultate dieser Versuchsreihe sind in Tabelle 7.1 dargestellt. In der ersten Spalte gibt ”Throttle” den Wert des Regelungswertes der CarControl für die Beschleu-

nigung an und die zweite Spalte beschreibt die resultierende Geschwindigkeit in Metern pro Sekunde. Des Weiteren ermöglicht diese Untersuchung eine Orientierung bei der Ermittlung der Regelungsparameter zur Fahrzeugbeschleunigung.

Evaluation der Fahrzeuggeschwindigkeit	
Throttle	**v in m/s**
1300	3,470
1400	3,186
1500	3,139
1600	2,653
1700	2,185
1725	2,008
1750	1,926
1800	1,684
1825	1,539
1835	1,501
1840	1,480
1850	1,072
1875	0,999
1890	0,963
1900	0,097

Tabelle 7.1: Ergebnisse Fahrzeugevaluation: Geschwindigkeit

Um in dieser Versuchsreihe noch zusätzlich die Rennstreckenbegrenzung zu evaluieren, wurde das Fahrzeug nach der zweiten Lichtschranke ungebremst in die Fahrbahnbegrenzung gesteuert. Diese empirische Erhebung hat ergeben, dass das Fahrzeug die Rennstrecke selbst bei hohen Geschwindigkeiten nicht verlassen kann.

Dadurch, dass der Beschleunigungsbefehl und die daraus resultierende Geschwindigkeit jetzt bekannt sind, kann in einem weiteren Versuch festgestellt werden mit welcher Geschwindigkeit das Fahrzeug noch kontrollierbar um die Kurven der Rennstrecke lenkbar ist.

7.1.1.2 Evaluation des Bremswegs

Um den Bremsweg zu bestimmen wurde der Versuchsaufbau erneut leicht modifiziert und die Kurve um einen Meter weiter zurück gesetzt. Sobald das Fahrzeug die zweite Lichtschranke durchquert sendet die CarControl dem Fahrzeug ein dauerhaftes Rückwärtsfahrsignal. Da das Fahrzeug technisch bedingt nicht unmittelbar rückwärts fahren kann, beginnt das Fahrzeug zu bremsen und bleibt schließlich stehen. Sobald das Fahrzeug nach der Lichtschranke zum Stillstand gekommen ist, wird der Bremsweg vermessen, siehe

Abbildung 7.3. In Tabelle 7.2 sind die evaluierten Bremswege zu den jeweiligen gefahrenen Geschwindigkeiten in Metern pro Sekunde dargestellt.

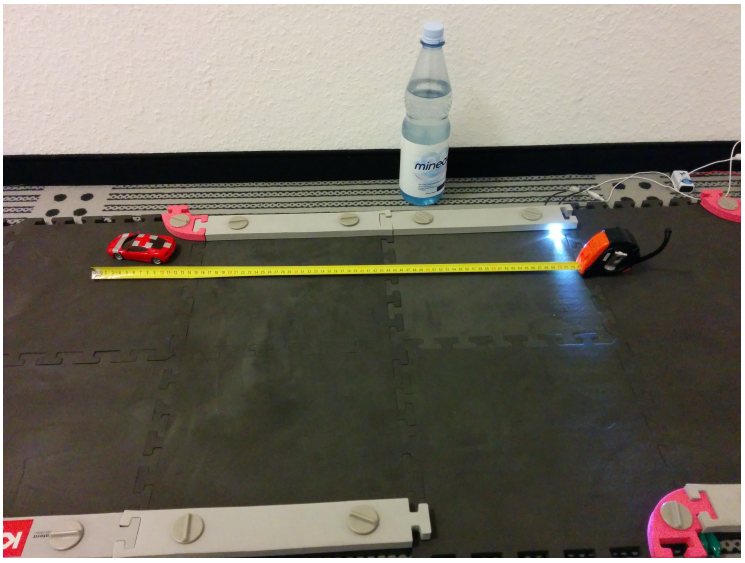


Abbildung 7.3: Das Fahrzeug ist nach der zweiten Lichtschranke zum Halten gekommen.

Evaluation des Bremswegs	
v in m/s	**Bremsweg in cm**
3,470	>110
3,186	>110
3,139	>110
2,653	>100
2,231	110
2,088	104
1,968	96,5
1,911	100,5
1,612	73,5
1,539	74
1,470	72
0,999	42
0,963	40
0,097	< 40

Tabelle 7.2: Ergebnisse der Fahrzeugevaluation: Bremsweg

7.1.1.3 Evaluation des Lenkwinkels

Da in den Datenblättern des Fahrzeuges keine Angaben zum maximalen Lenkwinkel getätigt wurden, musste dieser selbst evaluiert werden. Im Referenzprojekt der ETH Zürich

Lenkwinkel	Spannung	Regelungswert
-30° (max. links)	0,15 V	280
0° (neutral)	1,5 V	2130
30° (max. rechts)	2,85 V	3980

Tabelle 7.3: Ergebnisse der Fahrzeugevaluation: Lenkwinkel

[OE09, S. 69-70] wurde der Lenkwinkel bereits evaluiert. Dieser beträgt für den vollen Linkseinschlag -30° bei ca. 0,15 V, für den vollen Rechtseinschlag 30° bei ca. 2,85 V und für die Neutralstellung 1,5 V. Diese Informationen entsprechen auch unseren Untersuchungen. Die Spannung wurde mit einem Oszilloskop geprüft und der Lenkwinkel mit der Bestimmung des Winkels zwischen Fotos des Fahrzeuges mit maximalem Einschlag jeweils einer Richtung und der Neutralstellung.

Um in unserem Aufbau die korrekten Regelungswerte für die CarControl zu finden, wurden die Werte vom Regelungswert der Neutralstellung in beide Richtungen soweit iteriert, bis der maximale Lenkeinschlag gefunden war bzw. sich der Lenkeinschlag nicht mehr verändert hat. Der Regelungswert von 2130 für die Neutralstellung wurde bereits bei der Entwicklung der CarControl ermittelt. Das Ergebnis dieses Tests kann der Tabelle 7.3 entnommen werden. Die Regelungswerte zwischen den Lenkwinkeln von -30° und 0° und 0° und 30° werden, wie im Referenzprojekt der ETH Zürich [OE09, S. 69-70] durch Interpolation festgestellt.

7.1.2 Reflexionsfolie

Die Reflexionsfolie *Scotchlite Reflective Material 8850 Silver* lässt sich problemlos an der Karosserie des Fahrzeugs befestigen. Die angebrachten Marker haften stark am Plastik und können nur mit Kraftaufwand entfernt werden. Daher ist es nicht möglich, dass sich die Reflexionsfolie versehentlich im Rennbetrieb lösen kann. Die Reflexionseigenschaften des Materials sind äußerst zufriedenstellend. Untersuchungen, die im Rahmen der Entwicklung der Positionsbestimmung entstanden sind haben gezeigt, dass unabhängig von der Position des Fahrzeugs auf der Rennstrecke, die 1 cm^2 großen Marker im Kamerabild problemlos für Mensch und Positionsbestimmungssoftware erkennbar sind, sofern sich die Infrarotlichtquelle in unmittelbarer Nähe der Kamera befindet. Damit sind die Hardwareanforderungen HW1.4.1, HW1.4.2, HW1.4.3, HW1.4.4, HW1.4.5 erfüllt.

Wenn die Lichtquelle nicht neben der Kamera positioniert ist, sind auf den Kamerabildern keine Infrarotmarker erkennbar. Dies ist auf die Eigenschaft der Reflexionsfolie zurückzuführen, auftretendes Licht zurück zur Lichtquelle zu reflektieren. Damit ist auch HW1.4.6

erfüllt. Somit sind alle Unteranforderungen von HW1.4, woraus folgt, dass HW1.4 selbst ebenfalls auch erfüllt ist.

7.2 Subsystem 2: Positionsbestimmung

Im Folgenden werden die Hardwarekomponenten des Subsystems Positionsbestimmung evaluiert. Abbildung 7.4 zeigt ein Foto auf dem der Aufbau der Hardwarekomponenten der Subsysteme Positionsbestimmung und Rennstrecke zu sehen ist.

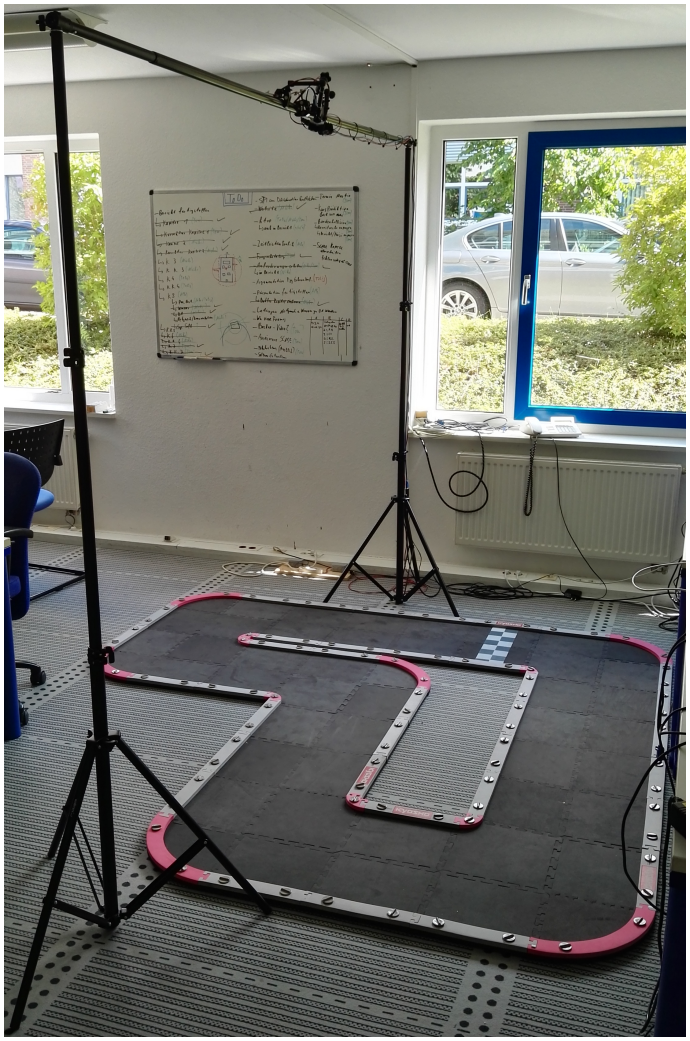


Abbildung 7.4: Aufbau der Hardwarekomponenten der Subsysteme Positionsbestimmung und Rennstrecke.

7.2.1 Kamera

Die Verwendete Kamera *Point Grey Flea FL3-U3-13Y3M-C* [Kam] liefert monochrome Bilder (gemäß Hardwareanforderung HW2.3.1.4) mit der geforderten Auflösung von

1280 · 1024 Pixeln und einer Framerate von annähernd 150 Hz. Dies zeigt der Screenshot der FlyCapture-Software [Fly] in Abbildung 7.5.

Die Hardwareanforderung HW2.3.1.1, welche eine Framerate von mindestens 100 Hz fordert, wird also durch diese Kamera erfüllt. Es ist jedoch zu beachten, dass beim Anschließen der Kamera zuerst die USB-Verbindung zwischen Rechner und Kamera hergestellt werden muss und die Stromversorgung der Kamera erst danach angeschlossen werden darf, da die Framerate sonst lediglich 40 Hz beträgt.

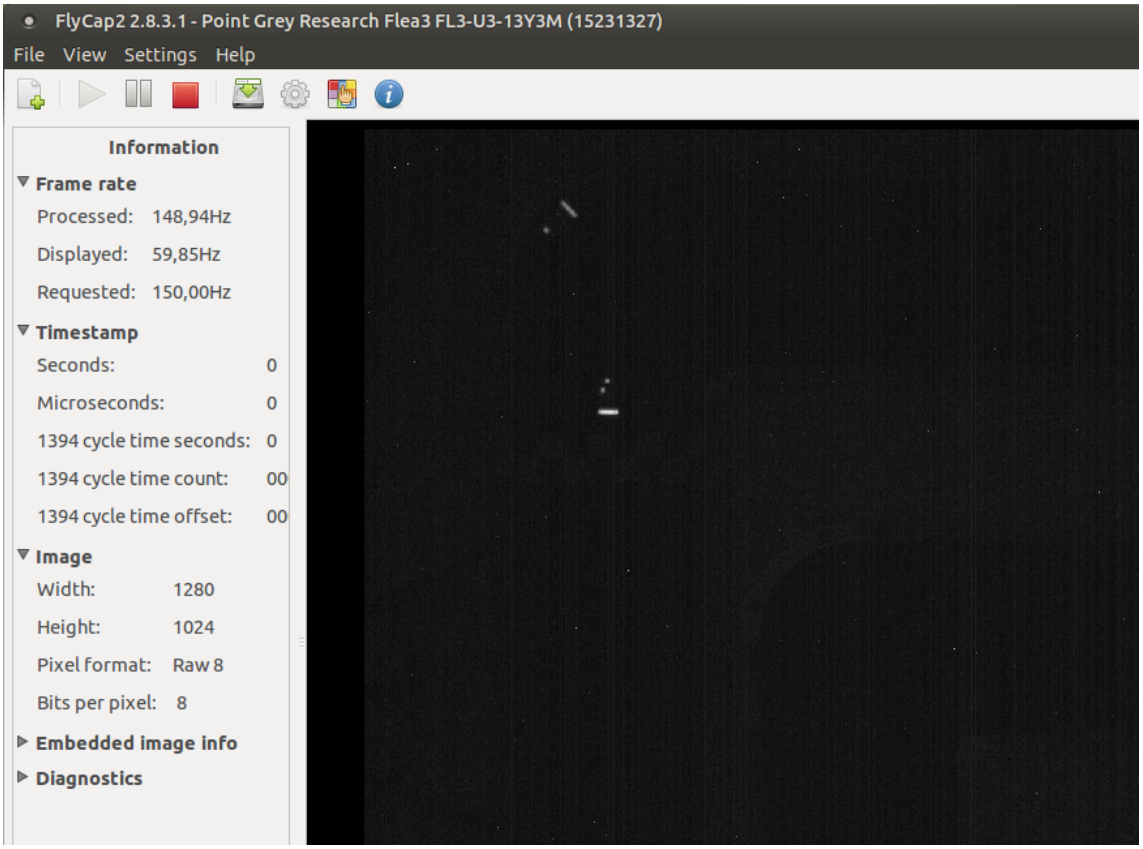


Abbildung 7.5: Ausschnitt aus einem Screenshot der FlyCapture-Software. Am linken Bildrand sind die Auflösung und die Framerate der Kamera zum Zeitpunkt der Aufnahme zu sehen.

In Abbildung 7.6 ist des Weiteren zu erkennen, dass die Kamera fähig ist Infrarotlicht zu erfassen, da die Reflexionsmarker unter Infrarotlichtbestrahlung hell aufleuchten. Die Hardwareanforderung HW2.3.1.5 ist also erfüllt.

Abbildung 7.4 zeigt, dass die Kamera zentral über der Rennstrecke montiert wurde. In Abbildung 7.7 aus Abschnitt 7.2.4 ist außerdem zu erkennen, dass aus dieser Position die gesamte Minimum Bounding Box der Rennstrecke von der Kamera mit einer ausreichend

hohen Auflösung aufgenommen wird, so dass die einzelnen Reflexionsmarker der Fahrzeuge zuverlässig zu erkennen sind (siehe Abbildung 7.7 unten rechts). Somit sind auch die Hardwareanforderungen HW2.3.1.2 und HW2.3.1.3 erfüllt. Die Kamera wird somit allen Hardwareanforderungen gerecht, so dass HW2.3.1 insgesamt erfüllt ist.

7.2.2 Objektiv

Durch das verwendete Objektiv *Lensagon CMFA0420ND* [Obj] ist die Kamera fähig, die gesamte Rennstrecke aus der berechneten Mindesthöhe von 2,05 m zu erfassen. Das o. g. Objektiv lässt sich direkt auf die in Abschnitt 7.2.1 beschriebene Kamera aufschrauben und erfüllt somit die Hardwareanforderungen HW2.3.2 und HW2.3.2.1.

7.2.3 Infrarotlampe

Die Infrarot-LEDs des Typs *OSLON 4 IR Wide PowerStar* [IR] emittieren Licht mit einer Wellenlänge von 850 nm. In Verbindung mit dem entsprechenden Kühlkörper und Netzteil wurden diese mit einer Gelenkhalterung in direkter Nähe der Kamera befestigt (siehe Abbildung 7.4). Abbildung 7.6 zeigt wie das Infrarotlicht von den Reflexionsfolie (siehe Kapitelabschnitt 7.1.2) zurück zur Kamera reflektiert wird, so dass die Reflexionsmarker der Fahrzeuge hell aufleuchten. Die Hardwareanforderungen HW2.2.1 bis HW2.2.4 sind somit erfüllt.

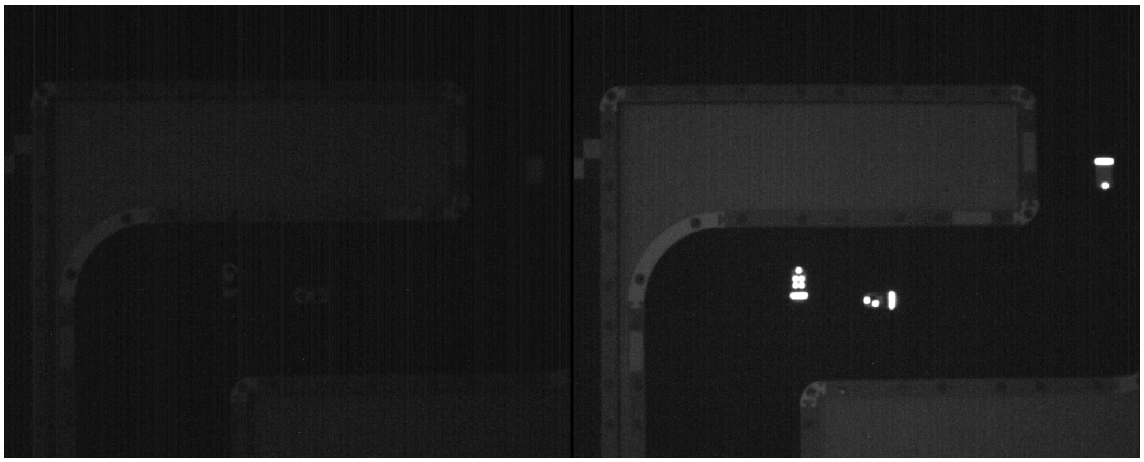


Abbildung 7.6: Kameraaufnahmen der Fahrzeuge (herangezoomt). Links ohne und rechts mit Infrarotbeleuchtung.

Der von der Infrarotlampe bestrahlte Bereich – bei Montage der Lampe in einer Höhe von mindestens 2,05 m – ist breit genug, um die gesamte Rennstrecke mit Infrarotlicht auszuleuchten (siehe auch Abbildung 7.7). Die Hardwareanforderung HW2.2.5 wird also durch die o. g. Infrarotlampe erfüllt. Um dies zu erreichen, muss die Lampe jedoch sehr genau

ausgerichtet werden, da sonst ein Teil der äußeren Kurven nicht stark genug bestrahlt wird, um eine zuverlässige Fahrzeugerfassung zu garantieren. Um die Montage des Systems zu vereinfachen, wurde eine weitere baugleiche Infrarotlampe inkl. Netzteil, Kühlkörper und Halterung erworben und ebenfalls in unmittelbarer Nähe der Kamera montiert (siehe Abbildung 7.8). Dies ist relevant, da das System auch als Demonstrator verwendet werden soll. Eine zweite Infrarotlampe vereinfacht nicht nur die Montage des Systems, sie macht es auch robuster gegenüber Störungen. Die Hardwareanforderung HW2.2 und deren Unteranforderungen werden aber bereits durch ein einzelnes Exemplar der o. g. Infrarotlampe erfüllt.

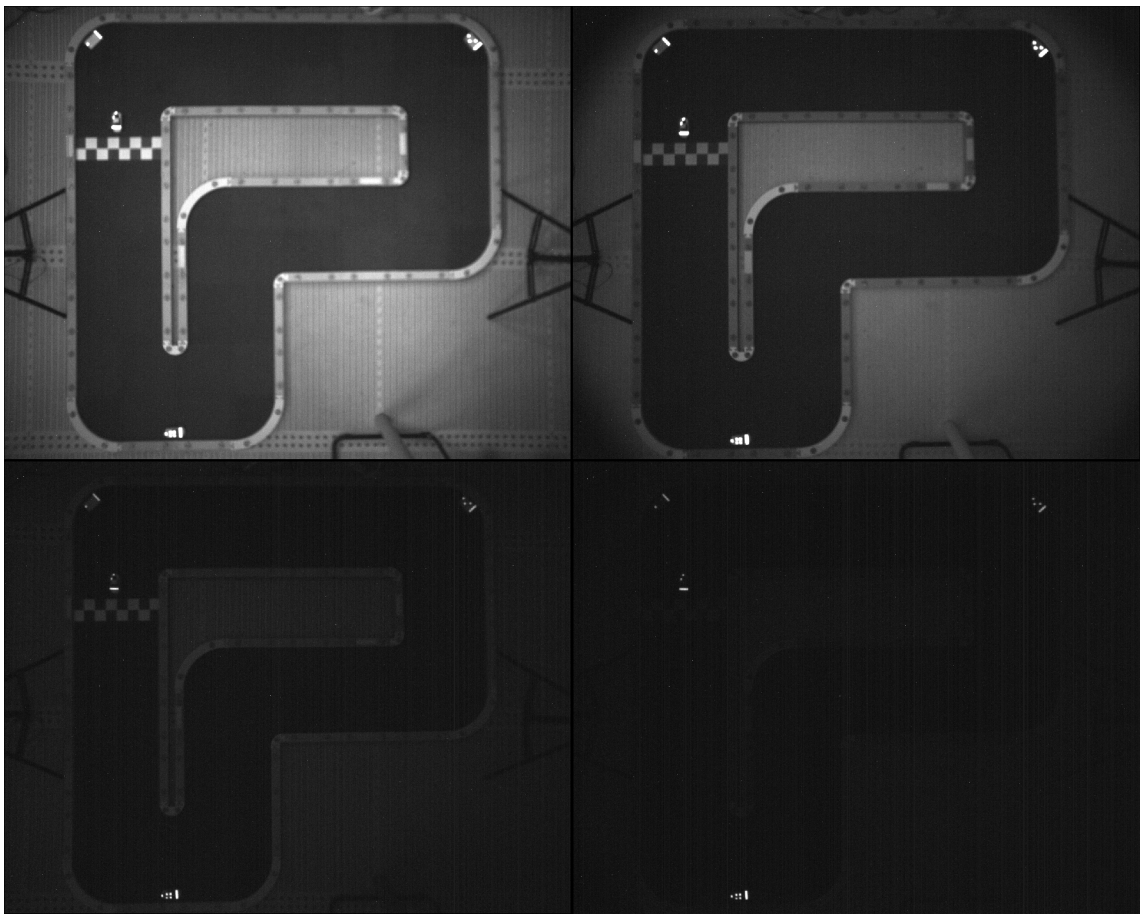


Abbildung 7.7: Kameraaufnahme der Rennstrecke mit Fahrzeugen in vier Varianten. Links ohne / rechts mit Infrarotfilter. Unten mit reduzierter Helligkeit. Bei allen Aufnahmen kam nur eine Infrarotlampe zum Einsatz und es wurde die selbe Raumbeleuchtung verwendet, während direkte Sonneneinstrahlung verhindert wurde.

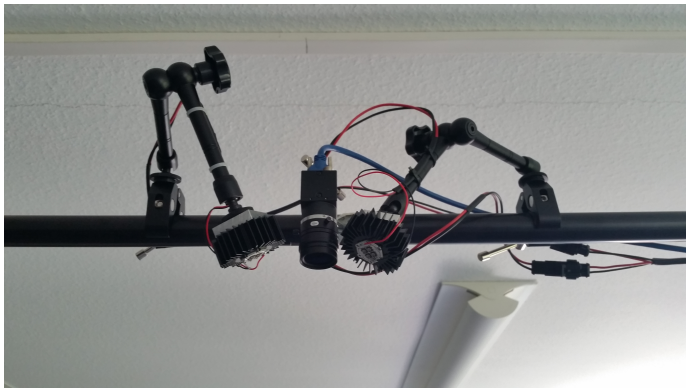


Abbildung 7.8: Kamera und Verbesserte Beleuchtungseinheit mit zwei Infrarotlampen.

7.2.4 Infrarotfilter

Der Infrarotfilter des Typs *MIDOPT FIL LP780/27* [Fil] absorbiert einen wesentlichen Teil des sichtbaren Umgebungslichtes. Da er darauf ausgelegt ist, nur Lichtwellen mit einer Länge unter 780 nm herauszufiltern, können die von der in Abschnitt 7.2.3 beschriebenen Infrarotlampe emittierten 850 nm langen Strahlen den Filter passieren. Abbildung 7.7 zeigt, dass dies – insbesondere bei abgedunkeltem Bild – die Fahrzeugerfassung vereinfacht, da die Reflexionsmarker der Fahrzeuge sich bei Verwendung des Filters im Bild stärker von der Umgebung abheben. Der verwendete Infrarotfilter erfüllt also die Hardwareanforderungen HW2.3.3.2 und HW2.3.3.3. Auch Anforderung HW2.3.3.1 ist erfüllt, da der Filter sich direkt auf das in Abschnitt 7.2.2 beschriebene Objektiv aufschrauben lässt. Somit ist HW2.3.3 insgesamt erfüllt.

Direkte Sonneneinstrahlung im Aufnahmebereich der Kamera sollte vermieden werden, da das Sonnenlicht im Vergleich zu künstlichem Umgebungslicht relativ viel Infrarotstrahlung enthält und dadurch die Objekterfassung stören kann (siehe Abbildung 7.9). Aus diesem Grund wurde die Rahmenbedingung Rah1.3 ergänzt, welche vorgibt, dass die Minimum Bounding Box der Rennstrecke während des Systembetriebs nicht direkter Sonneneinstrahlung ausgesetzt sein darf.

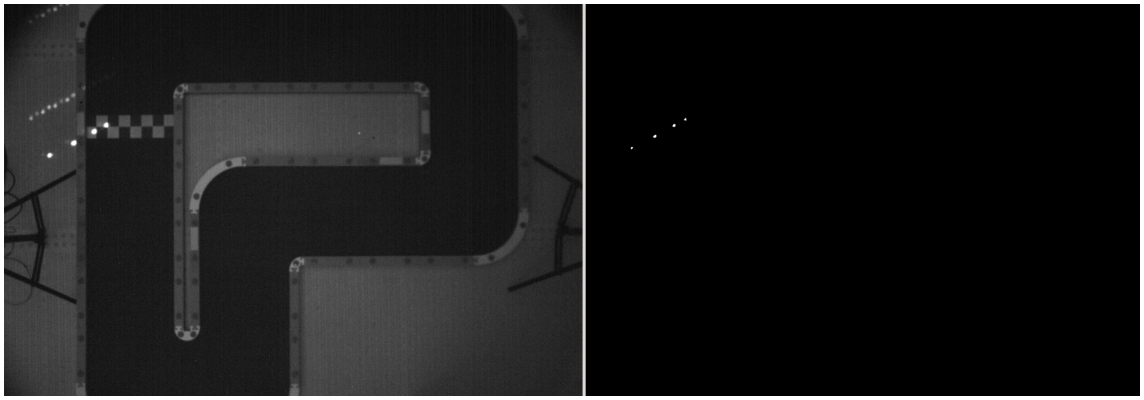


Abbildung 7.9: Störung der Bildererfassung durch direkte Sonneneinstrahlung. Auf der linken Seite ist ein Ausschnitt des ungefilterten Kamerabildes zu sehen. Die rechte Seite zeigt das daraus erzeugte Binärbild.

7.2.5 Gestell

Das in Abbildung 7.4 gezeigte Gestell ermöglicht es die Kamera und die Beleuchtungskomponenten der Positionsbestimmung in der gewünschten vertikalen und horizontalen Position stabil zu montieren. Die Hardwareanforderungen HW2.4.1 bis HW2.4.3 sind somit erfüllt. Die Halterungen der Komponenten erlauben außerdem eine gezielte Ausrichtung. Die Beleuchtungseinheit und die Bilderfassungskomponenten wurden – gemäß Hardwareanforderung HW2.4.4 – so dicht wie möglich nebeneinander am Gestell befestigt. Die Hardwareanforderung HW2.4 ist also insgesamt erfüllt.

7.2.6 Rechner

Der verwendete Rechner *OptiPlex 7020MT* des Herstellers *Dell* verfügt über den nötigen USB3-Anschluss, so dass die Bilder der Kamera mit einer Framerate von annähernd 150 Hz an den Rechner übermittelt werden können (siehe Abschnitt 7.2.1). Die Hardwareanforderung HW2.1 ist somit erfüllt.

7.2.7 Zusammenfassung

- Die Hardwareanforderung HW2.1 bzgl. des Rechners wird erfüllt.
- Die verwendete Infrarotlampe, mit Kühlkörper und Netzteil erfüllt die Hardwareanforderung HW2.2 an die Beleuchtungseinheit inkl. aller Unteranforderungen.
- Die Kamera, in Kombination mit dem Objektiv und dem Infrarotfilter erfüllt die

Hardwareanforderungen HW2.3.1, HW2.3.2 und HW2.3.3 inkl. aller Unteranforderungen und somit die gesamte Anforderung HW2.3 an die Bilderfassung.

- Die Hardwareanforderung HW2.4 an das Gestell wird inkl. aller Unteranforderungen erfüllt.

→ Die gesamte Basishardwareanforderung HW2 für das Subsystem Positionsbestimmung ist erfüllt.

7.3 Subsystem 3: Control-Unit

Im Folgenden werden die Hardwarekomponenten des Subsystems Control-Unit evaluiert. Das Foto 7.10 zeigt den Rechner, auf dem die Regelungssoftware ausgeführt wird. Auf dem Foto 7.11 ist das STM32F4-Discovery Board abgebildet, welches zur digital-analog Wandlung dient.

Abbildung 7.10: Foto des Rechners des Subsystem Control-Unit.

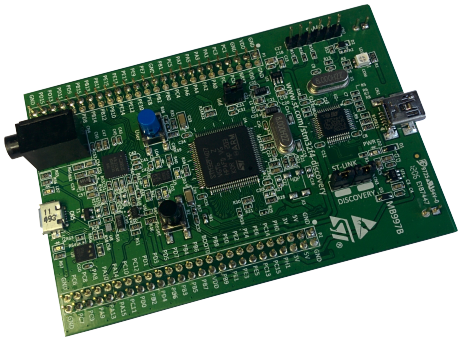


Abbildung 7.11: Foto eines STM32F4 Discovery Boards, welches zur digital-analog Wandlung dient.

Durch die auf den Fotos gezeigten Hardwarekomponenten sind die Anforderungen HW3.1 und HW3.2 erfüllt. Das in Foto 7.11 zu sehende Entwicklerboard ist vom Typ *STM32F4 Discovery* [Ent], welches zwei integrierte DAC-Bausteine und eine USB-Schnittstelle besitzt. Somit sind die Anforderungen HW3.2.1 sowie HW3.2.2 evaluiert. Das Entwicklungsboard ist bedingt durch das Testen der Softwareanforderungen (siehe Testfalldokument) SW2.2.2 und SW2.2.3 in der Lage die digitalen Regelungssignale des Rechners in Spannungsbereiche zwischen 0,00 und 3,00 V umzuwandeln und an die Fernsteuerung weiterzuleiten. Dadurch werden die Hardwareanforderungen HW3.2.3 und HW3.2.4 erfüllt.

Die Hardwareanforderung, dass die Fernsteuerung ein Fahrzeug steuern können muss (HW3.3.1) sowie die Hardwareanforderung, dass die Fernsteuerung mit dem entsprechenden Fahrzeug kompatibel sein muss (HW3.3.2) wird durch den verwendeten Typ der Fernsteuerung (*Perfex KT-18* [Fer]) mit den dazu passenden Fahrzeugen (*dNaNo*-RC-Cars (siehe Anforderung HW1.2) ”out-of-the-box” realisiert. Die Potentiometer der Fernsteuerung wurden, bedingt dadurch das sie sich im Gehäuse der Fernsteuerung und nicht auf der Platine befinden, deaktiviert. Dadurch sind die Hardwareanforderungen HW3.3.3 und HW3.3.4 evaluiert. Durch die Erfüllung der Hardwareanforderungen HW3.3.1 bis HW3.3.4 ergibt sich die Erfüllung der Anforderung, dass die Control-Unit mindestens eine Fernsteuerung besitzt (HW3.3).

Der DA-Wandler wurde von den Mitgliedern der Projektgruppe *RCCARS* selbst konstruiert, daher ist auch die nicht-funktionale Hardwareanforderung NHW1 erfüllt. Abbildung 7.12 zeigt die erste Version des DA-Wandlers.

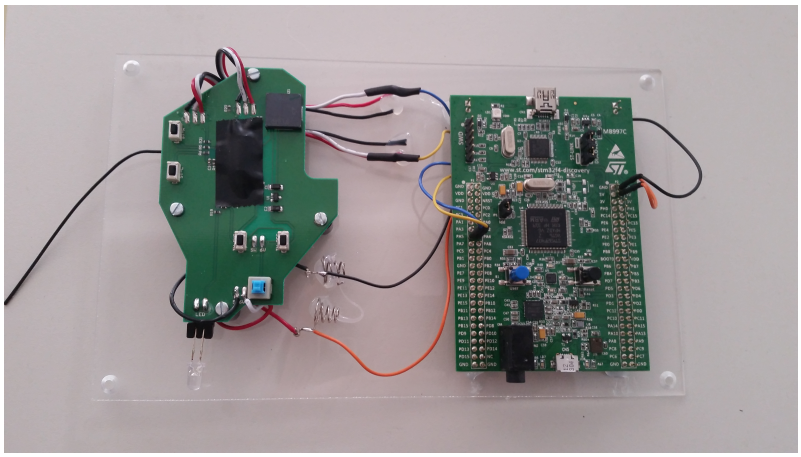


Abbildung 7.12: Foto des mit der Fernsteuerung verbundenen Entwicklungsboards.

Abbildung 7.13 zeigt die zweite Version des DA-Wandlers, welche mit dieser Weiterentwicklung zur CarControl umbenannt wurde. Die Platine der Fernsteuerung wurde wieder in dessen Gehäuse eingesetzt. Über das Batteriefach führt ein Kabel, welches die abgetrennten Potentiometer und die Platine der Fernsteuerung des Entwicklungsboards miteinander verbinden. Die Funktionalität der Signalumwandlung vom Rechner erhaltener Regelungssignale in analoge Spannung ist weiterhin möglich, sodass sämtliche Anforderungen weiterhin erfüllt bleiben. Zusätzlich enthält die CarControl weitere Funktionen, welche in Kapitelabschnitt 10.3.2 beschrieben werden.

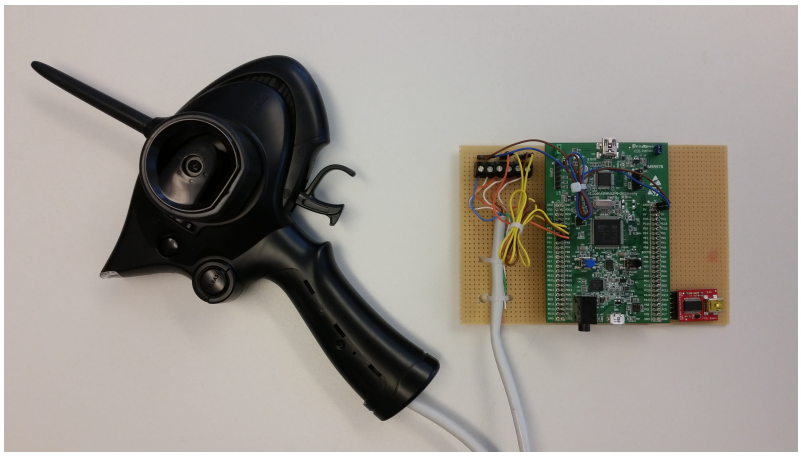


Abbildung 7.13: Zweite Version des DA-Wandlers. Die CarControl.

7.3.1 Zusammenfassung

- Die Hardwareanforderung HW3.1 bzgl. des Rechners wird erfüllt.
- Das Entwicklungsboard besitzt eine USB-Schnittstelle, zwei DAC, wandelt Regelungssignale im Spannungsbereich von 0 und 3 V um, leitet diese Spannungen an die Fernsteuerung weiter und setzt dies alles in maximal 10 ms um wodurch die HW3.2 erfüllt ist.
- Die Fernsteuerung kann ein entsprechendes Fahrzeug steuern, mit welchem sie kompatibel ist. Zusätzlich ist die Fernsteuerung mit dem Entwicklungsboard verbunden und durch die Potentiometer ersetzt. Daraus ergibt sich die Erfüllung der HW3.3.

→ Die gesamte Basishardwareanforderung HW3 für das Subsystem Control-Unit ist erfüllt.

7.4 Subsystem 4: Rennstrecke

Die verwendete Streckenbegrenzung grenzt den für das Fahrzeug befahrbaren vom unbefahrbaren Bereich ab. Im Rahmen der Untersuchungen der Fahrzeugdynamik hat sich herausgestellt, dass das Fahrzeug selbst mit einer hohen Geschwindigkeit mit der Rennstreckenbegrenzung kollidieren kann ohne dabei den befahrbaren Bereich zu verlassen. Außerdem ist der Verlauf der Rennstrecke offensichtlich eine geschlossene Runde. Damit ist die Hardwareanforderung HW4.1 und HW4.2 erfüllt. Obwohl die Rennstrecke laut Hersteller für die *Kyosho* Fahrzeuge genormt ist, muss vor dem Rennbetrieb darauf geachtet werden, dass die einzelnen Streckenkomponenten so nahtlos wie möglich ineinander übergehen. Durch die Herstellerangabe ist die Hardwareanforderung HW4.3 erfüllt.

8 Systemarchitektur und Schnittstellen

Wie bereits in Kapitelabschnitt 4 beschrieben, setzt sich das Gesamtsystem aus verschiedenen Komponenten zusammen, welche im Rahmen des Kapitelabschnitt Systemarchitektur (8.1) nochmals zusammengefasst werden. Die jeweiligen Schnittstellen werden im folgenden zuerst Textuell erläutert und abschließend tabellarisch zusammengefasst.

8.1 Systemarchitektur

Wie bereits in Kapitelabschnitt 4 beschrieben, setzt sich das System aus den vier Subsystemen Fahrzeug (Subsystem 1), Positionsbestimmung (Subsystem 2), Control-Unit (Subsystem 3) und Rennstrecke (Subsystem 4) zusammen (siehe Abbildung 3.8). Dabei besteht das Subsystem Fahrzeug aus dem Fahrzeug selbst und der auf dem Fahrzeug befestigten Reflexionsmarker. Das Subsystem Positionsbestimmung besteht aus dem Gestell, der Kamera, dem Objektiv, dem Infrarotfilter und dem Rechner. Die Control-Unit besteht aus dem Rechner und der CarControl. Der Rechner ist im ersten Szenario der Selbe wie der Rechner der Positionsbestimmung. Das Subsystem Rennstrecke besteht aus zusammensteckbaren Streckensegmenten und den Reflexionsmarkern.

Aus diesem Aufbau ergibt sich ein Signalfluss, welcher bereits in Kapitelabschnitt 4.8 beschrieben und in Abbildung 4.7 dargestellt ist. Das Fahrzeug befindet sich auf der Rennstrecke und wird mit Infrarotlicht beleuchtet. Die Kamera ist in der Lage das Fahrzeug bildlich zu erfassen und mittels Infrarotfilters und -lichtes die Reflexionsmarker des Fahrzeugs stärker sichtbar zu machen. Durch das verwendete Weitwinkelobjektiv kann die gesamte Rennstrecke und somit das Fahrzeug an jedem Punkt der Rennstrecke erfasst werden. Der Rechner berechnet aus dem erfassten Bild der Kamera die Fahrzeugpose und die entsprechenden Regelungswerte für das Fahrzeug. Die Regelungswerte werden anschließend über die CarControl - eine umgebaute Fernsteuerung mit einem Entwicklungsboard - an das Fahrzeug gesendet.

Damit ein Administrator das System überwachen und bedienen bzw. zwischen den Systemzuständen (siehe Kapitelabschnitt 4.6, Abbildung 4.6) wechseln kann, wird eine Systemsteuerungssoftware implementiert. Die Systemsteuerungssoftware soll dem Administrator ein Benutzerinterface bieten, womit unter Anderem das Starten und Stoppen des Rennbetriebes und das Auslesen von Informationen wie z.B. Fehlercodes möglich ist.

Das Gesamtsystem lässt sich insgesamt in fünf Komponenten unterteilen, welche aus den vier Subsystemen und der Systemsteuerungssoftware besteht. Hieraus ergibt sich ein Kommunikationskreislauf, wie in Abbildung 8.1 dargestellt ist. Im Folgenden werden die Ein- und Ausgabeschnittstellen jeder Komponente definiert.

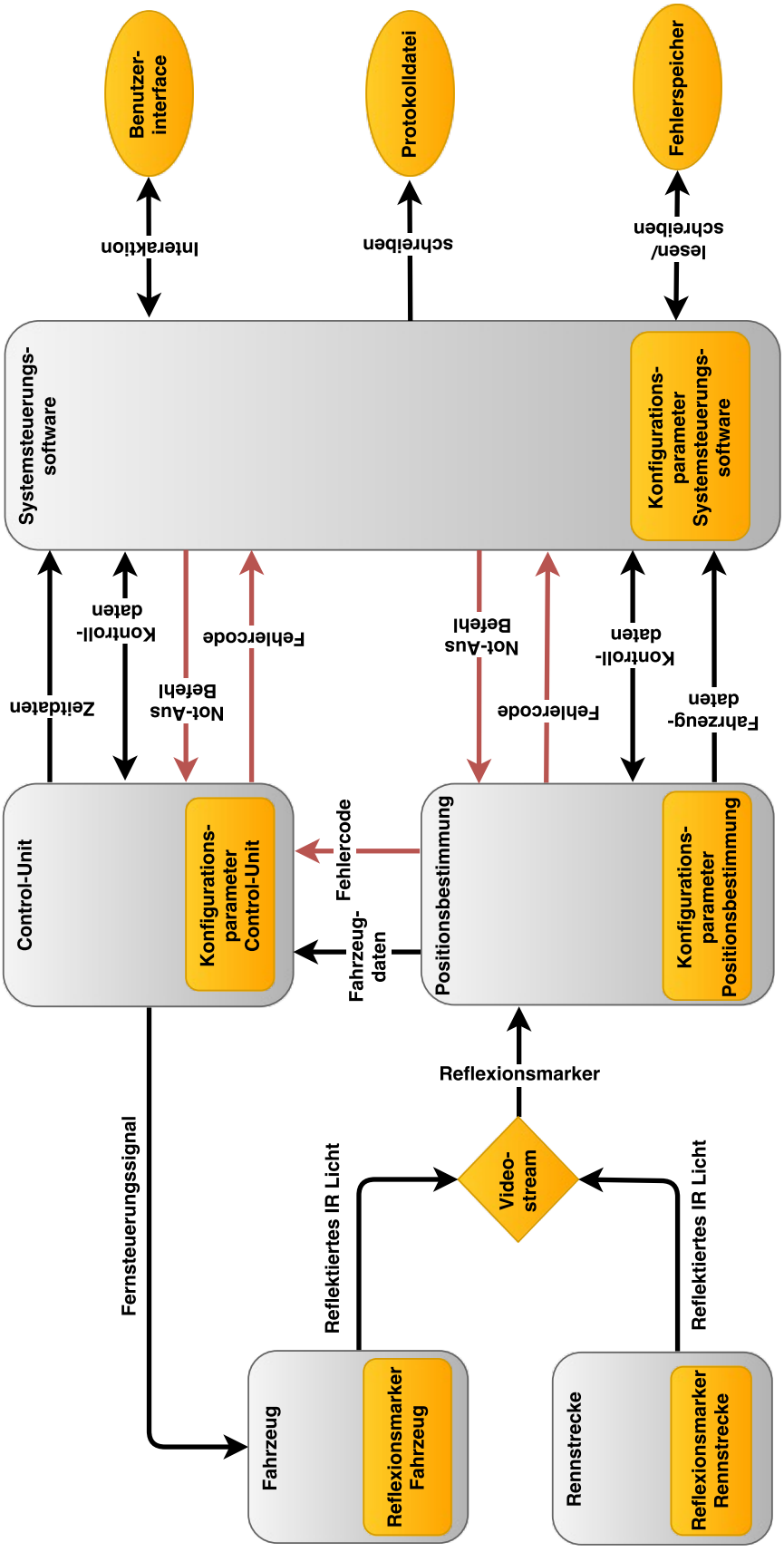


Abbildung 8.1: Schematische Darstellung der Schnittstellen des Systems

8.2 Subsystem 1: Fahrzeug

Das Subsystem Fahrzeug stellt die Schnittstelle zwischen der Control-Unit und der Positionsbestimmung dar. Das Fahrzeug empfängt die digitalen Fernsteuerungssignale für die Längs- und Querführung von der Control-Unit und wird über die Reflexionsmarker von der Positionsbestimmung ständig erkannt. Das Fahrzeug hat die im Folgenden definierten Ein- und Ausgabeschnittstellen.

8.2.1 Eingabeschnittstellen

- **Fernsteuerungssignal**
  Das Fahrzeug ist mit der Control-Unit drahtlos verbunden und empfängt digitale Fernsteuerungssignale von der CarControl der Control-Unit. Diese Schnittstelle ist durch den Hersteller vorgegeben und durch die Projektgruppe nicht beeinflussbar.

- **Infrarotlicht** Das Fahrzeug trägt Reflexionsmarker, mit welchen das Fahrzeug ständig mit Infrarotlicht beleuchtet wird.

8.2.2 Ausgabeschnittstellen

- **Reflektiertes Infrarotlicht**
  Das Infrarotlicht, welches auf die Reflexionsmarker fällt, wird zur Lichtquelle zurück reflektiert. Dieses Verfahren stellt eine optische Verbindung zwischen der Positionsbestimmung und dem Fahrzeug dar. Das Infrarotlicht wird im Einfallswinkel zurück zur Lichtquelle reflektiert. Direkt neben der Lichtquelle ist eine Kamera befestigt, welche das reflektierte Infrarotlicht der Reflexionsmarker erfassen soll. Die Reflexionsmarker sind hierbei in einem festgelegten Muster angeordnet (siehe Abbildung 8.2). Das Muster setzt sich aus Rechtecken zusammen. Der vordere Balken und das hintere Quadrat geben die Richtung an. Die vier mittig platzierten Quadrate geben die Identität an. Die Identität, welche sich aus den vier quadratischen Markern zusammensetzt, lässt eine binäre Codierung von bis zu 16 Fahrzeugen zu.

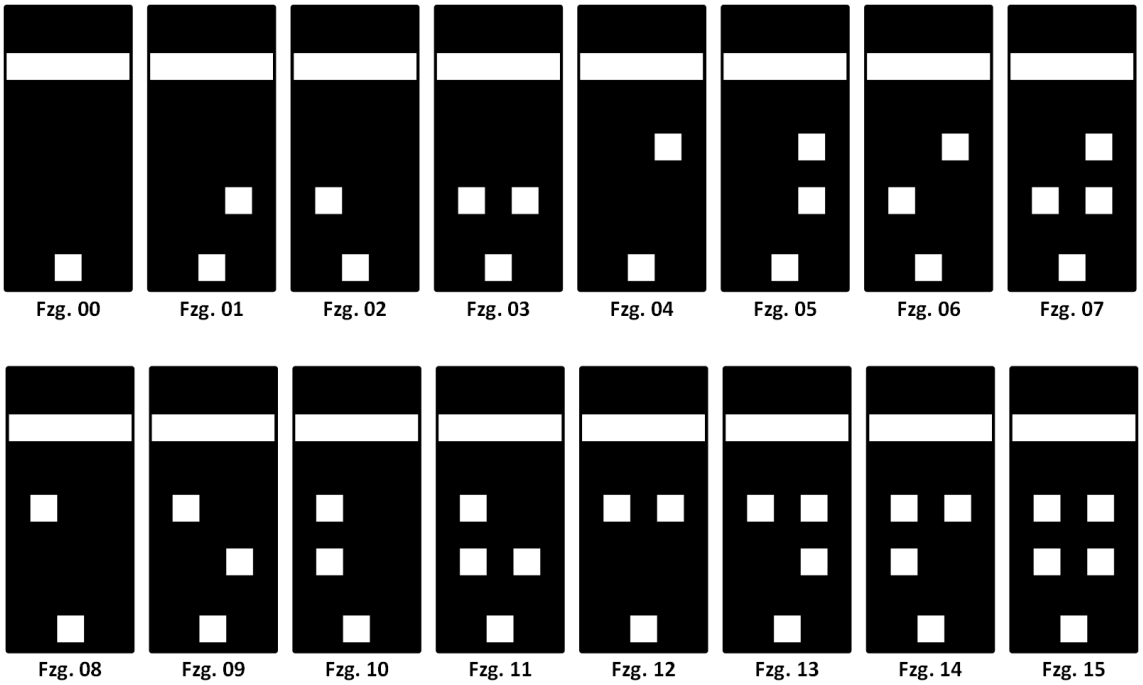


Abbildung 8.2: Fahrzeugmuster. Anordnung der Marker.

Eingabeschnittstellen - Subsystem Fahrzeug			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Digitales Fernsteuerungssignal	—	—	—
Infrarotlicht	—	—	—

Tabelle 8.1: Eingabeschnittstellen des Subsystems Fahrzeug

Ausgabeschnittstellen - Subsystem Fahrzeug			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Reflektiertes Infrarotlicht	Fahrzeug ID (binär codiert)	int	0 . . . 15

Tabelle 8.2: Ausgabeschnittstellen des Subsystems Fahrzeug

8.3 Subsystem 2: Positionsbestimmung

Die Positionsbestimmung ist für die Erfassung der Fahrzeugpose und dessen Weiterleitung an die Control-Unit verantwortlich. Für den Betrieb der Positionsbestimmung werden Konfigurationsparameter und ein kontinuierlicher Videostream der Kamera benötigt.

Des Weiteren soll die Positionsbestimmung Kontrollbefehle erhalten, um die Fahrzeugerkennung zu starten oder zu stoppen und gegebenenfalls einen Not-Aus auslösen zu können. Mithilfe des Videostreams ist die Positionsbestimmung in der Lage, die Pose der Fahrzeuge zu ermitteln und - falls nötig - einen Fehlercode auszugeben. Die definierten Ein- und Ausgabeschnittstellen der Positionsbestimmung werden im Folgenden beschrieben.

8.3.1 Konfigurationsparameter

Vor der Initialisierung der Positionsbestimmung müssen Konfigurationsparameter für den Betrieb festgelegt werden. Hierzu gehören Parameter zum Verhalten der Software, Variablen für die Bildverarbeitung und der Kamerakalibrierung. Alle diese Parameter werden in einer Textdatei zusammengefasst und von der Positionsbestimmungssoftware vor der Initialisierung eingelesen. Die Parameter zum Einstellen des Verhaltens der Software sind jeweils als Integer-Wert angegeben. So kann u.a. eingestellt werden, ob die Positionsbestimmungssoftware auf Befehle von der Systemsteuerungssoftware wartet oder beispielsweise der Videostream ausgegeben werden soll. Die verschiedenen Variablen für die Bildverarbeitung werden jeweils als Integer-Wert (int) angegeben und ermöglichen Anpassungen am Filter. Die Variablen für der Kamerakalibrierung sind als Double-Wert (double) angegeben. Sämtliche Parameter sind in Kapitelabschnitt 14.4 mit dem entsprechenden Datentypen, dem Wertebereich und einer Beschreibung aufgelistet.

8.3.2 Eingabeschnittstellen

- **Videostream**
  Der Videostream liefert der Positionsbestimmung die benötigten Bilder zum Ermitteln der Fahrzeugposen. Der Videostream wird über eine USB 3.0 Verbindung zwischen der Kamera und dem Rechner realisiert.

- **Kontrollbefehl**
  Für das Wechseln der Systemzustände muss die Systemsteuerungssoftware einen Kontrollbefehl senden. Mittels Kontrollbefehls als Integer-Wert (int) kann die Systemsteuerungssoftware das Wechseln des Systemzustandes der Positionsbestimmung verlangen. Mit dem Wert 0 ist der Systemustand ”Standby”/”Rennbetrieb” kodiert. Mit dem Wert 2 ist der Systemzustand ”Initialisierung” kodiert. Mit dem Wert 3 ist der Systemzustand ”Inspektion” kodiert. Mit dem Wert 4 ist der Systemzustand ”Aus” kodiert.

8.3.3 Ausgabeschnittstellen

- **Fahrzeugdaten**
  Damit die Control-Unit den aktuellen Standpunkt des Fahrzeugs kennt und auf dieser Grundlage das Fahrzeug über die Rennstrecke steuern kann, wird ein Fahrzeugdatenpaket an die Control-Unit und die Systemsteuerungssoftware gesendet. Die Positionsbestimmung ermittelt die X und Y-Koordinaten sämtlicher erkannter Fahrzeuge und stellt diese als Double-Wert (double) im Bereich 0,00 bis X_{max} bzw. 0,00 bis Y_{max} dar, wobei X_{max} der Breite und Y_{max} der Länge der Rennstrecke in Zentimetern entspricht. Zusätzlich werden die Ausrichtungswinkel sämtlicher erkannter Fahrzeuge als Double-Wert (double), die Verfügbarkeit der Fahrzeuge als Integer-Wert (int) und ein Zeitstempel vom Datentypen timeval mitgeteilt. Letzterer dient der Feststellung des Alters der Fahrzeugposen und als Fingerabdruck des Netzwerkdatenpakets.

- **Kontrolldaten**
  Um der Systemsteuerungssoftware eine erfolgreiche Initialisierung der Positionsbestimmung mitzuteilen, wird ein Kontrollsignal versendet. Dieses ist als Integer-Wert (int) 1 kodiert.

- **Fehlercode**
  Die Positionsbestimmung ist in der Lange während des Betriebs Fehler zu erkennen. Die Fehler werden in Form eines Fehlercodes ausgeben. Der Fehlercode wird

als Integer-Wert (int) an die Systemsteuerungssoftware und die Control-Unit versendet. Jeder Fehlercode entspricht dabei einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

Eingabeschnittstellen - Subsystem Positionsbestimmung			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Videostream	Videostream	CameraStream IOManager	object
Kontrollbefehl	Wechsel d. Systemzustände	int	0 . . . 4

Tabelle 8.3: Eingabeschnittstellen des Subsystems Positionsbestimmung

Ausgabeschnittstellen - Subsystem Positionsbestimmung			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1
	X-Koordinate	double[16]	0 . . . 236
	Y-Koordinate	double[16]	0 . . . 236
	Ausrichtung	double[16]	0 . . . 360
	Zeitstempel der Fahrzeugpose	timeval	—
Kontrollbefehl	Initialisierung erfolgreich	int	1
Fehlercode	Fehlercode	int	faultcode > 0

Tabelle 8.4: Ausgabeschnittstellen des Subsystems Positionsbestimmung

8.4 Subsystem 3: Control-Unit

In diesem Kapitelabschnitt werden die Schnittstellen der Control-Unit vorgestellt, welche für die Kontrolle und die Ansteuerung des Rennfahrzeugs zuständig ist. Die Control-Unit empfängt Fahrzeugdatenpakete sowie etwaige Fehlercodes von der Positionsbestimmung, der Systemsteuerungssoftware oder anderen Control-Units sowie Kontrollbefehle und einen Not-Aus Befehl von der Systemsteuerungssoftware. Des Weiteren liest die Control-Unit einmalig Konfigurationsparameter beim Systemstart ein, kann Fehlercodes und Zeitstempel an die Systemsteuerungssoftware, digitale Fernsteuerungssignale an das Fahrzeug und Kontrollbefehle an die Systemsteuerungssoftware senden. Die definierten Ein- und Ausgabeschnittstellen sowie die Konfigurationsparameter werden im Folgenden detaillierter erläutert.

8.4.1 Konfigurationsparameter

Im Rahmen einer Vorinitialisierung der Control-Unit müssen Konfigurationsparameter für die Initialisierung und den Betrieb festgelegt werden. Hierzu gehören u.a. die Fahrzeug ID, die Schnittstelle zur CarControl und Parameter für die Netzwerkkommunikation. Alle diese Parameter werden in einer Textdatei zusammengefasst und zusammen mit den Rennstreckendaten während der Initialisierung der Control-Unit eingelesen.
Sämtliche Parameter sind in Kapitelabschnitt 14.5 mit dem entsprechenden Datentypen, dem Wertebereich und einer Beschreibung aufgelistet.

8.4.2 Eingabeschnittstellen

- **Fahrzeugdaten**
  Damit die Control-Unit den aktuellen Standpunkt des Fahrzeugs kennt und auf dieser Grundlage das Fahrzeug über die Rennstrecke steuern kann, wird das Fahrzeugdatenpaket von der Positionsbestimmung empfangen.
  Die Control-Unit erhält von der Positionsbestimmung die X- und Y-Koordinaten sämtlicher erkannter Fahrzeuge als Double-Wert (double) im Bereich 0,00 bis X_{max} bzw. 0,00 bis Y_{max}, wobei X_{max} der Breite und Y_{max} der Länge der Rennstrecke in Zentimetern entspricht. Zusätzlich werden die Ausrichtungswinkel sämtlicher erkannter Fahrzeuge als Double-Wert (double), die Verfügbarkeit der Fahrzeuge und ein Zeitstempel vom Datentypen timeval für die Feststellung des Alters der Fahrzeugposen und als Fingerabdruck des Netzwerkdatenpakets mitgeteilt.

- **Kontrollbefehl**
  Für das Wechseln der Systemzustände muss die Systemsteuerungssoftware einen Kontrollbefehl senden. Mittels Kontrollbefehls als Integer-Wert (int) kann die Systemsteuerungssoftware das Wechseln des Systemzustandes der Control-Unit verlangen. Mit dem Wert 0 ist der Systemzustand ”Standby” kodiert. Mit dem Wert 1 ist der Systemzustand ”Rennbetrieb” kodiert. Mit dem Wert 2 ist der Systemzustand ”Initialisierung” kodiert. Mit dem Wert 3 ist der Systemzustand ”Inspektion” kodiert. Mit dem Wert 4 ist der Systemzustand ”Aus” kodiert.

- **Fehlercode**
  Damit die Control-Unit auf sicherheitskritische Fehlfunktionen des Systems schnellstmöglich reagieren kann, empfängt diese Fehlercodes der einzelnen Komponenten. Der Fehlercode wird als Integer-Wert (int) empfangen. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

- **Not-Aus Befehl**
  Im Fehlerfall muss die Control-Unit dazu in der Lage sein, die Fahrzeuge sofort anzuhalten. Hierzu erhält die Control-Unit einen Not-Aus Befehl. Der Not-Aus Befehl wird als Integer-Wert (int) 1 empfangen.

8.4.3 Ausgabeschnittstellen

- **Fernsteuerungssignal**
  Die Control-Unit ist drahtlos mit dem Fahrzeug verbunden und sendet digitale Fernsteuerungssignale an dieses. Diese Schnittstelle ist durch den Hersteller vorgegeben und durch die Projektgruppe nicht beeinflussbar.

- **Kontrollbefehl**
  Um der Systemsteuerungssoftware eine erfolgreiche Initialisierung der Control-Unit mitzuteilen, wird ein Kontrollbefehl versendet. Dieses ist als Integer-Wert (int) 1 kodiert.

- **Fehlercode**
  Die Control-Unit ist zur Fehlererkennung während des Betriebsablaufes in der Lage und gibt diese in Form eines Fehlercodes aus. Der Fehlercode wird als Integer-Wert (int) an die Systemsteuerungssoftware und weitere Control-Unit's übermittelt. Die Fehlercodes entsprechen jeweils einem fest definierten Fehler, welche in Kapitelabschnitt 14.1 aufgelistet sind.

- **Zeitdaten**
  Um die Einhaltung der Realzeitanforderungen überprüfen zu können, wird der Zeitstempel der an das Fahrzeug gesendeten Regelungswerte, ausgegeben. Der Zeitstempel ist vom Datentyp timeval und wird zusammen mit dem Zeitstempel der Fahrzeugpose an die Systemsteuerungssoftware übermittelt.

Eingabeschnittstellen - Control-Unit			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1
	X-Koordinate	double[16]	0 … 236
	Y-Koordinate	double[16]	0 … 236
	Ausrichtung	double[16]	0 … 360
	Zeitstempel der Fahrzeugpose	timeval	—
Kontrollbefehl	Wechsel d. Systemzustände	int	0 … 4
Fehlercode	Fehlercode	int	faultcode > 0
Not-Aus Befehl	Wechsel in den Fehlerzustand	int	1

Tabelle 8.5: Eingabeschnittstellen der Control-Unit

Ausgabeschnittstellen - Control-Unit			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Digitales Fernsteuerungssignal	—	—	—
Kontrollbefehl	Initialisierung erfolgreich	int	1
Fehlercode	Fehlercode	int	faultcode > 0
Zeitdaten	Zeitstempel der Fahrzeugpose	timeval	—
	Zeitstempel der gesendeten Regelungswerte	timeval	—

Tabelle 8.6: Ausgabeschnittstellen der Control-Unit

8.4.4 Struktur der Control-Unit und interne Schnittstellen

Die Control-Unit unterteilt sich, wie in Abbildung 8.3 dargestellt ist, in die drei Komponenten Wrapper Code, *SCADE* Modell und CarControl. Diese Aufteilung ist aufgrund der Hardwareaufteilung und der Nutzung des *SCADE* Modells notwendig. *SCADE* generiert aus dem gebauten Modell mehrere C-Module, welche in einem C-Programmcode integriert werden. Diese Module stellen nur die modellierten Berechnungsroutinen zur Verfügung. Es fehlen somit die Ein- und Ausgabeschnittstellen zum System. Diese Schnittstellen werden mittels des Wrapper Codes realisiert. Die CarControl ist eine eigene Hardwarekomponente mit einer eigenen Software, welche auf dem Mikrocontroller der CarControl betrieben wird und mit dem Rechner der Control-Unit verbunden ist.

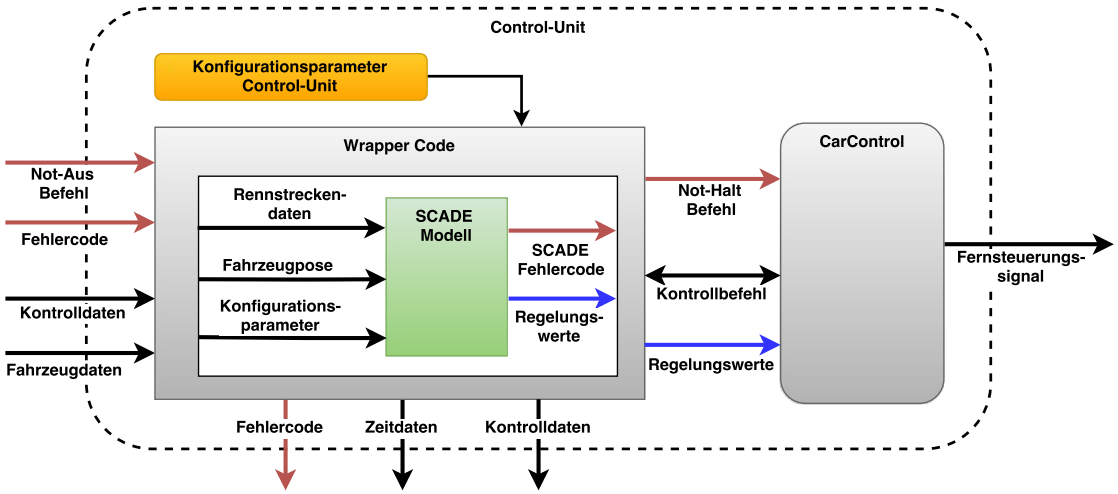


Abbildung 8.3: Schemeatische Darstellung der Control-Unit und der einzelnen Komponenten mit den Ein- und Ausgabeschnittstellen.

Im Folgenden werden die einzelnen Bestandteile und ihre Schnittstellen detailliert erläutert.

8.4.4.1 Wrapper Code

Der Wrapper Code hat definierte Ein- und Ausgabeschnittstellen und stellt die Schnittstelle zum *SCADE* Modell sowie der CarControl dar. Er liest die Konfigurationsparameter ein, empfängt das Fahrzeugdatenpaket sowie mögliche Fehlercodes von der Positionsbestimmung. Ebenso empfängt der Wrapper-Code die Kontrollbefehle und den Not-Aus Befehl von der Systemsteuerung und sendet Fehlercodes, Zeitdaten und einen Kontrollbefehl an die Systemsteuerungssoftware und Regelungswerte für das Fahrzeug die im *SCADE* Modell erzeugt werden und einen Not-Halt Befehl an die CarControl weiter. Sämtliche Schnittstellen des Wrapper Codes werden im Folgenden beschrieben.

Eingabeschnittstellen

- **Fahrzeugdaten**
  Der Wrapper-Code nimmt das Fahrzeugdatenpaket entgegen und bereitet dies für die weitere Bearbeitung auf.

- **Kontrollbefehl**
  Der Kontrollbefehl wird als Integer-Wert (int) mit einem Wert von 0 bis 4 von der Systemsteuerungssoftware empfangen. Die 0 steht hierbei für den Systemzustand ”Standby”, die 1 für den Systemzustand ”Rennbetrieb”, die 2 für den Systemzustand ”Initialisierung”, die 3 für den Systemzustand ”Inspektion” und die 4 für

den Systemzustand ”Aus”. Um der ControlUnit mitzuteilen, dass die Verbindung von der CarControl zum Rechner aufgebaut wurde und korrekt arbeitet, erhält die Control-Unit ein ’CC’ als Feedback auf die Nachricht ’C’.

- **Fehlercode**
  Der eingehende Fehlercode (Integer-Wert) wird vom Wrapper-Code angenommen, ausgewertet und entsprechend behandelt. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

- **Not-Aus Befehl**
  Der Not-Aus Befehl wird vom Wrapper-Code angenommen, ausgewertet und entsprechend behandelt.

- **SCADE Fehlercode**
  Das SCADE Modell kann während der Berechnung Fehler ermitteln und diese in Form eines Fehlercodes ausgeben. Der Fehlercode wird als Integer-Wert (int) in einer globalen Variable bereitgestellt. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

- **Regelungswerte**
  Der Wrapper-Code nimmt die von dem *SCADE* Modell bereitgestellten Regelungswerte für die Längs- und Querführung an und leitet diese an die CarControl weiter.

Ausgabeschnittstellen

- **Regelungswerte**
  Damit das Fahrzeug die korrekte Längs- und Querführung umsetzen kann, werden die aus dem *SCADE* Modell berechneten Regelungswerte an die CarControl ausgegeben. Die Regelungswerte werden als Character-Wert (char) in Form einer Byte-Folge für jeweils die Längs- und Querführung ausgegeben. Hierfür wird der Regelungswert mit einem ”S” für die Querführung (Steering) oder einem ”T” für die Längsführung (Throttle) angeführt und mit der Endekennung ”0x0D” abgeschlossen. Alternativ können die Regelungsdaten auch gemeinsam ausgegeben werden. Hierzu wird die Nachricht mit dem Regelungswert für die Längsführung an die Nachricht mit dem Regelungswert für die Querführung angehängt und mit einem ”X” angeführt.

- **Kontrollbefehl**
  Um der Systemsteuerungssoftware eine erfolgreiche Initialisierung der Control-Unit mitzuteilen, wird ein Kontrollbefehl versendet. Dieses ist als Integer-Wert (int) 1 kodiert.
  Um der CarControl mitzuteilen, dass eine Verbindung vom Rechner aus aufgebaut wurde, wird der Character 'C' an die CarControl gesendet. Um einen SoftReset der CarControl auszuführen, wird der Character 'R' gesendet. Um den Wechsel des Betriebsmodus ausführen zu können, wird der Character 'M', gefolgt von der Nummer des jeweiligen Betriebsmodus gesendet.

- **Not-Halt Befehl**
  Damit das Fahrzeug im Fehlerfall sofort die Längs- und Querführung auf die Neutralstellung setzt und das Fahrzeug anhält, wird ein Not-Halt Befehl abgeschickt. Der Not-Halt Befehl wird in Form eines Character-Wertes (char) in Form einer Byte-Folge an die CarControl geschickt. Der Wert lautet hierbei "N", welcher die CarControl ebenfalls in einen sicheren Fehlerzustand überführen soll. Auslöser für den Not-Halt Befehl können intern erkannte Fehler als auch eingehende Fehlercodes oder der Not-Aus Befehl sein.

- **Fehlercode**
  Sowohl der Wrapper Code, als auch das *SCADE* Modell sind zur Fehlererkennung während des Betriebsablaufes in der Lage und geben diese in Form eines Fehlercodes aus. Der Fehlercode wird als Integer-Wert (int) an die Systemsteuerungssoftware und weitere Control-Unit's übermittelt. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

- **Zeitdaten**
  Um die Einhaltung der Realzeitanforderungen überprüfen zu können, wird der Zeitstempel, der an die CarControl gesendeten Regelungswerte, ausgegeben. Der Zeitstempel ist vom Datentyp timeval und wird zusammen mit dem Zeitstempel der Fahrzeugpose an die Systemsteuerungssoftware übermittelt.

- **Rennstreckendaten**
  Um dem *SCADE* Modell mitzuteilen, in welchem Bereich der Rennstrecke sich das Fahrzeug befinden darf und welche Trajektorie es fahren soll, müssen Rennstreckendaten bereitgestellt werden. Die Rennstreckendaten werden in einem zweidimensionalen Array in einer globalen Variable bereitgestellt.

- **Fahrzeugpose**
  Um dem *SCADE* Modell mitzuteilen, in welchem Bereich der Rennstrecke sich das

Fahrzeug aktuell befindet und wie es ausgerichtet ist, wird die Fahrzeugpose zu der entsprechenden Fahrzeug ID bereitgestellt. Die Fahrzeugpose wird dabei in globalen Variablen bereitgestellt. Die X- und Y-Koordinate und der Ausrichtungswinkel werden in jeweils einem Double-Wert (double) gespeichert.

- **Konfigurationsparameter**
  Der Wrapper Code gibt einen Teil der eingelesenen Konfigurationsparamteter der Control-Unit an das SCADE Modell weiter um Funktionen wie die Bandenkollisionserkennung zu (de-)aktivieren oder die Trimmung des Fahrzeuges einzustellen.

Eingabeschnittstellen - Wrapper Code			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1
	X-Koordinate	double[16]	0 … 236
	Y-Koordinate	double[16]	0 … 236
	Ausrichtung	double[16]	0 … 360
	Zeitstempel der Fahrzeugpose	timeval	—
Kontrollbefehl	Wechsel d. Betriebszustands	byte	0 … 4
Fehlercode	Fehlercode	int	faultcode > 0
Not-Aus Befehl	Wechsel in den Fehlerzustand	int	1
SCADE Fehlercode	SCADE Fehlercode	int	faultcode > 0
Regelungswerte	Querführung	int	280 … 3980
	Längsführung	int	1100 … 3000

Tabelle 8.7: Eingabeschnittstellen des Wrapper Codes

Ausgabeschnittstellen - Wrapper Code			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Regelungswerte	Querführung	byte	0x0D ... 0x53
	Längsführung	byte	0x0D ... 0x54
Kontrollbefehl	Initialisierung erfolgreich	int	1
	Prüfungsanfrage d. Verbindung	byte	0x43
	Auslösen eines SoftReset	byte	0x52
	Wechsel des Modus	byte	0x4D
Not-Halt Befehl	Not-Halt des Fahrzeugs	byte	0x4E
Fehlercode	Fehlercode	int	error > 0
Zeitdaten	Zeitstempel der Fahrzeugpose	timeval	—
	Zeitstempel der gesendeten Regelungswerte	timeval	—
Rennstreckendaten	digitale Abbildung	int[237][237]	0 ... 2
Fahrzeugpose	X-Koordinate	double	0 ... 236
	Y-Koordinate	double	0 ... 236
	Ausrichtung	double	0 ... 360

Tabelle 8.8: Ausgabeschnittstellen des Wrapper Codes

8.4.4.2 SCADE Modell

Das *SCADE* Modell, welches die eigentliche Längs- und Querführung des Fahrzeugs regelt, hat definierte Ein- und Ausgabeschnittstellen zum Wrapper Code, welche im Folgenden beschrieben werden.

Eingabeschnittstellen

- **Rennstreckendaten**
  Damit das *SCADE* Modell die Regelungswerte für das Fahrzeug bestimmen kann, müssen die Rennstreckendaten der Rennstrecke bekannt sein, wo sich das Fahrzeug befinden darf und welche Trajektorie es fahren soll. Die Rennstreckendaten werden in einem zwei-dimensionalen Array in einer globalen Variable bereitgestellt.

- **Fahrzeugpose**
  Damit das *SCADE* Modell die Regelungswerte für das Fahrzeug bestimmen kann, muss die Fahrzeugpose bekannt sein. Die Fahrzeugpose wird in Form einer X-Koordinate, einer Y-Koordinate und der Ausrichtung in jeweils einem Double-Wert

(double) abgebildet. Alle drei Informationen werden in jeweils einer globalen Variable bereitgestellt.

- **Konfigurationsparameter**
  Der Wrapper Code gibt einen Teil der eingelesenen Konfigurationsparamteter der Control-Unit an das SCADE Modell weiter um Funktionen wie die Bandenkollisionserkennung zu (de-)aktivieren oder die Trimmung des Fahrzeuges einzustellen.

Ausgabeschnittstellen

- **Regelungswerte**
  Damit das Fahrzeug die korrekte Längs- und Querführung umsetzen kann, werden Regelungswerte errechnet und ausgegeben. Die Regelungswerte werden als Integer-Wert (int) für jeweils die Längs- und Querführung ausgegeben. Die Längsführung arbeitet in einem Wertebereich von 1100 und 3000. Der Wert 1100 repräsentiert hierbei die volle Vorwärtsbeschleunigung und der Wert 3000 die volle Rückwärtsbeschleunigung. Die Querführung arbeitet in einem Wertebereich von 280 und 3980. Der Wert 280 repräsentiert hierbei den vollen Linkseinschlag und der Wert 3980 den vollen Rechtseinschlag der Vorderachse.

- **SCADE Fehlercode**
  Das SCADE Modell kann während der Berechnung Fehler ermitteln und diese in Form eines Fehlercodes ausgeben. Der Fehlercode wird als Integer-Wert (int) in einer globalen Variable bereitgestellt. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

Eingabeschnittstellen - SCADE Modell			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Rennstreckendaten	digitale Abbildung	int[237][237]	0 ... 2
Fahrzeugpose	X-Koordinate	double	0 ... 236
	Y-Koordinate	double	0 ... 236
	Ausrichtung	double	0 ... 360

Tabelle 8.9: Eingabeschnittstellen des SCADE Modells

Ausgabeschnittstellen - SCADE Modell			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Regelungswerte	Querführung	int	280 ... 3980
	Längsführung	int	1100 ... 3000
SCADE Fehlercode	SCADE Fehlercode	int	error > 0

Tabelle 8.10: Ausgabeschnittstellen des SCADE Modells

8.4.4.3 CarControl

Die CarControl empfängt die errechneten Regelungswerte des *SCADE* Modells und übermittelt diese in Form von digitalen Steuerungssignalen an das Fahrzeug. Die CarControl hat definierte Ein- und Ausgabeschnittstellen, welche im Folgenden beschrieben werden.

Eingabeschnittstellen

- **Regelungswerte**
  Damit das Fahrzeug die korrekte Längs- und Querführung umsetzen kann, werden Regelungswerte empfangen. Die Regelungswerte werden als Integer-Wert für jeweils die Längs- und Querführung bereitgestellt. Die Längsführung arbeitet in einem Wertebereich von 1100 und 3000. Der Wert 1100 repräsentiert hierbei die volle Vorwärtsbeschleunigung und der Wert 3000 die volle Rückwärtsbeschleunigung. Die Querführung arbeitet in einem Wertebereich von 280 und 3980. Der Wert

280 repräsentiert hierbei den vollen Linkseinschlag und der Wert 3980 den vollen Rechtseinschlag der Vorderachse.

- **Kontrollbefehl**
  Um der CarControl mitzuteilen, dass eine Verbindung vom Wrapper Code aus aufgebaut wurde, wird der Character 'C' an die CarControl gesendet. Um einen Soft-Reset auszulösen, wird ein 'R' gesendet. Um den Betriebsmodus der CarControl zu welchseln, wird ein 'M', gefolgt von der Nummer des Betriebsmodus, gesendet.

- **Not-Halt Befehl**
  Damit das Fahrzeug die Längs- und Querführung sofort auf die Neutralstellung bringt und das Fahrzeug somit anhält, wird ein Not-Halt Befehl empfangen. Der Not-Halt Befehl wird in Form einer fest definierten Byte folge empfangen.

Ausgabeschnittstellen

- **Fernsteuerungssignale**
  Die Control-Unit ist drahtlos mit dem Fahrzeug verbunden und sendet digitale Fernsteuerungssignale an dieses. Diese Schnittstelle ist durch den Hersteller vorgegeben und durch die Projektgruppe nicht beeinflussbar.

- **Kontrollbefehl**
  Um dem Wrapper Code mitzuteilen, dass die Verbindung von der CarControl zum Rechner aufgebaut wurde und korrekt arbeitet, erhält die Control-Unit ein 'CC' als Feedback auf die Nachricht 'C'.

Eingabeschnittstellen - CarControlSoftware			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Regelungswerte	Querführung	byte	0x0D ... 0x53
	Längsführung	byte	0x0D ... 0x54
Kontrollbefehl	Prüfungsanfrage d. Verbindung	byte	0x43
	Auslösen eines SoftReset	byte	0x52
	Wechsel des Modus	byte	0x4D
Not-Halt Befehl	Not-Halt des Fahrzeugs	byte	0x4E

Tabelle 8.11: Eingabeschnittstellen der CarControlSoftware

Ausgabeschnittstellen - CarControlSoftware			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Digitales Fernsteuerungssignal	—	—	—
Kontrollbefehl	Feedback auf Prüfungs-anfrage d. Verbindung	byte	0x43

Tabelle 8.12: Ausgabeschnittstellen der CarControlSoftware

8.5 Subsystem 4: Rennstrecke

Das Subsystem Rennstrecke ist ein Subsystem, welches ausschließlich aus Hardwarekomponenten besteht. Über Reflexionsmarker soll die Rennstrecke ständig von der Positionsbestimmung erfasst werden können.
Es hat definierte Ein- und Ausgabeschnittstellen, welche im Folgenden beschrieben werden.

8.5.1 Eingabeschnittstellen

- **Infrarotlicht**
  Die Rennstrecke trägt Reflexionsmarker an den Streckenbegrenzungen, welche mit Infrarotlicht beleuchtet werden.

8.5.2 Ausgabeschnittstellen

- **Reflektiertes Infrarotlicht**
  Das Infrarotlicht, welches auf die Reflexionsmarker fällt, wird zur Lichtquelle zurück reflektiert. Dieses Verfahren stellt eine optische Verbindung zwischen der Positionsbestimmung und der Rennstrecke dar. Die Reflexionsmarker sind hierbei in einem festgelegten Muster angeordnet. Das Muster definiert die Rennstreckenbegrenzung bzw. die Maße der Rennstrecke und ermöglicht so die Bestimmung der Ausrichtung sowie die Erfassung einer ungewollten Verschiebung.

Eingabeschnittstellen - Subsystem Rennstrecke			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Infrarotlicht	Reflexionsmarker	—	—

Tabelle 8.13: Eingabeschnittstellen des Subsystems Rennstrecke

Ausgabeschnittstellen - Subsystem Rennstrecke			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Reflektiertes Infrarotlicht	Rennstreckenpose / -maße	—	—

Tabelle 8.14: Ausgabeschnittstellen des Subsystems Rennstrecke

8.6 Systemsteuerungssoftware

Die Systemsteuerungssoftware bietet dem Administrator ein Interface für die Steuerung und Überwachung der Subsysteme Positionsbestimmung und Control-Unit. Über dieses Benutzerinterface können außerdem bestimmte Parameter des Systembetriebs konfiguriert werden. Des Weiteren soll sie Fahrzeugposen, Fehlercodes und Zeitdaten der Subsysteme empfangen und einen Fehlerspeicher mit Fehlercodes beschreiben und auslesen können. Zusätzlich soll die Systemsteuerungssoftware den Administrator über den aktuellen Systemzustand informieren, eine Protokolldatei führen und einen Not-Aus sämtlicher Komponenten auslösen können. Die Systemsteuerungssoftware hat definierte Ein- und Ausgabeschnittstellen, welche im Folgenden beschrieben werden.

8.6.1 Benutzerinterface

Das Benutzerinterface der Systemsteuerungssoftware besitzt Buttons für das Wechseln der Systemzustände sowie Textfelder für die Ausgabe von Systeminformationen und die Eingabe von Konfigurationsparametern. Es werden Buttons für das Bestätigen der Inspektion, zum Starten des Rennbetriebs, zum Stoppen des Rennbetriebs, zum erneuten Durchführen einer Inspektion, zum Abschalten des Systems und für einen manuellen Not-Aus bereitgestellt.

Benutzerinterface - Systemsteuerungssoftware		
Beschreibung	**Datentyp**	**Wertebereich**
Textfeld - Erwartete Fahrzeuganzahl	int	0 . . . 16
Textbereich - Systeminformationen	String	—
Textbereich - Fehlermeldungen	String	—
Button - Inspektion bestätigen	—	—
Button - Rennbetrieb starten	—	—
Button - Rennbetrieb stoppen	—	—
Button - Wartung	—	—
Button - System beenden	—	—
Button - Manueller Not-Aus	—	—

Tabelle 8.15: Schnittstellen des Benutzerinterfaces der Systemsteuerungssoftware. Nicht aufgeführt sind die Textfelder sämtlicher Konfigurationsparameter. Diese können Tabelle 14.4 aus Anhang 14.6 entnommen werden.

Die Dateien des Fehlerspeichers und der Protokolle werden automatisch im Ausführungsverzeichnis der Systemsteuerungssoftware erzeugt. Daher werden keine Konfigurationsparameter für die jeweiligen Dateipfade benötigt.

8.6.2 Konfigurationsparameter

Vor der Initialisierung der Systemsteuerungssoftware müssen Konfigurationsparameter für den Betrieb festgelegt werden. Hierzu gehören neben der erwarteten Anzahl der am Rennen teilnehmenden Fahrzeuge, insbesondere Konfigurationsparameter für die Festlegung von bestimmten Toleranzen und die Vorgabe der einzuhaltenden Realzeitanforderungen. Sämtliche Parameter sind in Anhang 14.6 mit den entsprechenden Datentypen, dem Wertebereich und einer Beschreibung aufgelistet.

8.6.3 Eingabeschnittstellen

- **Fahrzeugdaten**
  Damit die Systemsteuerungssoftware die Positionsbestimmung überwachen kann und den aktuellen Standpunkt sämtlicher Fahrzeuge kennt, wird das Fahrzeugdatenpaket von der Positionsbestimmung empfangen. Die Systemsteuerung erhält von der Positionsbestimmung die X- und Y-Koordinaten sämtlicher erkannter Fahrzeuge als Double-Wert (double) im Bereich 0,00 bis X_{max} bzw. 0,00 bis Y_{max} dar, wobei X_{max} der Breite und Y_{max} der Länge der Rennstrecke in Zentimetern entspricht. Zusätzlich werden die Ausrichtungswinkel sämtlicher erkannter Fahrzeuge als Double-Wert (double), die Verfügbarkeit der Fahrzeuge und ein Zeitstempel vom Datentypen timeval für die Feststellung des Alters der Fahrzeugposen mitgeteilt.

- **Kontrollbefehl**
  Um der Systemsteuerungssoftware eine erfolgreiche Initialisierung der Control-Unit's und der Positionsbestimmung mitzuteilen, wird von der jeweiligen Komponente ein Kontrollbefehl versendet. Dieses ist als Integer-Wert (int) 1 kodiert.

- **Fehlercode**
  Damit die Systemsteuerungssoftware auf Fehlfunktionen des Systems reagieren kann, empfängt diese Fehlercodes der einzelnen Komponenten. Der Fehlercode wird als Integer-Wert (int) empfangen. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in Kapitelabschnitt 14.1 aufgelistet ist.

- **Zeitdaten**
  Um die Einhaltung der Realzeitanforderungen überprüfen zu können, wird der Zeits-

tempel, der an die CarControl gesendeten Regelungswerte von der Control-Unit empfangen. Der Zeitstempel ist vom Datentyp timeval und wird zusammen mit dem Zeitstempel der Fahrzeugpose über das Netzwerk empfangen.

- **Fehlerspeicher**
  Um einen korrekten und fehlerfreien Systembetrieb zu gewährleisten, wird der Fehlerspeicher ausgelesen, welcher Fehler aus dem vorherigen Systembetrieb enthalten kann. Der Fehlerspeicher enthält sämtliche Fehlercodes als Integer-Werte (int), welche in einer Textdatei abgelegt sind und in Kapitelabschnitt 14.1 aufgelistet sind. Der Fehlerspeicher muss vor dem Start des Systems leer sein und ggf. manuell vom Administrator im Vorfeld geleert werden.

8.6.4 Ausgabeschnittstellen

- **Kontrollbefehl**
  Für das Wechseln der Systemzustände sendet die Systemsteuerungssoftware einen Kontrollbefehl. Mittels Kontrollbefehls als Integer-Wert (int) kann die Systemsteuerungssoftware das Wechseln des Systemzustandes aller Komponenten verlangen. Mit dem Wert 0 ist der Systemzustand ”Standby” kodiert. Mit dem Wert 1 ist der Systemzustand ”Rennbetrieb” kodiert. Mit dem Wert 2 ist der Systemzustand ”Initialisierung” kodiert. Mit dem Wert 3 ist der Systemzustand ”Inspektion” kodiert. Mit dem Wert 4 ist der Systemzustand ”Aus” kodiert.

- **Fehlerspeicher**
  Um einen korrekten und fehlerfreien Systembetrieb zu gewährleisten wird der Fehlerspeicher in einem Fehlerfall von der Systemsteuerungssoftware beschrieben. Der Fehlerspeicher erhält sämtliche Fehlercodes als Integer-Werte (int), welche in einer Textdatei abgelegt sind und in Kapitelabschnitt 14.1 aufgelistet sind.

- **Protokoll**
  Um einen korrekten, fehlerfreien und nachvollziehbaren Systembetrieb zu gewährleisten kann eine Protokolldatei beschrieben werden. Sie enthält neben Betriebsparametern sämtliche Fehlercodes als Integer-Werte (int). Die Protokolldatei wird bei jedem Systemstart neu angelegt und mit der zur Systemstartzeit aktuellen Uhrzeit und dem Tagesdatum benannt. So wird sichergestellt, dass Protokolldateien nicht überschrieben werden.

- **Not-Aus Befehl**
  Damit sämtliche Fahrzeuge sofort die Längs- und Querführung auf die Neutralstellung bringen und die Fahrzeuge anhalten, wird ein Not-Aus Befehl gesendet. Der Not-Aus Befehl wird als Integer-Wert (int) 1 gesendet.

Eingabeschnittstellen - Systemsteuerungssoftware			
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1
	X-Koordinate	double[16]	0 ... 236
	Y-Koordinate	double[16]	0 ... 236
	Ausrichtung	double[16]	0 ... 360
	Zeitstempel der Fahrzeugpose	timeval	—
Kontrollbefehl	Initialisierung erfolgreich	int	1
Fehlercode	Fehlercode	int	error > 0
Zeitdaten	Zeitstempel der Fahrzeugpose	timeval	—
	Zeitstempel der gesendeten Regelungswerte	timeval	—
Fehlerspeicher	Fehlerspeicher	—	—

Tabelle 8.16: Eingabeschnittstellen der Systemsteuerungssoftware

Ausgabeschnittstellen - Systemsteuerungssoftware			
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**
Kontrollbefehl	Wechsel d. Systemzustände	int	0 ... 4
Fehlerspeicher	Fehlerspeicher	—	—
Protokoll	Protokolldatei	—	—
Not-Aus Befehl	Wechsel in den Fehlerzustand	int	1

Tabelle 8.17: Ausgabeschnittstellen der Systemsteuerungssoftware

9 Sicherheitskonzept

In diesem Kapitel wird ein Sicherheitskonzept für den Systemaufbau des Projektes *RC-CARS* aufgestellt. Es werden mögliche Fehlercodes und Schadenszenarien mit dem Ziel beschrieben ein definiertes Schutzniveau zu erreichen. Im Fokus steht das Fahrzeug als sicherheitskritisches Objekt, welches das einzige bewegte Objekt im System darstellt. Durch einen Fehlerfall kann das Fahrzeug ggf. nicht mehr kontrolliert werden, wobei die Ursache sowohl ein Komplettausfall als auch ein Fehlverhalten einzelner Komponenten sein kann. Im Rahmen des Sicherheitskonzepts werden für jedes Subsystem Schadenszenarien beschrieben und geprüft, wie in der jeweiligen Situation ein sicherer Systemzustand hergestellt werden kann indem das Fahrzeug sofort angehalten wird. Des Weiteren werden die Struktur der Fehlercodes erläutert (siehe Kapitelabschnitt 14.1) und die Realzeitbedingungen erläutert.

9.1 Fehlercodes

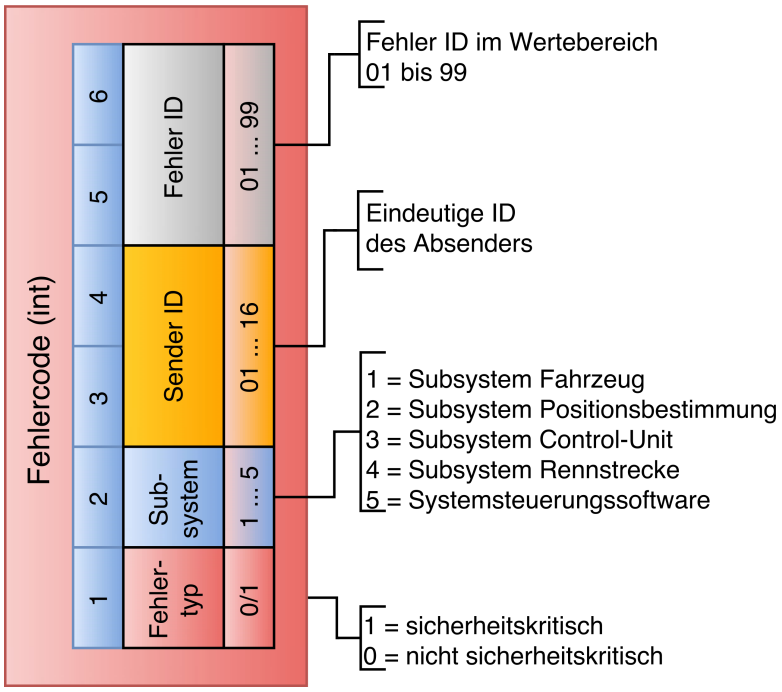


Abbildung 9.1: Codierung des Fehlercodes.

Teil des Sicherheitskonzepts ist der Einsatz von Fehlerdatenpaketen mit einem spezifischen Fehlercode, welche von den einzelnen Systemkomponenten erzeugt werden können. Die Fehlercodes setzen sich jeweils aus insgesamt sechs Ziffern zusammen. Die Kodierung der Fehlercodes ist in Abbildung 9.1 dargestellt. Jede Stelle hat dabei eine bestimmte Bedeutung. Dadurch wird erreicht, dass beim Auftreten eines Fehlers anhand des Fehlercodes nachvollzogen werden kann, um was für eine Art Fehler es sich handelt und wo dieser Fehler aufgetreten ist. Die erste Stelle des Fehlercodes kategorisiert, ob es sich um einen sicherheitskritischen (1) oder nicht-sicherheitskritischen (0) Fehler handelt. Die zweite Stelle beschreibt, in welchem Subsystem der Fehler aufgetreten ist. Die 1 steht für das Subsystem Fahrzeug, die 2 für das Subsystem Positionsbestimmung, die 3 für das Subsystem Control-Unit, die 4 für das Subsystem Rennstrecke und die 5 für das Subsystem Systemsteuerung. Hierbei ist zu beachten, dass die Subsysteme Fahrzeug und Rennstrecke zur Zeit keine Fehlercodes produzieren können. In zukünftigen Ausbaustufen kann sich dies jedoch ändern, wenn beispielsweise die Fahrzeuge eine neue Hardwareplattform erhalten und die Rennstrecke mit Lichtschranken oder ähnlicher Technik ausgestattet wird. Die dritte und vierte Stelle des Fehlercodes beschreibt die jeweilige ID des Senders, in dem der Fehler aufgetreten ist. Dies hilft dabei zu identifizieren, z.B. in welcher Instanz einer Control-Unit oder bei welchem Fahrzeug ein Fehler aufgetreten ist. Die fünfte und sechste Position des Fehlercodes beschreibt die Fehler ID, welche einen Wertebereich von 01 bis 99 besitzt. Passend zu dieser Zusammenstellung der Fehlercodes befindet sich im Anhang (siehe Kapitelabschnitt 14.1) eine aktuelle Fehlercodetabelle. Diese beinhaltet sämtliche Fehlercodes und deren Bedeutungen.

9.2 Not-Aus Befehl

Der Not-Aus Befehl kann nur von der Systemsteuerungssoftware ausgelöst bzw. versendet werden. Dieser kann automatisch auf Grundlage eines empfangenen sicherheitskritischen Fehlercodes oder manuell über einen Not-Aus Button ausgelöst werden. Wenn das System über den Not-Aus Befehl in den sicheren Fehlerzustand überführt wird, lösen sämtliche Control-Unit's einen Not-Halt derer Fahrzeuge aus.

9.3 Schadenszenarien

9.3.1 Fehler in der Kommunikation

Ausfall der Kommunikation

Wenn die gesamte Kommunikation zwischen der Systemsteuerung, der Positionsbestim-

mung und den Control-Unit's abbricht, können keine Netzwerkdatenpakete gesendet und empfangen werden. In diesem Fall kann das System über die Systemsteuerung nicht mehr beeinflusst bzw. die Fahrzeuge nicht mehr angehalten werden. Für diesen Fall besitzt jede Control-Unit einen Watchdog. Der Watchdog prüft mithilfe von Plausibilitätsüberprüfungen die empfangenen Netzwerkdatenpakete. So wird beispielsweise die Differenz zwischen den empfangenen Fahrzeugdatenpaketen stätig geprüft. Sollte der Empfang eines Fahrzeugdatenpakets länger als 20 ms andauern, wird ein Not-Halt des eigenen Fahrzeuges ausgelöst und zusätzlich ein Fehlercode verschickt.

Datenintegrität
Um sicherzustellen, dass der Inhalt eines Netzwerkdatenpakets bei der Übertragung nicht verändert wird, muss die Datenintegrität gewährleistet werden. Dazu wird von dem jeweiligen UDP-Socket zu jedem zu versendeten Netzwerkdatenpaket eine Prüfsumme berechnet und dem Netzwerkdatenpaket beigefügt. Anhand dieser Prüfsumme kann der UDP-Socket des Empfängers die Datenintegrität überprüfen und entsprechende Maßnahmen vornehmen.

9.3.2 Fehler der Positionsbestimmung

Fehler im Videostream
Ein Fehler im Videostream kann entstehen, wenn ein oder mehrere Fahrzeuge oder die Rennstrecke im Betrieb nicht mehr erkannt werden. Eine mögliche Ursache wäre z.B. ein Eingriff in das System, bei dem das Kameragestell bewegt worden ist. Die Positionsbestimmung ist daher in der Lage einen entsprechenden Fehlercode zu versenden und dies somit den anderen Systemkomponenten mitzuteilen. Die Control-Unit's lösen in diesem Fall direkt einen Not-Halt der eigenen Fahrzeuge und die Systemsteuerung einen Not-Aus des gesamten Systems aus.

Fehler in der Positionsbestimmung
Die Positionsbestimmung könnte durch einen Softwarefehler abstürzen oder sich in einem Zustand befinden, in welchem dasselbe Fahrzeugdatenpaket wiederholt ausgesendet wird. Der erste Fall wird durch den Watchdog der Control-Unit (siehe Kapitelabschnitt 9.3.1) und die Systemsteuerung detektiert. Für den Fall, dass die Positionsbestimmung wiederholt dieselben Netzwerkdatenpakete aussendet, kann dies über einen Zeitstempelvergleich der Datenpakete auf der Empfängerseite erkannt werden. Somit ist sichergestellt, dass wiederholt gleiche versendete Fahrzeugdaten mit dem gleichen Zeitstempel detektiert und ein Not-Aus bzw. Not-Halt ausgelöst wird. Auch erwartet die Control-Unit bei ausgesendeten Regelungswerten für das Fahrzeug eine Veränderung der Fahrzeugpo-

se, wodurch durch redundante Plausibilitätsüberprüfungen ein gewisses Maß an Sicherheit gewährleistet wird.

9.3.3 Fehler der Systemsteuerung

Die Systemsteuerung bzw. die Systemsteuerungssoftware ist als zentrale Steuerungs- und Überwachungskomponente unter anderem für das automatische Auslösen eines Not-Aus Befehls (bedingt durch einen sicherheitskritischen Fehlercode) und das manuelle Auslösen eines Not-Aus Befehls durch den Administrator zuständig. Im Fehlerfall der Systemsteuerung kann der Rennbetrieb nicht mehr beendet oder ein Not-Aus Befehl ausgelöst werden. Dadurch, dass die Positionsbestimmung und die Control-Unit's ebenfalls Fehlercodes senden können, kann das System in einem weiteren Fehlerfall ohne die Systemsteuerung automatisch in den sicheren Fehlerzustand überführt werden. In diesem Fall würden die Fahrzeuge durch den von der Positionsbestimmung oder einer der Control-Unit's erhaltenen sicherheitskritischen Fehlercodes einen Not-Halt ausführen.
Um der Positionsbestimmung und den Control-Unit's frühzeitig einen Ausfall der Systemsteuerung mitzuteilen, wird der Kontrollbefehl des aktuellen Systemzustandes als Keepalive-Befehl kontinuierlich in einem bestimmten Zeitabstand ausgesendet. Wenn dieser Kontrollbefehl nicht mehr übertragen wird, so ist dies der Indikator, dass die Systemsteuerung ausgefallen ist und ein Not-Halt der Fahrzeuge ausgelöst werden sollte.

9.3.4 Fehler der Control-Unit

Da sich die Control-Unit aus den Komponenten "Wrapper Code", "SCADE Modell" und "CarControl" zusammensetzt, ergeben sich hier drei dedizierte Fehlerquellen. Zum Ersten und Zweiten der Wrapper Code und das *SCADE* Modell welche auf dem Rechner der Control-Unit ausgeführt werden, zum Dritten die Software der CarControl. Alle Komponenten können den Dienst komplett einstellen oder sich in einem Zustand befinden, in welchem sich ein bestimmter Zustand wiederholt. Im Fehlerfall der ersten beiden Komponenten ist die CarControl dazu in der Lage zu erkennen, dass keine erwarteten Regelungswerte oder Kontrollbefehle gesendet werden. In diesem Fall führt sie einen Not-Halt des Fahrzeugs aus. Durch die auf der CarControl implementierte Software ist die CarControl ebenfalls in der Lage eine Plausibilitätsüberprüfung der Regelungswerte durchzuführen und im Falle von fehlerhaften Regelungswerten einen Not-Halt auszulösen. Im Fehlerfall der CarControl stellt sich ein Not-Halt des Fahrzeugs als schwierig dar. Die CarControl ist das einzige Verbindungsglied zum Fahrzeug. Wenn die CarControl den Dienst komplett einstellt, so wird im best-case der Fernsteuerung die Betriebsspannung entzogen und das Fahrzeug rollt aus. Dies ist ebenfalls durch das manuelle Trennen der Verbindung

zwischen dem Rechner und der CarControl möglich. Im worst-case hält die CarControl den aktuellen Regelungswert dauerhaft an und das Fahrzeug kollidiert schließlich mit der aktuellen Beschleunigung ungebremst mit der Rennstreckenbegrenzung. Durch den aktuellen Hardwareaufbau und den fehlenden Kommunikationsrückkanal zwischen dem Fahrzeug und der Control-Unit lässt sich dieser Fehlerfall nicht anderweitig lösen. Die Control-Unit wäre jedoch dazu in der Lage, über eine Plausibilitätsüberprüfung der aktuellen und einer erwarteten Fahrzeugpose einen sicherheitskritischen Fehlercode an die Systemsteuerungssoftware und alle weiteren Control-Unit's zu senden um somit einen Not-Halt aller übrigen Fahrzeuge auszulösen.

9.4 Realzeitbetrachtung

Eine schnelle und zuverlässige Kommunikation zwischen Sensoren und Aktoren ist essentiell für ein autonomes Fahrzeugsteuerungssystem. Im Fall von *RCCARS* besteht die Sensorik aus einer Kamera, welche Bilder der Fahrzeuge auf der Rennstrecke erfasst. Die Positionsbestimmung ermittelt die jeweilige Fahrzeugpose und übermittelt diese an die Control-Unit, welche die Regelungsparameter berechnet und diese mit Hilfe der CarControl an das Fahrzeug sendet. Dabei gelten strikte Realzeitanforderungen, welche im folgenden hergeleitet werden.
Die verwendete Kamera *Flea 3* [Kam] liefert Bilder mit einer maximalen Wiederholrate von 150 Hz. Laut [Ver14] führt eine Wiederholrate von 100 Hz zu einer guten Kombination von Präzision und Performanz in Bezug auf die Ermittlung der Fahrzeugposen und das regelmäßige Senden von Regelungswerten an das Fahrzeug. Basierend auf den Ergebnissen der Hardwareevaluation ist davon auszugehen das der Verlust von zwei Bildern in der Positionsbestimmung durch eine nur auf den Positionsdaten basierende Regelung nicht mehr auszugleichen ist. Daraus ergibt sich das auch die Control-Unit nicht länger als 10 ms für die Verarbeitung der Positionsdaten benötigen sollte, da andernfalls ein Bild der Positionsbestimmung verloren gehen kann. In dieser Zeitspanne legt das Fahrzeug unter Berücksichtigung der Mindestdurchschnittsgeschwindigkeit von 1,5 m/s bereits eine Wegstrecke von 1,5 cm zurück.
In Abbildung 9.2 ist das Zusammenspiel der einzelnen parallel ausgeführten Subsysteme und der Verlust eines Bildes im Zeitabschnitt bis 50 ms exemplarisch dargestellt. Von den Netzwerklatenzen bei der Datenübertragung kann an dieser Stelle aufgrund der Systemausführung auf einem einzigen Rechner abstrahiert werden.

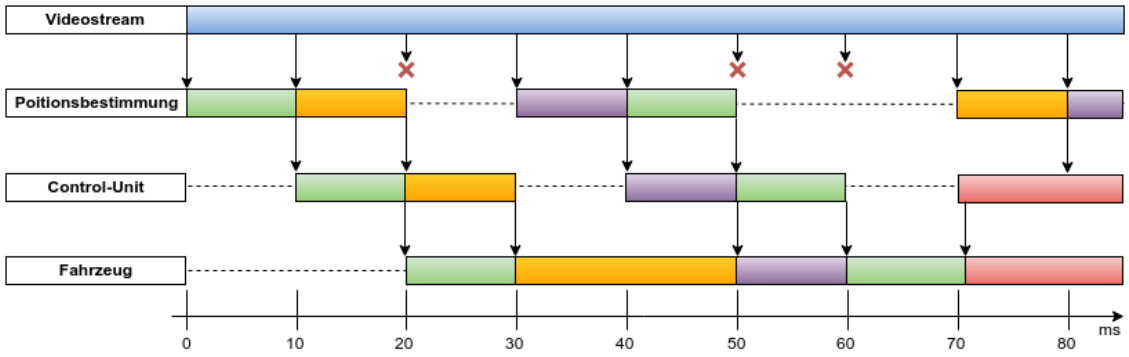


Abbildung 9.2: Visualisierung des Realzeitverhaltens des Systems

Der ab dem Zeitpunkt 50 ms skizzierte Verlust von zwei Bildern in der Positionsbestimmung wird durch das Sicherheitskonzept mittels Watchdog und Timestamps in der Control-Unit erkannt und führt automatisch zum Not-Halt des Fahrzeugs. Damit wird verhindert, dass das Fahrzeug länger als 20 ms ohne aktuelle Regelungswerte fährt. Ohne den Not-Halt durch die Control-Unit wäre die obere Schranke von 20 ms nicht zu garantieren. Durch dieses Konzept werden die in der Anforderungsdefinition festgehaltenen Realzeitanforderungen HW5.1.2, SW1.4, SW2.3 und SW4 erfüllt.

10 Softwareentwicklung

In diesem Kapitelabschnitt wird der aktuelle Stand der Softwareentwicklung und der Softwarearchitektur vorgestellt. Als Entwicklungsumgebungen wurden Eclipse Mars, Coocox und Scade benutzt.

10.1 Positionsbestimmung

Die Positionsbestimmungssoftware ist für die Erfassung der Fahrzeugpose auf der Rennstrecke und die Weiterleitung dieser an die Control-Units zuständig. Dafür muss die Software in der Lage sein Bilder von der Kamera zu empfangen und zu bearbeiten, um ‚beispielsweise Schmutzeffekte zu entfernen. Auf dem bearbeiteten Bild kann dann die Fahrzeugerkennung durchgeführt werden. Zudem muss die Software in der Lage sein, sowohl die Fahrzeugpose als auch Fehlermeldungen über das Netzwerk an die verschiedenen Control-Units und die Systemsteuerungssoftware zu versenden. Die gesamte Software wurde in C++ implementiert. Dies hängt einerseits mit der Kompatibilität zum C Code der Control-Unit und zum Anderen mit der openCV Bibliothek zusammen, welche für die Bildbearbeitung und Objekterkennung verwendet wurde und in C++ vorliegt.

10.1.1 Programmablauf

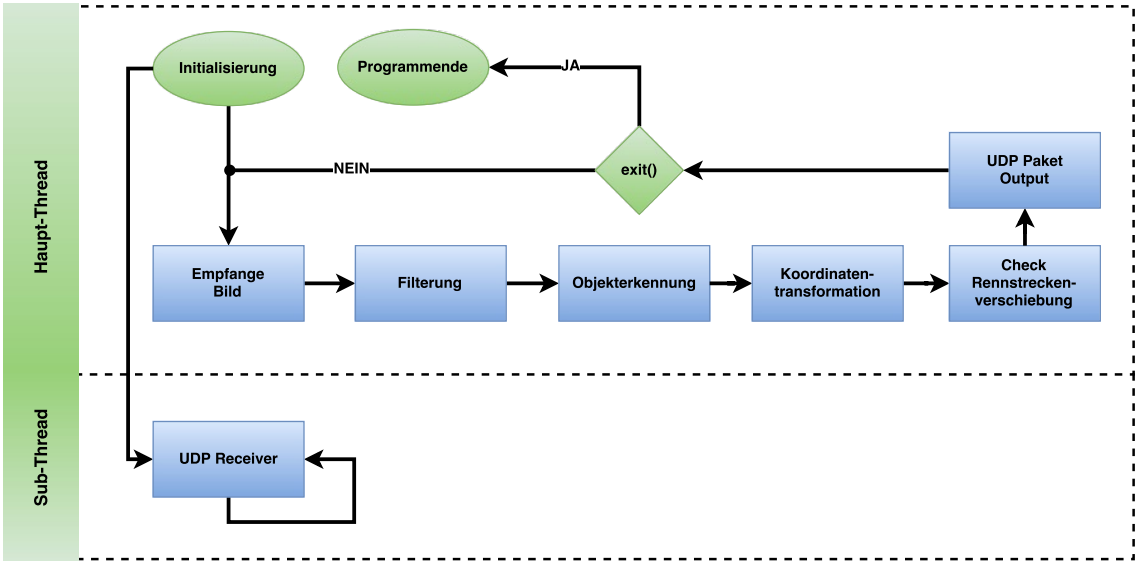


Abbildung 10.1: Vereinfachter Programmablauf der Positionsbestimmungssoftware

In Abbildung 10.1 ist der Ablauf der Positionsbestimmungssoftware in vereinfachter Form dargestellt. Das Programm beginnt mit einer Initialisierung, bei der alle notwendigen Objekte und Matrizen erzeugt werden. Zudem wird initial die Position der Rennstrecke und die sich auf ihr befindlichen Fahrzeuge bestimmt.

Nach der Initialisierung werden in einer Schleife Bilder von der Kamera empfangen. Jedes Bild wird umgewandelt und in ein Binärbild gefiltert, welches nur noch aus schwarzen und weißen Pixeln besteht. Auf diesem Binärbild wird im Anschluss die Objekterkennung durchgeführt, bei der die Pose der Fahrzeuge und die Position der Rennstrecke ermittelt werden. Im Anschluss werden die Fahrzeugkoordinaten in relative Rennstreckenkoordinaten umgerechnet. Bevor die ermittelten Daten dann als UDP Paket ins Netzwerk versendet werden, wird noch geprüft, ob sie die Rennstrecke bzw. das Gerüst seit dem letzten Bild bewegt haben.

Parallel zum Programmablauf wartet ein Thread auf Datenpakete von der Systemsteuerungssoftware und wertet diese aus.

10.1.2 Aufbau

Die Positionsbestimmungssoftware setzt sich aus 11 Klassen zusammen. Jede einzelne Klasse ist nur für ein spezifisches Teilproblem der Positionsbestimmung zuständig.

Durch diese klare Trennung der Zuständigkeiten ist das System zum Einen modular aufgebaut und zum Anderen auch erweiterbar. Für Funktionen, die zwar vorgesehen aber noch nicht implementiert wurden, sind bereits Platzhalter innerhalb der entsprechenden Klassen reserviert worden. Das folgende Klassendiagramm 10.2 verdeutlicht, welche Klasse für welche Teilaufgabe zuständig ist und wie die einzelnen Klassen zusammen hängen. Im Rahmen der Entwicklung sind diverse Prototypen der Positionsbestimmung entstanden. Dabei wurden Methoden und Funktionen entwickelt, die in der aktuellsten Reversion nicht mehr verwendet werden. Um das Klassendiagramm übersichtlicher zu gestalten, werden daher nur die für den Systembetrieb relevanten Methoden und Funktionen aufgelistet. Übergabeparameter werden aus dem selben Grund nicht aufgeführt.

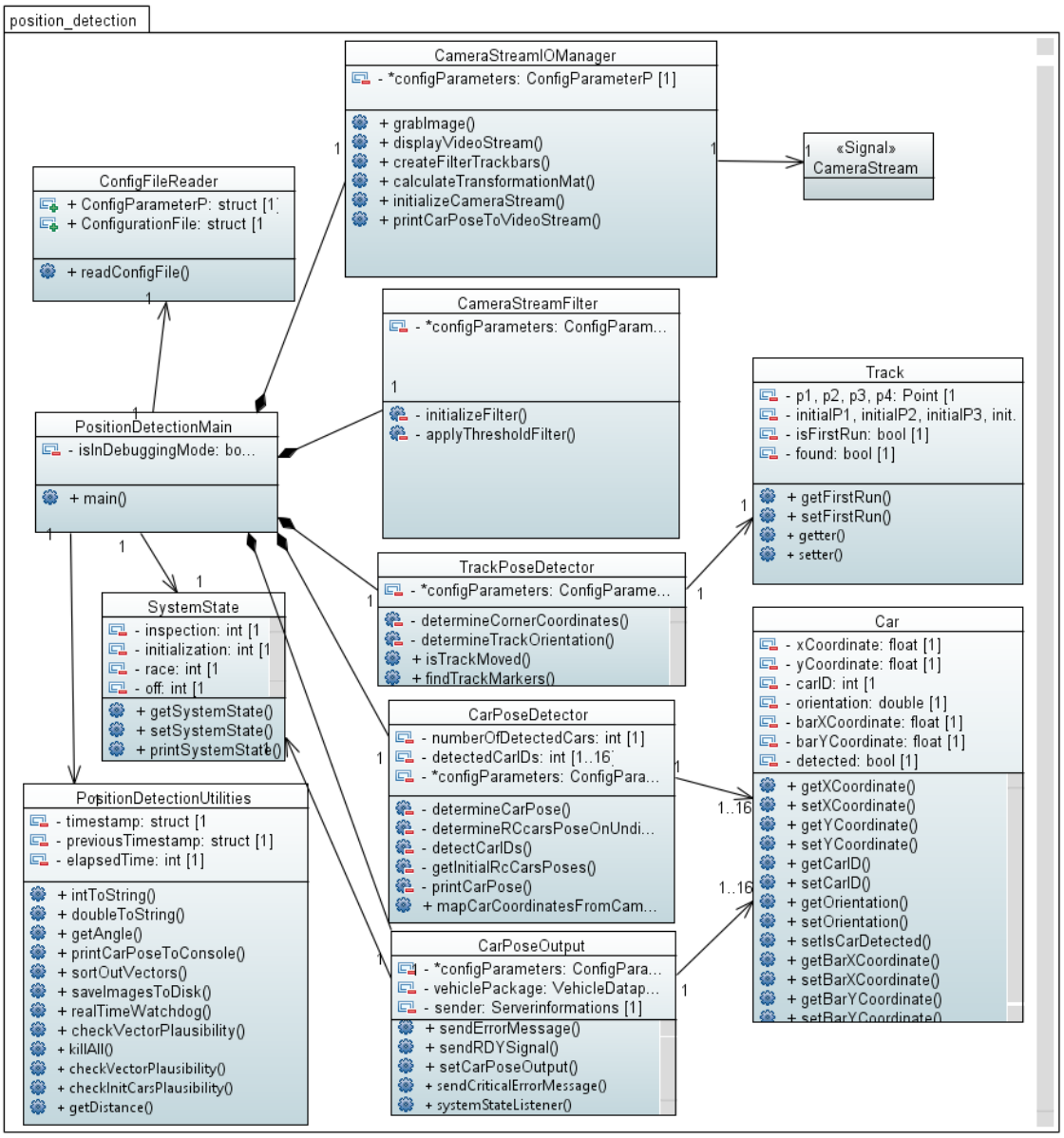


Abbildung 10.2: Ausschnitt aus dem Quellcode des CameraStreamIOmanagers.

In der Klasse *PositionDetectionMain* werden alle Objekte erzeugt, die für den Programmablauf notwendig sind. Die Bilderfassung ist in der Klasse *CameraStreamIOManager* implementiert, die Bildbearbeitung in der Klasse *CameraStreamFilter*. Die Objekterkennung ist in den Klassen *TrackPose-* bzw. *CarPoseDetector* realisiert und die Ausgabe der Daten über Netzwerk erfolgt in der Klasse *CarPoseOut*. Objekte der Klasse *Car* repräsentieren ein Fahrzeug auf der Rennstrecke, welche selbst durch die *Track* Klasse dargestellt wird. Die unterschiedlichen Systemzustände der Positionsbestimmung sind in der Klasse *Systemstate* realisiert. Für nähere Informationen zu den einzelnen Klassen, Methoden und Funktionen sei an dieser Stelle auf das generierte Doxygen Dokument verwiesen.

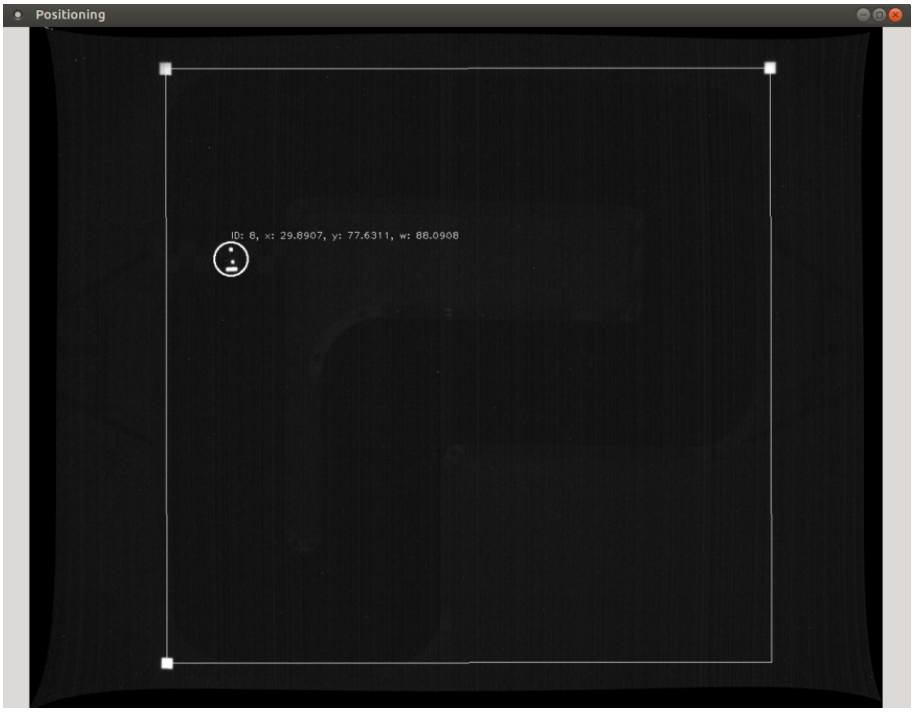


Abbildung 10.3: Entzerrtes Kamerabild nach Objekterkennung.

Um dem Administrator einen Überblick zu verschaffen, wird optional die Position des Fahrzeugs durch einen Kreis im vollständig entzerrten Videostream gekennzeichnet (siehe Abbildung 10.3). Rechts oberhalb des detektierten Fahrzeugs stehen die Fahrzeugidentität, Koordinaten und der Ausrichtungswinkel. Die erkannte Rennstrecke wird durch ein weißes Quadrat hervorgehoben. Im Folgenden wird auf Prozesse innerhalb der Positionsbestimmungssoftware, ausgewählte Klassen, deren Funktionsumfang und Ausgaben eingegangen.

10.1.3 Initialisierung

Die Initialisierung wird in der *Positiondetection Main* ausgeführt. Beim Starten der Software wird die Konfigurationsdatei ausgelesen, entsprechende Definitionen übernommen und Objekte erzeugt.

Außerdem wird überprüft, ob eine Verbindung zur Kamera hergestellt werden kann. Ist diese Überprüfung erfolgreich, wird ebenfalls geprüft, ob mindestens 100 Bilder pro Sekunde von der Kamera abgerufen werden können. Nachdem sichergestellt ist, dass die Kamera das System mit einer ausreichenden Anzahl an Bildern versorgen kann, wird schließlich ein erstes Kamerabild empfangen und verarbeitet. Im Rahmen der Verarbeitung wird das Bild zunächst entzerrt und gefiltert. Im gefilterten Bild werden im Anschluss

geschlossene Konturen gesucht und nach ihrer Größe sortiert. Die größten Konturen stellen die Rennstreckenmarker an den Rändern der Rennstrecke dar. Mit Hilfe dieser Marker wird die Lage der Rennstrecke festgestellt und entsprechend der Position dieser eine Transformationsmatrix errechnet, die es im späteren Systembetrieb erlaubt, Fahrzeugpositionen von Kamerakoordinaten in relative Rennstreckenkoordinanten umzurechnen. Zudem wird an dieser Stelle auch festgestellt, ob genau drei Rennstreckenmarker an den erwarteten Stellen im Kamerabild gefunden wurden. Falls dies nicht der Fall ist, schlägt die Initialisierung fehl, da angenommen wird, dass entweder nicht die gesamte Rennstrecke sichtbar ist oder sich gar keine Rennstrecke unter dem Kameragerüst befindet.

Zusätzlich wird bei der Initialisierung die initiale Pose der Fahrzeuge auf der Rennstrecke bestimmt. Dabei wird für jeden gefunden Fahrzeugbalkenmarker, welcher sich an der Fahrzeugfront befindet, ein Fahrzeugmarker gesucht, der das Heck des Fahrzeugs darstellt. Hierbei ist sowohl der Abstand der beiden Marker als auch der eingeschlossene Winkel entscheidend. Abhängig von der Anzahl der gefunden Fahrzeuge, werden *Car*-Objekte erzeugt, die die Fahrzeuge innerhalb der Positionsbestimmung repräsentieren. Zudem wird jedem gefunden Fahrzeug eine eindeutige ID zugewiesen. Diese wird aus der Konstellation der Marker auf der Fahrzeugkarosserie bestimmt. Stimmt die Anzahl der initialisierten Fahrzeugen jedoch nicht mit der Anzahl der erkannten Fahrzeugbalkenmarker überein, so ist bei der Feststellung der initialen Posen ein Fehler aufgetreten und das Programm beendet sich an dieser Stelle mit einer entsprechenden Fehlermeldung.

Sollte die Initialisierung fehlerfrei durchgeführt werden, kann die Positionsbestimmung beginnen Fahrzeugdaten an die anderen Subsysteme über das Netzwerk zu versenden.

10.1.4 Bilderfassung

Die Bilderfassung wird in der Klasse *CameraStreamIOManager* durchgeführt. Der Funktionsumfang beginnt mit dem Verbinden der Kamera, ermöglicht das Einlesen des Kamerastreams, sowie das Ausgeben von Bildern und Infotexten in Fenstern. Darüber hinaus werden hier auch Schieberegler erzeugt, die zur Bildmanipulation verwendet werden können. Zur direkten Ansteuerung der Kamera stellt der Hersteller das *FlyCapture SDK* zur Verfügung. Die API ermöglicht u.a. das Prüfen der Kameraverbindung und das Auslesen von Kameraparametern. Dazu gehören u.a. die aktuelle Auflösung oder die Firmwareversion.

Um das aktuellste Bild der Kamera zu erhalten, wird die Funktion `RetrieveBuffer()` ausgeführt, welche ein *Flycapture2 Image* zurück gibt. Dieses Bild wird in eine Matrix

umgewandelt.
Diese Umwandlung ist notwendig, da alle verwendeten *openCV* Methoden die Bilddaten in Matrixform verarbeiten. Im Fall von `grabImage()` besteht die Matrix zunächst aus 1280 Zeilen und 1024 Spalten, was der Kameraauflösung entspricht. In jedem Matrizeneintrag befindet sich ein Wert zwischen 0 und 255. Mit dem Wert 0 wird ein schwarzer und mit dem Wert 255 ein weißer Pixel repräsentiert. Alle anderen Werte repräsentieren Graustufen zwischen Schwarz und Weiß. Dieses initiale Graustufenbild muss jedoch noch bearbeitet werden, da die Bilder der Kamera teilweise Schmutzeffekte wie Streifen oder Rauschen aufweisen (vergleiche Abbildung 10.4). Erst nach einer Aufbereitung können die Objekterkennungsalgorithmen angewendet werden.

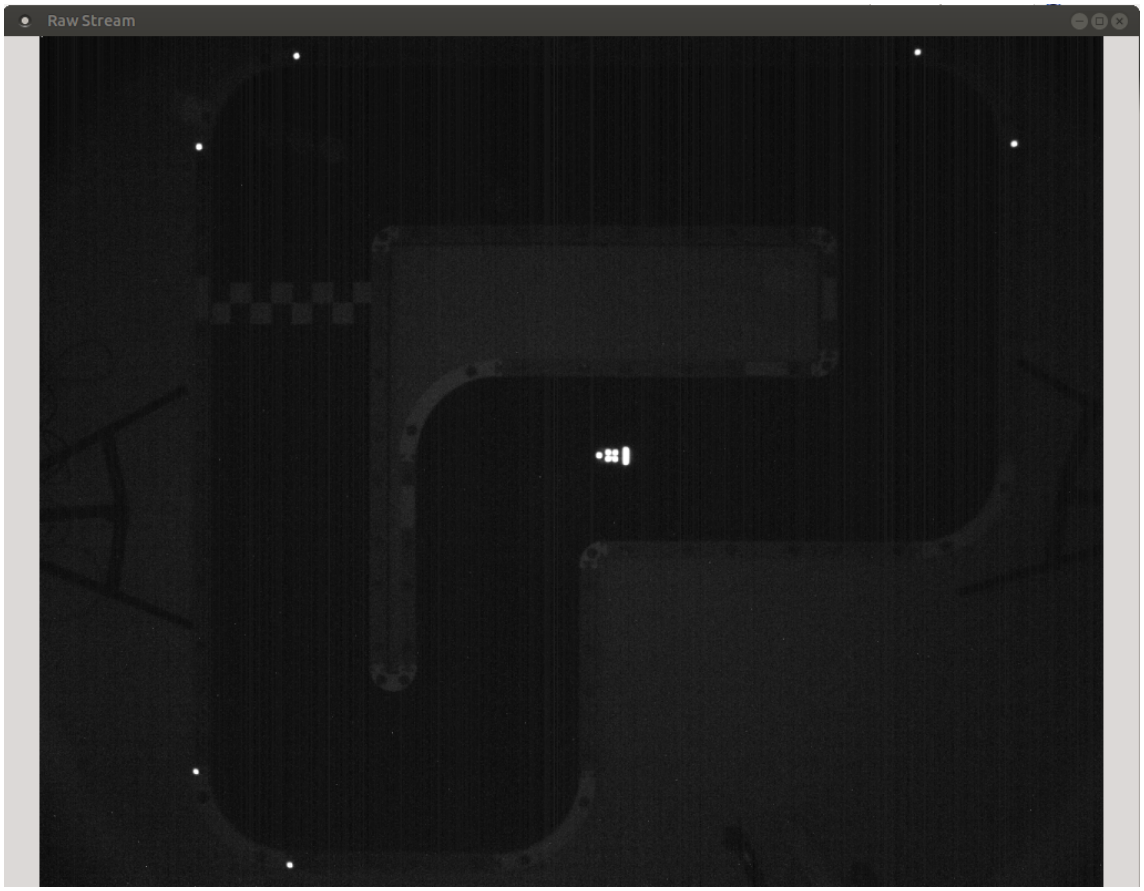


Abbildung 10.4: Unbearbeitetes Kamerabild und verzerrtes der CamStreamIOManager Klasse.

Obwohl es in Abbildung 10.4 kaum erkennbar ist, ist jedes Bild der Kamera bedingt durch das Weitwinkelobjektiv verzerrt. Diese Verzerrung wird jedoch im Rahmen der Objekterkennung korrigiert.

10.1.5 Bildbearbeitung

Die Bildbearbeitung wird ausschließlich von der Klasse *CameraStreamFilter* durchgeführt. In dieser Klasse wird die Matrix mit den Bildinformationen so bearbeitet, dass am Ende ein Binärbild entsteht. Dies wird durch ein drei dreistufiges Verfahren erreicht.

Im ersten Schritt kann eins von insgesamt fünf verschiedenen Schwellwertverfahren gewählt werden. Standardmäßig wird das *Truncate* Verfahren verwendet. Dabei wird für jeden Pixel im zu bearbeitenden Bild src(x, y) ein festgelegter Schwellwert zwischen 0 und 255 gesetzt, sofern gilt, dass src(x,y) größer als der Schwellwert ist. Andernfalls bleibt der Pixelwert unverändert im bearbeiten Bild.

Im zweiten Schritt wird das verarbeitete Bild mit dem *MedianBlur*-Verfahren verwischt. Dabei wird jeder Pixel im Bild mit dem Mittelwert der Helligkeit der Nachbarpixel überschrieben. Dies ermöglicht einzelne unerwünschte Pixel aus dem Bild zu filtern. Damit lässt sich das Rauschen, wie es in Abb. 10.4 sichtbar ist vollständig entfernen.

Im letzten Schritt wird mittels des *inRange-Threshold*-Verfahrens ein Binärbild erstellt. Bei diesem Verfahren wird eine untere und obere Helligkeitsgrenze für Pixel durch den Administrator vorgegeben. Liegt ein Pixel innerhalb dieser Grenze, so wird der Wert dieses Pixels auf 255 gesetzt. Andernfalls wird dem Pixel der Wert 0 zugeordnet.

Im Rahmen der Systementwicklung wurden diverse Verfahren evaluiert. Es hat sich herausgestellt, dass eine Kombination dieser drei Verfahren zufriedenstellende Ergebnisse hervorbringt. An dieser Stelle soll jedoch angemerkt werden, dass das Finden passender Filterparameter ein aufwändiger Prozess ist. Daher sollten die Standardparameter nur minimal den Lichtverhältnissen angepasst werden.

10.1.6 Objekterkennung

Für die Objekterkennung werden die Klassen *Trackpose-* und *Carposedetector* verwendet. Die Fahrzeug- und Rennstreckenmarker werden aus Gründen der Laufzeitoptimierung in nur einem einzigen Schritt im gesamten Bild gesucht und anschließend von verschiedenen Methoden der *Trackpose-* bzw. *Carposedetector*-Klasse ausgewertet.

10.1.6.1 Fahrzeugerkennung

Die Methode `determineRCcarsPoseOnUndistortedTrack()` sucht in einem Binärbild mit Hilfe der openCV Methode `findContours()` nach Konturen und speichert diese in

einem Vektor, welcher wiederum einen Vektor enthält, der die gefunden Pixel, die sich zu Konturen ergänzen, enthält. Anschließend werden die Konturen in Abhängigkeit von ihren einschließenden Flächen in vier Kategorien sortiert. Konturen die eine Fläche zwischen 0 und 5 Pixeln einschließen, sind zu klein, werden daher als Rauschen interpretiert und nicht beachtet. Die nächstgrößeren Konturen sind die Fahrzeugmarker mit einer Größe von 1cm · 1cm. Mittels der gewichteten Mittelwerten der Helligkeitswerten können anschließend die Koordinaten der Konturen festgestellt und in einem Vektor gespeichert werden. Selbiges geschieht mit den Balkenmarkern an der Fahrzeugfront und Rennstreckenmarkern, welche jedoch im Rahmen der Fahrzeugerkennung vorerst nicht weiter betrachtet werden. Da sämtliche Marker zwar auf einem bearbeiteten, jedoch verzerrten Bild gefunden wurden, werden im Anschluss die Koordinaten mittels der *openCV* Methode `undistortPoints()` entzerrt. In älteren Prototypen wurde noch jedes Bild vor der Bearbeitung vollständig entzerrt. Dies hat sich negativ auf die Laufzeit ausgewirkt und wurde verworfen, um nicht die Realzeitanforderungen zu verletzen.

Bevor die Fahrzeugerkennung durchgeführt werden kann, sind die Vektorinhalte auf ihre Plausibilität zu überprüfen. So darf beispielsweise kein Vektor leer sein, da dies bedeutet, dass keine Kontur mit der erwarteten Größe gefunden wurde. Sind die Vektorinhalte jedoch plausibel, kann im nächsten Schritt die Fahrzeugpose bestimmt werden. Obwohl alle 16 Fahrzeuge initialisiert werden können, ist der aktuelle Prototyp nur in der Lage, die Pose eines einzelnen Fahrzeugs im Rennbetrieb zu bestimmen. Falls sich mehr als ein Fahrzeug auf der Rennstrecke befindet, gibt die Positionsbestimmung einen entsprechenden Vermerk aus.

Um die Pose eines einzelnen Fahrzeugs zu bestimmen, wird für den gefundenen Balkenmarker im Radius von 25 bis 34 Pixeln ein Fahrzeugmarker gesucht. Dieser Abstand entspricht dem Abstand zum Heck des Fahrzeugs. Ist ein solcher Marker gefunden, so gilt auch das Fahrzeug als erkannt. Die Position entspricht dann der Mitte zwischen Balken- und Endmarker. Eine erneute Identitätsüberprüfung ist nicht notwendig, da nur ein einziges Fahrzeug initialisiert ist und sich die ID nicht ändern kann. Um den Ausrichtungswinkel festzustellen, wird die Hilfsfunktion `getAngle()` aus der *PositiondetectionUtilities* Klasse aufgerufen. Mittels der `atan2` Funktion wird der Winkel zwischen Balken und Endmarker bestimmt und anschließen von Radiant und Grad umgerechnet. Da der Arkustangens nur auf [-180 °, 180 °] abbildet, muss der Winkel entsprechend angepasst werden um Ausrichtungswinkel zwischen 0° und 360° zu erhalten.

Im letzten Schritt werden die erfassten Fahrzeugdaten in ein *Car* Objekt geschrieben und

die Objekterkennung ist für das Fahrzeug abgeschlossen.
Spätere Prototypen sollen in der Lage sein mehrere Fahrzeuge im Rennbetrieb zu erkennen. Aus diesem Grund ist in der `determineRCcarsPoseOnUndistortedTrack` Methode bereits ein entsprechender Platz vorgesehen.

10.1.6.2 Identitätserkennung

Damit das System verschiedene Fahrzeuge von einander unterscheiden kann, wurden Identitäten mittels Marker auf dem Dach der Fahrzeuge codiert. Mit bis zu vier Markern können 16 verschiedenen Fahrzeuge erkannt werden. Die Methode `detectCarIDs()` berechnet aus der Position dieser Identitätsmarker die Identität des entsprechenden Fahrzeugs.

Für jedes gefundene Fahrzeug werden in einem Radius von 10 Pixeln um den Fahrzeugmittelpunkt Marker gesucht und in einem Vektor gespeichert. Das Verfahren wertet im Anschluss den Inhalt des Vektors aus, indem vorerst nur die Anzahl der Elemente im Vektor betrachtet werden. Ist der Vektor leer, so wurden keine Identitätsmarker gefunden und somit erhält das Fahrzeug die ID 0. Für den Fall, dass vier Marker im Vektor stehen, muss das Fahrzeug die Identität 15.

Sollte sich nur ein einziger Marker im Vektor befinden, kann das Fahrzeug die Identität 1, 2, 4 oder 8 besitzen. Anhängig vom Winkel zwischen End- und Identitätsmarker und Abstand zwischen Fahrzeugbalken und Identitätsmarker wird die Identität eindeutig bestimmt. Der Algorithmus für genau einen Marker arbeitet wie folgt: Falls der Abstand zwischen Identitätsmarker und Balken kleiner ist als der Abstand zwischen Identitätsmarker und Endmarker, ist die ID entweder 4 oder 8, sonst 1 oder 2. Prüfe im Anschluss, ob der Identitätsmarker links oder rechts von der Fahrzeugmitte liegt. Berechne dazu den Winkel zwischen Fahrzeugende und ID-Marker. Ist der Winkel größer als der Ausrichtungswinkel des Fahrzeugs, so muss der Marker rechts vom Mittelpunkt liegen. Damit hat das Fahrzeug die Identität 4. Andernfalls bleibt nur noch die ID 8 übrig. Äquivalent dazu kann zwischen ID 1 und 2 unterschieden werden.

Die Feststellung der ID für zwei und drei Marker ist im Prinzip analog. Es werden Äquivalenzklassen gebildet und Abstände und Winkel berechnet, um Fälle genauer eingrenzen zu können.

10.1.6.3 Rennstreckenerkennung

Die Rennstreckenmarker werden innerhalb der Methode `determineRCcarsPoseOnUndistortedTrack()` mit den restlichen Fahrzeugmarkern gefunden und in einem Vektor gespeichert. Die Auswertung der Vektorinhalte erfolgt mittels der `findTrackMarker()` Methode der Klasse *TrackPoseDetector*. Im Radius von 300 Pixeln um den Ursprung des Koordinatensystems wird der erste Rennstreckenmarker gesucht. Die anderen beiden Rennstreckenmarker werden in den jeweiligen Ecken der Rennstrecke gesucht. Da kein vierter Marker existiert wird die Position für diesen Marker aus der Position der anderen Markern berechnet.

Dabei fungieren die Koordinaten der Rennstreckenecke P2 als Stützvektor und der Vektor von Rennstreckenecke P1 und Rennstreckenecke P2 als Richtungsvektor. Sollten die Rennstreckenmarker zum ersten Mal ausgewertet werden, werden ebenfalls die initialen Rennstreckenecken gesetzt und im Anschluss der boolsche Rennstreckenparameter `firstRun` auf falsch gesetzt.

Dies ermöglicht der Funktion `isTrackMoved()` festzustellen, ob die Rennstrecke oder das Gerüst nach der Initialisierung bewegt wurden. Aufgrund von kleinen Messfehlern ist eine Abweichung der Markerpositionen von insgesamt drei Pixeln erlaubt.

10.1.7 Netzwerkpaket

Die Positionsbestimmungssoftware ist in der Lage drei verschiedene Typen von Netzwerkpaketen per UDP Broadcast zu versenden und Kontrolldatenpakete von der Systemsteuerungssoftware zu empfangen und auszuwerten.

10.1.7.1 Fahrzeugposenpaket

In jedem Durchlauf sendet die Software ein Fahrzeugposenpaket in das Netzwerk; unabhängig davon ob die Pose des Fahrzeugs gefunden werden konnte. Es enthält Informationen über das Alter des ausgewerteten Bildes und den Status des Fahrzeugs. Außerdem werden die Koordinaten und der Ausrichtungswinkel übermittelt (siehe Abb. 10.5).

```
---------- MESSAGE ----------
Identifier: 1
Timestamp: 1478660140 680970
Message-Sender: 200
Received At: 0 0

Vehicle         VehicleCheck    X-Koords              Y-Koords              Alignments
#0              0               0.000000              0.000000              0.000000
#1              0               0.000000              0.000000              0.000000
#2              0               0.000000              0.000000              0.000000
#3              0               0.000000              0.000000              0.000000
#4              0               0.000000              0.000000              0.000000
#5              0               0.000000              0.000000              0.000000
#6              0               0.000000              0.000000              0.000000
#7              0               0.000000              0.000000              0.000000
#8              0               0.000000              0.000000              0.000000
#9              0               0.000000              0.000000              0.000000
#10             0               0.000000              0.000000              0.000000
#11             0               0.000000              0.000000              0.000000
#12             0               0.000000              0.000000              0.000000
#13             0               0.000000              0.000000              0.000000
#14             0               0.000000              0.000000              0.000000
#15             1               29.121769             74.288567             88.152390
-----------------------------
```

Abbildung 10.5: Versendetes Fahrzeugdatenpaket.

10.1.7.2 Fehlerdatenpaket

Diverse interne Fehler der Positionsbestimmungssoftware können der Systemsteuerungssoftware mitgeteilt werden. So kann die Watchdog Methode die anderen Subsysteme informieren, dass die Realzeitanforderungen nicht eingehalten wurde oder die Fahrzeugpose im letzten Bild nicht bestimmt werden konnte.

10.1.7.3 Kritisches Fehlerdatenpaket

Die Software ist ebenfalls in der Lage kritische Fehlerdatenpakete zu versenden. Dies ist der Fall, wenn die Positionsbestimmung beispielsweise Probleme mit der Kamera feststellt oder sich beendet. Das Versenden eines solchen Paketes hat zur Folge, dass die Systemsteuerungssoftware einen systemweiten Nothalt auslöst.

10.1.7.4 Kontrolldatenpaket

Die Systemsteuerungssoftware ist in der Lage die Software der anderen Subsysteme fern zusteuern. Diesbezüglich wird bei der Initialisierung ein Thread erstellt, welcher im Hintergrund nach Datenpaketen im Netzwerk lauscht und diese bei der Ankunft auswertet. Zunächst wird überprüft, ob das Paket von der Systemsteuerung versendet wurde. Sofern ein solches Paket vorliegt und es sich dabei um ein Kontrolldatenpaket handelt, passt die Positionsbestimmungssoftware ihren Systemzustand an. Enthält das Paket beispielsweise das Kontrollsignal 4, so wird die Positionsbestimmung beendet.

10.1.8 Kamerakalibrierung

Für eine effektive Objekterkennung ist es notwendig eine Kamerakalibrierung vorzunehmen, um die Fahrzeugkoordinanten in einem orthogonalen und nicht verzerrten Koordinatensystem bestimmen zu können.

Diesbezüglich wird zwischen intrinsischen und extrinsischen Kameraparametern unterschieden. Während die intrinsischen Parameter die Verzerrung und den Versatz des Bildes beschreiben, kann mittels der extrinsischen Parametern die Transition zwischen Kamera- und Rennstreckenkoordinatensystem wiedergegeben werden. Im Rahmen der Kalibrierung werden diese Parameter ermittelt. Die Bestimmung der extrinsischen Parameter ist äquivalent zum Verfahren der Universität Uppsala [CAR14].

10.1.9 Intrinsische Kameraparameter

Innerhalb der intrinsischen Parametern wird zwischen Variablen unterschieden, welche die radiale und die tangiale Verzerrung des Bildes beschreiben. Durch die radiale Verzerrung erscheinen beispielsweise gerade Linien im Bild gebogen. Die tangiale Verzerrung hingegen verschiebt die radiale Verzerrung um den Mittelpunkt. Beide Phänomene treten wegen des verwendeten Weitwinkelobjektives auf und müssen softwareseitig korrigiert werden.

Dazu wird mit ein Schachbrettmuster verwendet, bei dem die Anzahl der Kacheln und deren Größe bekannt ist. Die *openCV* Funktionen `findChessboardCorners()` und `calibrateCamera()` berechnen aus den Ecken der erkannten Schachbrettkacheln die Verzerrung des Bildes und speichern die intrinsischen Parameter und die Kameramatrix in eine externe Datei. Die in dieser Datei befindlichen Parameter ermöglichen es dann durch den Methodenaufruf `undistortPoints` einzelne verzerrte Koordinaten zu entzerren.

10.1.10 Extrinsische Kameraparameter

Mit den bekannten intrinsischen Parametern, können zunächst die Rennstreckenmarker auf einem entzerrten Bild bestimmt werden. Anschließend kann mit den drei erkannten Eckmarkern der Rennstrecke, die die Achse des lokalen Koordinatensystems repräsentieren wiederum eine Transformationsmatrix berechnet werden. Diese Matrix ermöglicht es die zweidimensionalen Kamerakoordinanten in zweidimensionale Rennstreckenkoordinanten umzuwandeln. Dafür sind mehrere lineare Abbildungen notwendig.

Es wird zunächst angenommen, dass die Rennstrecke verschoben, verdreht und durch die nicht horizontale Lage zwischen Fotosensor der Kamera und Rennstrecke versetzt erscheint. Um diese drei Parameter berücksichtigen zu können, wird eine 3 x 3 Transformationsmatrix $\mathbb{M}$ genötigt, die eben diese Lage der Rennstrecke im Kamerabild beschreibt. Da die Kamerakoordinaten nur zweidimensional sind, müssen diese zunächst vom $\mathbb{R}^2$ ins $\mathbb{R}^3$ projiziert werden. Nun kann der dreidimensionale Koordinatenvektor mit der Transformationsmatrix multipliziert werden und im Anschluss wieder zurück ins $\mathbb{R}^2$ abgebildet werden.

Um $\mathbb{M}$ bestimmten zu können, müssen die vier Ecken der Rennstrecke (x_1,y_1) bis (x_4,y_4) vom $\mathbb{R}^2$ ins $\mathbb{R}^3$ gehoben werden, um im Anschluss eine Transformationsmatrix zu erhalten, die von orthogonalen Basisvektoren auf Kamerakoordinaten abbildet (siehe 10.1).

$$\mathbb{M}_{cam} = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} \left(\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_4 \\ y_4 \\ 1 \end{pmatrix} \right) \tag{10.1}$$

Somit muss eine lineare Abbildung mit der inversen Matrix $\mathbb{M}_{cam}^{-1}$ als Transformationsmatrix von Kamerakoordinaten auf orthogonale Basisvektoren abbilden. Nun wird die Matrix $\mathbb{M}_r$ benötigt, die von diesen Basisvektoren auf Rennstreckenkoordinaten abbildet. Diese Matrix wird analog zu $\mathbb{M}_{cam}$ bestimmt. Damit ist auch die gewünschte Transformationmatrix $\mathbb{M}$ vollständig beschrieben, die als Produkt von $\mathbb{M}_r$ und $\mathbb{M}_{cam}^{-1}$ berechnet wird. Mittels einer letzten linearen Abbildung können Kamerakoordinanten in Rennstreckenkoordinanten transformiert werden. Es gilt:

$$\begin{pmatrix} x_r \\ y_r \\ w_r \end{pmatrix} = \mathbb{M} \cdot \begin{pmatrix} x_{cam} \\ y_{cam} \\ w_{cam} \end{pmatrix} \tag{10.2}$$

Die dreidimensionalen Koordinaten können wie folgt in zweidimensionale Koordinaten umgewandelt werden:

$$\begin{pmatrix} x_r \\ y_r \\ w_r \end{pmatrix} \mapsto \begin{pmatrix} \frac{x_r}{w_r} \\ \frac{y_r}{w_r} \end{pmatrix} \tag{10.3}$$

10.1.11 Laufzeit

Die Laufzeit der Positionsbestimmung ist in der aktuellen Version des Prototypen mit etwa 4 ms vom Empfangen des Bilder bis zum Versenden des Datenpaketes schnell genug um jedes Bild bei einer Framerate von 150 fps empfangen und auswerten zu können. Im Vergleich zum vorigen Prototypen der etwa 70 ms für den selben Durchlauf benötigt hatte, ist dies eine deutliche Verbesserung.

Im Rahmen der Optimierung wurden zunächst alle verwendeten Methoden mit Zeitstempeln versehen um festzustellen, an welchen Stellen die Software optimiert werden muss. Dabei stellte sich heraus, dass das Drehen jedes ein kommenden Bildes sehr zeitintensiv war. Außerdem wurden verschiedene Verfahren zur Objekterkennung implementiert und getestet. Während in älteren Iterationen die Objekterkennung auf drei verschiedenen Matrizen ausgeführt wurde, werden nun sämtliche Marker in einem einzigen Bild ausgewertet.

Außerdem wurde an vielen Stellen auf Pointer umgestellt und redundanter Code entfernt.

10.2 Kommunikation

Wie in Kapitelabschnitt 4.5 beschrieben, soll die Kommunikation zwischen den Komponenten Positionsbestimmung, Systemsteuerung und den Control-Unit's über das Netzwerk und das UDP Protokoll erfolgen. Der erste Ansatz mit den separaten Modulen *UDP-Server* und *UDP-Client* wurde überarbeitet und zu dem Modul *NetworkMessenger* zusammengefasst. Grund hierfür war die Verwirrung, welche durch die Begriffe Client (Sender) und Server (Empfänger) entstand. Des Weiteren sollen alle Komponenten sowohl empfangen, als auch senden können. Es müssten somit immer beide Module eingebunden werden. So wurden diese zu einem gemeinsamen Modul zusammengefasst.
Des Weiteren wurde das Modul *NetworkMessengerStore* bzw. ein Nachrichtenspeicher implementiert, welches die Funktionalitäten zur Verwaltung von empfangenen Nachrichten bzw. dessen Informationen in Ringpuffern zur Verfügung stellt.
Beide Module sind voneinander abhängig und sollen ohne Anpassungen universell in jeder Komponente eingesetzt werden können, welche in der Programmiersprache C/C++ implementiert ist. Damit lassen sich beide Module in der Positionsbestimmung, der Control-Unit und mit Hilfe der JNA Bibliothek auch in der Systemsteuerungssoftware verwenden, welche in Java implementiert ist. Die Entwickler der jeweiligen Komponente müssen die Module lediglich in das jeweilige Projekt inkludieren und die benötigten Funktionen aufrufen. Bevor die Funktionen aufrufbar sind, müssen der Nachrichtenspeicher, ein UDP-

Sender und ein UDP-Empfänger initialisiert werden.

10.2.1 Netzwerkpaket

Für die Kommunikation zwischen den einzelnen Komponenten existieren, wie in Kapitelabschnitt 4.5 beschrieben, mehrere Nachrichtentypen. Jede dieser Nachrichtentypen verwendet unterschiedliche Variablen und Datentypen, was den Empfang dieser Nachrichten komplexer gestaltet. Jede Nachricht, welche über einen UDP-Socket gesendet und empfangen wird, muss eindeutig definiert sein. Daher wurde ein einheitliches Netzwerkdatenpaket definiert, welches je nach Nachrichtentyp mit einem Identifikator und den entsprechenden Variablen beschrieben wird. Das Netzwerkdatenpaket besteht aus einer Struktur, welches sämtliche Variablen aller Nachrichtentypen, den Identifikator, zwei Zeitstempel und einen Absender enthält.

In der Tabelle 10.1 werden sämtliche Variablen des Netzwerkpaketes aufgelistet, deren Nutzen kurz beschrieben und in dem jeweiligen Nachrichtentypen zugeordnet.

10.2.2 NetworkMessenger

Der NetworkMessenger bietet sämtliche Funktionalitäten für die Initialisierung von UDP-Sockets und zum Empfangen und Senden von Netzwerkdatenpaketen.

10.2.2.1 Initialisierung

Für die Initialisierung müssen Exemplare von bestimmten Datentypen erzeugt werden, auf welche im gesamten Programmablauf zugegriffen werden kann. Für einen UDP-Sender und einen UDP-Empfänger müssen jeweils ein Exemplar vom Datentypen `Serverdata` erzeugt werden. Der Datentyp `Serverdata` ist eine Struktur, welche sämtliche Informationen zum Senden oder Empfangen für den UDP-Socket enthält. In dieser Struktur müssen die Variablen `sendToLocalhostOnly` oder `receiveFromLocalhostOnly` und `broadcastPort` gesetzt werden. Die Variable `sendToLocalhostOnly` wird hierbei auf den Wert ”1” gesetzt, wenn versendete Nachrichtenpakete nur an den Localhost (Broadcast-IP: 127.0.0.1) gesendet werden sollen. Analog dazu kann diese Variable auf den Wert ”0” gesetzt werden, wenn versendete Nachrichtenpakete an alle Netzwerkgeräte (Broadcast-IP: 255.255.255.255) gesendet werden sollen. Gleiches gilt für die Variable `receiveFromLocalhostOnly`, wenn Nachrichten nur vom Localhost oder von sämtlichen Netzwerkgeräten empfangen werden sollen.
Zusätzlich muss für den UDP-Empfänger ein Exemplar vom Datentypen `Messagestore`

Nachrichtentyp mit Identifikator	Variable	Beschreibung
	int identifier	Der Identifikator welcher den Nachrichtentypen angibt.
	timeval timestamp	Zeitstempel welcher das Alter der Daten angibt.
	timeval receivedAt	Zeitstempel welcher zum Empfangs-zeitpunkt vom Empfänger beschrieben wird.
	int messageSender	Absender-ID der Nachricht.
Fahrzeugdaten (1)	int availableCardata[16]	Array mit Informationen, ob Fahrzeug-informationen vorhanden sind (Index entspricht der Fahrzeug-ID).
	double xCoord[16]	Array mit X-Koordinaten sämtlicher Fahrzeuge (Index entspricht der Fahrzeug-ID).
	double yCoord[16]	Array mit Y-Koordinaten sämtlicher Fahrzeuge (Index entspricht der Fahrzeug-ID).
	double alignment[16]	Array mit den Ausrichtungswinkeln sämtlicher Fahrzeuge (Index entspricht der Fahrzeug-ID).
Kontrolldaten (2)	int controlSignal	Kontrollsignal zum Wechsel der Systemzustände.
	int readySignal	Kontrollsignal zur Bestätigung der korrekten Initialisierung.
Fehlerdaten (3)	int emergencyStop	Not-Aus-Befehl, welcher nur von der Systemsteuerung versendet wird.
	int faultCode	Fehlercode (siehe Kapitelabschnitt 14.1).
Zeitdaten (4)	timeval StartTimeStamp	Entspricht dem Zeitstempel *timestamp*.
	timeval EndTimeStamp	Zeitstempel, welcher nach dem Senden von Regelungswerten zum Fahrzeug von der Control-Unit gesetzt wird (Gesamt-programmlaufzeit).

Tabelle 10.1: Variablen der jeweiligen Nachrichtentypen und einer dazugehörigen Kurzbeschreibung.

erzeugt werden, damit empfangene Nachrichten bzw. deren Informationen direkt in den Ringpuffern gespeichert und bereitgestellt werden können. Weitere Informationen zum Ringpuffer folgen in Kapitelabschnitt 10.2.3.
Die Initialisierung wird mit den Funktionen `initializeUDPReceiver`, `initializeMessagestore` und `initializeUDPSender` ausgeführt. Eine Deinitialisierung kann mit den Funktionen `closeUDPReceiver` und `closeUDPSender` ausgeführt werden.

10.2.2.2 Netzwerkpakete versenden

Um Netzwerpakete senden zu können, muss ein UDP-Sender initialisiert sein (siehe Kapitelabschnitt 10.2.2.1) und ein Exemplar des Datentypen `Networkpackage` erzeugt werden. Beide werden als Pointer in den Funktionen zum Senden übergeben.
Mit der Funktion `cleanNetzworkpackage` wird das Netzwerkpaket zurückgesetzt. Dies bedeutet, dass sämtliche darin enthaltenen Variablen auf den Wert ”0” gesetzt werden und das Netzwerkpaket bereit zum Beschreiben ist.
Mittels der Funktionen `setNetworkpackageTimestamp` und `setNetworkpackageSender` muss der Sender dem Empfänger kenntlich machen, wie alt die Daten im Netzwerkpaket sind und von wem diese Abgeschickt wurden. Anschließend können mittels einer der Funktionen `setNetworkpackageCardata`, `setNetworkpackageControldata`, `setNetworkpackageFaultdata` oder `setNetworkpackageTimedata` die jeweiligen Variablen des Nachrichtentypen beschrieben werden. Die Funktion `setNetworkpackageCardata` muss iterativ für die Fahrzeugdaten jedes einzelnen Fahrzeuges ausgeführt werden. Die übrigen drei Funktionen lassen sich nur einmalig ausführen, denn der Identifikator wird von der jeweiligen Funktion automatisch gesetzt. Damit wird das Netzwerkpaket als benutzt gekennzeichnet, was bedeutet, dass der Identifikator einen Wert größer als ”0” trägt. Ist der Wert des Identifikators gleich ”0”, so ist das Netzwerkpaket als unbenutzt bzw. zurückgesetzt gekennzeichnet. Damit wird das Beschreiben eines Netzwerkpaketes mit verschiedenen Nachrichtentypen verhindert.
Mit der Funktion `sendNetworkpackage` wird das beschriebene Netzwerkpaket über den UDP-Socket bzw. den UDP-Sender verschickt. Nach dem Senden bzw. vor dem nächsten Beschreiben des Netzwerkpakets muss dieses mittels der Funktion `cleanNetzworkpackage` zurückgesetzt werden.

10.2.2.3 Netzwerkpakete empfangen

Um Netzwerkpakete empfangen zu können, müssen ein UDP-Empfänger und der Nachrichtenspeicher (siehe Kapitelabschnitt 10.2.2.1) initialisiert sein. Beide werden als Pointer in der Funktion zum Empfangen übergeben. Der Nachrichtenspeicher wird für den Zugriff auf den Ringpuffer benötigt. Weitere Informationen zum Ringpuffer folgen in

Kapitelabschnitt 10.2.3.
Um Nachrichten empfangen zu können, wird die Funktion `receiveNetworkpackage` genutzt. Um keine Nachrichten zu verpassen bzw. ständig empfangsbereit zu sein, sollte diese Funktion in einer endlichen Schleife ausgeführt werden. Sobald eine Nachricht empfangen wurde, wird die Funktion `storeMessage` ausgeführt, welche die Nachricht bzw. dessen Informationen in einem Nachrichtenspeicher ablegt. Weiteres zur Handhabung und Funktion des Nachrichtenspeichers ist in Kapitelabschnitt 10.2.3 beschrieben.

10.2.3 NetworkMessengerStore

Ein Netzwerkpaket zu empfangen ist mittels der Funktion `receiveNetworkpackage` (siehe Kapitelabschnitt 10.2.2.3) möglich. Das Problematische besteht darin dieses Netzwerkpaket so bereitzustellen, dass weitere Prozesse/Threads dieses bzw. dessen Informationen lesen können. Lese-Prozesse wären hier ein Watchdog oder das *SCADE*-Modell. Ein Schreiber-Prozess wäre hierbei der UDP-Empfänger, welcher das empfangene Nachrichtenpaket in globalen Variablen bereitstellt. Hierdurch kann ein Lese-Schreiber-Problem entstehen. Während einer der Prozesse die globalen Variablen ausliest, überschreibt der Empfänger diese. Geschieht dies zur selben Zeit, können unvollständige oder nicht zusammenhängende Datensätze entstehen. Des Weiteren kann ein Nachrichtenpaket verschiedene Nachrichtentypen enthalten. Mit dem Empfang von beispielsweise Kontrolldaten würde der Empfänger die Variablen der Fahrzeugdaten überschreiben. Prozesse, welche auf die Fahrzeugdaten zugreifen, würden diese Daten nicht mehr lesen können und müssten auf neue Fahrzeugdaten warten. Somit musste eine Lösung entworfen werden um die Informationen jedes Nachrichtentyps unabhängig vom empfangenen Netzwerkpaket bereitstellen zu können und das Lese-Schreiber-Problem zu lösen.

Um jede Nachricht bereitstellen zu können, wurde der `NetworkMessengerStore` bzw. ein Nachrichtenspeicher implementiert. Der Nachrichtenspeicher besteht aus den vier Strukturen `Vehicledata`, `Controldata`, `Faultdata` und *Timedata*. `Vehicledata` enthält hierbei die Fahrzeugdaten, `Controldata` die Kontrolldaten, `Faultdata` die Fehlerdaten und `Timedata` die Zeitdaten. Jede dieser Strukturen enthält neben den relevanten Variablen ebenfalls die Absender-ID und den Empfangszeitpunkt des Nachrichtenpakets als Zeitstempel und wird als Arrays mit jeweils drei Zeigern (siehe Abbildung 10.6) initialisiert. Ein Lesezeiger, welcher auf den Arrayindex mit dem aktuellen Datensatz (i) zeigt. Ein weiterer Lesezeiger, welcher auf den Arrayindex mit den letzten Datensatz ($i-1$) zeigt. Und ein Schreiberzeiger, welcher auf den Arrayindex zeigt, in den der UDP-Empfänger den nächsten Datensatz ablegen soll ($i+1$) zeigt. Da jeder Nachrichtentyp einen eigenen Ringpuffer besitzt, bleibt die Übersicht und Rückverfolgbarkeit für die An-

wender bestehen.

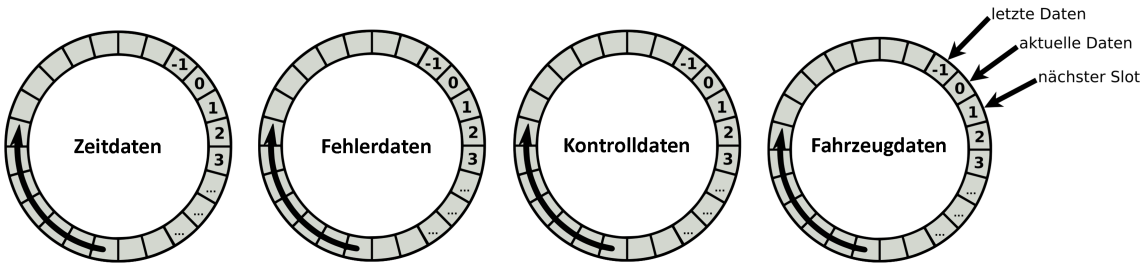


Abbildung 10.6: Ringpuffer des *NetworkMessageStore*

Jeder dieser Arrays stellt hierbei einen Ringpuffer mit den entsprechenden Zeigern für den jeweiligen Nachrichtentyps dar. Alle Zeiger werden hierbei nach jedem Schreibvorgang in den Ringpuffer verschoben, sodass immer der aktuelle, der letzte und der nächste Arrayindex ermittelt werden können, der Ringpuffer nicht überlaufen kann und alte Informationen nicht mehr betrachtet werden, wenn bereits neuere Informationen vorhanden sind.

Die Funktion `receiveNetworkpackage` führt die Funktion `storeMessage` aus, nachdem ein Netzwerkpaket empfangen wurde. Die Funktion `storeMessage` prüft anhand des Identifikators um welchen Nachrichtentyp es sich handelt und legt dessen Informationen in den entsprechenden Ringpuffer mit dem Index des Schreiberzeigers ab. Ist das ablegen des Datensatzes erfolgreich abgeschlossen, so wandern alle Zeiger um eine Stelle $(i+1)$ weiter, sodass der Schreibzeiger auf den nächsten Arrayindex und der Lesezeiger auf den eben beschriebenen Arrayindex mit den aktuellen Informationen zeigt.

10.2.4 Zusammenfassung

Abbildung 10.7 fasst das Senden (siehe Kapitelabschnitt 10.2.2.2) und Empfangen (siehe Kapitelabschnitt 10.2.2.3 und 10.2.3) von Netzwerkpaketen schematisch dar.

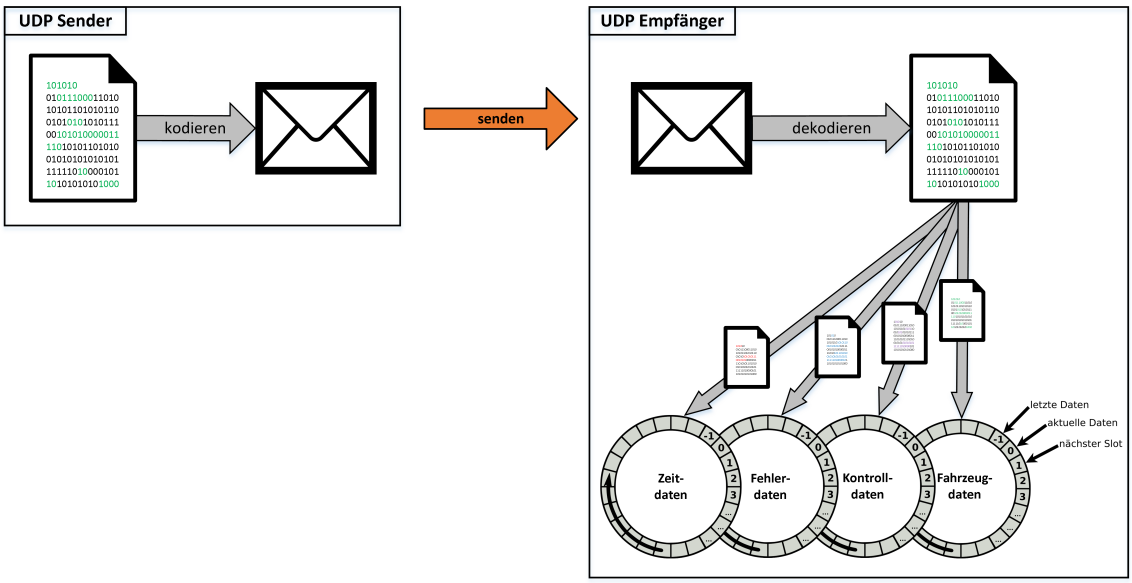


Abbildung 10.7: Schemeatische Darstellung der Übertragung von Netzwerkdatenpaketen

Der UDP-Sender kodiert seine Informationen mittels einer bereitgestellten Funktion in einem Netzwerkpaket. Dieses wird - je nach Einstellung vor der Initialisierung - an alle UDP-Empfänger im Netzwerk oder nur an einen UDP-Empfänger auf dem Localhost gesendet.

Der UDP-Empfänger empfängt das Netzwerkpaket, dekodiert dieses indem er anhand des Identifikators die relevanten Informationen für den jeweiligen Nachrichtentypen ausliest und diese im Nachrichtenspeicher bzw. Ringpuffer des jeweiligen Nachrichtentypen ablegt. Zusätzlich werden zu den Informationen der Empfangszeitpunkt und die Absender-ID abgelegt.

Diese Lösung gleicht einem klassischen Ringpuffer, welcher jedoch nach dem FIFO-Prinzip (first-in-first-out) handelt. In unserem Anwendungsszenario wäre es unangebracht veraltete vor neueren Informationen abzuarbeiten. Daher wandern nach dem Schreiben in den Ringpuffer sowohl die Lese-, als auch der Schreiberzeiger um alte Datensätze direkt zu verwerfen. Somit kann das Lese-Schreiber-Problem verhindert werden. Da von jedem Nachrichtentyp ein eigener Ringpuffer existiert, können die Nachrichtentypen nicht gegenseitig überschrieben werden. Jeder Lese-Prozess kann weiterhin auf alle aktuellen und die letzten Informationen zugreifen.

10.3 Control-Unit

Die Control-Unit setzt sich aus den drei Komponenten Wrapper Code, *SCADE* Modell und CarControl zusammen. Der Wrapper Code und das *SCADE* Modell bilden hierbei eine ausführbare Datei, welche in der Programmiersprache C implementiert wurde und auf dem Rechner der Control-Unit ausgeführt wird. Auf dem Entwicklungsboard der CarControl befindet ein Mikrocontroller, auf welchem ebenfalls Software ausgeführt wird. Die Kommunikation zwischen dem Rechner der Control-Unit und der CarControl erfolgt über eine serielle Schnittstelle, welche mittels eines FTDI Basic-Breakout über USB realisiert wird.
Im folgenden wird auf die aktuelle Implementierung der drei Komponenten der Control-Unit eingegangen.

10.3.1 Wrapper Code

Der Wrapper Code stellt das Grundgerüst der Control-Unit dar. Er bietet die Funktionalität zur Kommunikation mit anderen Systemkomponenten, zum Einbinden des *SCADE* Modells, zur Kommunikation mit der CarControl, zum Überwachen der Realzeitbedingungen und der Plausibilität und zum Wechseln der Systemzustände und den damit verbundenen Aktionen.

Der Wrapper Code wurde in der Programmiersprache C implementiert. Dies ist mit dem *KCG* Codegenerator von *SCADE* zu begründen, welcher den Programmcode des *SCADE* Modells in der Programmiersprache C generiert. Zum Zeitpunkt der Entwicklung wurde uns von erfahrenen Programmierern empfohlen, trotz der Kompatibilität des von *SCADE* generierten Programmcode zu C++, die Programmiersprache C zu verwenden, da keines der Projektgruppenmitglieder Erfahrungen in den Programmiersprachen C und C++ hat und eine Mischung dieser Programmiersprachen nur für erfahrene Programmierer empfohlen wird.

Im folgenden werden die einzelnen Funktionalitäten und die daraus resultierende Softwarearchitektur der Control-Unit vorgestellt.

10.3.1.1 FileReader

Während der Implementierungsphasen mussten bestimmte Parameter festgelegt werden, um Funktionen zu de- oder aktivieren. Da zum Schluss ein fertiges Softwareprodukt entstehen sollte, kann dem Endnutzer nicht überlassen werden, bestimmte Programmcode-

zeilen anzupassen oder einzukommentieren, um bestimmte Funktionalitäten anzupassen. Bereits in der frühen Entwicklungsphasen war bewusst, dass das Einlesen von Parametern aus Dateien unumgänglich ist. So wurden die zwei Module *ConfigFileReader* und *RacetrackFileReader* implementiert.

ConfigFileReader

Bei dem *ConfigFileReader* handelt es sich um ein in der Programmiersprache C implementiertes Modul, welches Parameter aus einer Textdatei bzw. einer Konfigurationsdatei einliest. Der Dateiname der Konfigurationsdatei muss hierbei im Programmcode der Control-Unit fest einkodiert werden, damit diese immer gefunden werden kann. Im aktuellen Entwicklungszustand heißt die Datei `controlunit.config` und muss sich im gleichen Verzeichnis befinden, wie die ausführbare Programmdatei der Control-Unit. In dieser Konfigurationsdatei befinden sich sämtliche Parameter, welche für den Betrieb der Control-Unit notwendig sind. Hierzu zählen Parameter wie die Fahrzeug-ID, der Name der seriellen Schnittstelle zur CarControl, der Broadcast-Port für die UDP-Sockets und Funktionsschalter für einzelne Funktionalitäten. Das Einlesen der Konfigurationsdatei wird mit der Funktion `readConfigFile()` gestartet. Sämtliche Parameter werden in einer Struktur des Datentypen `ConfigParameterCU` abgelegt, welche sämtliche Variablen der Konfigurationsparameter enthält. Über diese Struktur sind die Parameter im gesamten Programmcode aufrufbar. Eine vollständige Auflistung sämtlicher Konfigurationsparameter, dessen Datentyp, Wertebereich und einer Kurzbeschreibung sind in Kapitelabschnitt 14.6 in Tabelle 14.3 zu finden.

RacetrackFileReader

Für den Betrieb der Control-Unit bzw. für die Berechnung der Regelungswerte wird eine Abbildung bzw. ein Modell der zu befahrenden Rennstrecke benötigt. Dieses liegt in Form einer Rennstreckendatei (siehe Kapitelabschnitt 10.3.3.4) vor und muss eingelesen werden. Da die benötigten Rennstreckendaten in einer kommaseparierten CSV Datei vorliegen, wurde an dieser Stelle das Modul `RacetrackFileReader` implementiert. Mittels der Funktion `readMapFile()` wird die CSV Datei - dessen Dateiname in den Konfigurationsparametern hinterlegt ist - geöffnet und zeilenweise eingelesen. Jede Zeile wird hierbei in einen Puffer geladen, Zeichen für Zeichen analysiert und in ein von SCADE generiertes, zweidimensionales Integer-Array des Datentypen `mapArray` übertragen. Dieses Verfahren wird wiederholt, bis sämtliche Zeilen der CSV Datei ausgelesen wurden. Jeder Eintrag in diesem zweidimensionalen Array entspricht einer Fläche von 1x1 cm auf der Rennstrecke. Bei der verwendeten Rennstreckengröße von 237x237 cm entspricht das 236 Zeilen und 236 Spalten, da Tabellenkalkulationsprogramme ab dem Wert 1 zählen und

der erste Arrayindex den Wert 0 trägt. Die Codierung eines Eintrags mit Null, Eins oder Zwei repräsentiert die jeweilige Beschaffenheit der Rennstrecke. Der befahrbare Bereich wird durch eine 1 dargestellt. Der nicht-befahrbare Bereich durch eine 0. Die Zieltrajektorie kennzeichnet eine 2. Hierbei ist zu beachten, dass die Rennstreckenbegrenzung etwa 5 cm breit und nicht befahrbar ist. Über den jeweiligen Arrayindex bzw. die Koordinate, kann die Beschaffenheit der Rennstrecke so abgefragt werden.

10.3.1.2 Initialisierung

Bevor die Control-Unit korrekt arbeiten kann, muss eine Initialisierung durchgeführt werden, um benötigte Konfigurationsparameter für die einzelnen Funktionalitäten einzulesen und Grundfunktionalitäten bereitzustellen. Ein Teil der Konfigurationsparameter wird jedoch bereits benötigt bevor die Systemsteuerung die offizielle Initialisierung startet, wie beispielsweise ein UDP-Empfanger für die Kommunikation mit der Systemsteuerungs selbst, um den Kontrollbefehl für die Initialisierung zu erhalten. Hierzu wurde die Initialisierungsroutine in eine erste und zweite Initialisierungsroutine (`firstStageInitialization()` und `secondStageInitialization()`) aufgeteilt. Die erste Initialisierungsroutine wird hierbei direkt nach dem Programmstart der Control-Unit ausgeführt. Die zweite Initialisierungsroutine wird erst nach erhalt des Kontrollbefehls für die Initialisierung von der Systemsteuerung ausgeführt.

`firstStageInitialization()`
Die erste Initialisierungsroutine bzw. die erste Stufe der Initialisierung wird direkt bei Programmstart der Control-Unit ausgeführt. Sie führt die Funktion `readConfigFile()` aus, um die Konfigurationsdatei einzulesen (siehe Kapitelabschnitt 10.3.1.1). Anhand dieser Konfigurationsparameter werden der UDP-Empfänger und UDP-Sender initialisiert. In diesem Zustand wartet die Control-Unit auf den Kontrollbefehl der Systemsteuerung, um mit der Initialisierung beginnen zu dürfen und ist bereits dazu in der Lage weitere Kontrollbefehle und Fehlerdaten empfangen und senden zu können. Weiteres zur Initialisierung wird in Kapitelabschnitt 10.2.2.1 und 10.3.1.6 beschrieben.

`secondStageInitialization()`
Die zweite Initialisierungsroutine bzw. zweite Stufe der Initialisierung wird nur auf Anweisung der Systemsteuerung ausgeführt. Sie Initialisiert die serielle Schnittstelle zur CarControl, prüft die Verbindung und ändert den aktuellen Modus der CarControl. Weiteres zu der Kommunikation und den Funktionalitäten der CarControl werden in Kapitelabschnitt 10.3.1.6 detaillierter beschrieben. Um der Systemsteuerung eine korrekte Initialisierung mitzuteilen, wird der entsprechende Kontrollbefehl (`readySignal`) verschickt.

10.3.1.3 Netzwerkempfang

Wie in Kapitelabschnitt 10.2.2.3 und 10.2.3 beschrieben, kann über die Funktion `receiveNetworkPackage()` ein Netzwerkpaket empfangen und im Nachrichtenspeicher abgelegt werden. Um keines der Fahrzeug- , Kontroll- oder Fehlerdaten zu verpassen, wird die Funktion`receiveNetworkpackage()` in einer While-Schleife ausgeführt. Somit ist gewährleistet, dass sämtliche Nachrichten empfangen werden. Über das Exemplar des Nachrichtenspeicher, in welchem sämtliche empfangenen Daten abgelegt werden, und dem Lesezeiger kann auf die aktuellsten Fahrzeug-, Kontroll- oder Fehlerdaten zugegriffen werden. Warum der Einsatz eines Nachrichtenspeichers bzw. von Ringpuffern notwendig ist, wird in Kapitelabschnitt 10.3.1.8 beschrieben.

10.3.1.4 Trajektorienfindung

Bevor das *SCADE* Modell arbeiten kann, wird die Funktion `findIntersections()` ausgeführt. Diese Funktion liest aus dem Nachrichtenspeicher die aktuellen Fahrzeugdaten bzw. die X- und Y-Koordinate des Fahrzeuges und legt einen virtuellen Kreis mit dem Radius *R* um das Fahrzeug. Je nach Größe des Kreises bildet dieser immer null bis zwei Schnittpunkte mit der Trajektorie. Die Abbildung 10.8 veranschaulicht dies. Die blaue Linie stellt die Trajektorie dar und simuliert eine Kurve. Der grüne Kreis bildet dabei zwei Schnittpunkte (rot markiert) mit der Trajektorie. Die zwei gelb-orangenen Linie in der Abbildung 10.8 beschreiben die Ausrichtung des Fahrzeugs, sowie den Schnittpunkt der Zieltrajektorie zum Mittelpunkt des Fahrzeugs. Anhand dieser Linien wird im *SCADE* Modell der benötigte Lenkeinschlag bestimmt, um die Ausrichtung im Hinblick zur Trajektorie zu korrigieren (siehe Kapitelabschnitt 10.3.3.12).
Im Optimalfall schneidet der Kreis, wie in der Abbildung 10.8 dargestellt, mit zwei Punkten die Trajektorie. Ist das Fahrzeug weit von der Trajektorie entfernt, könnte es jedoch auch zu lediglich einem oder gar null Schnittpunkten kommen. Wird kein Schnittpunkt gefunden, existiert keine Zieltrajektorie die angesteuert werden kann. In dem Fall wird die Funktion `handleToFailedTrajectoryFinding()` ausgeführt. Weiteres zur Fehlerbehandlung kann dem Kapitelabschnitt 10.3.1.11 entnommen werden.

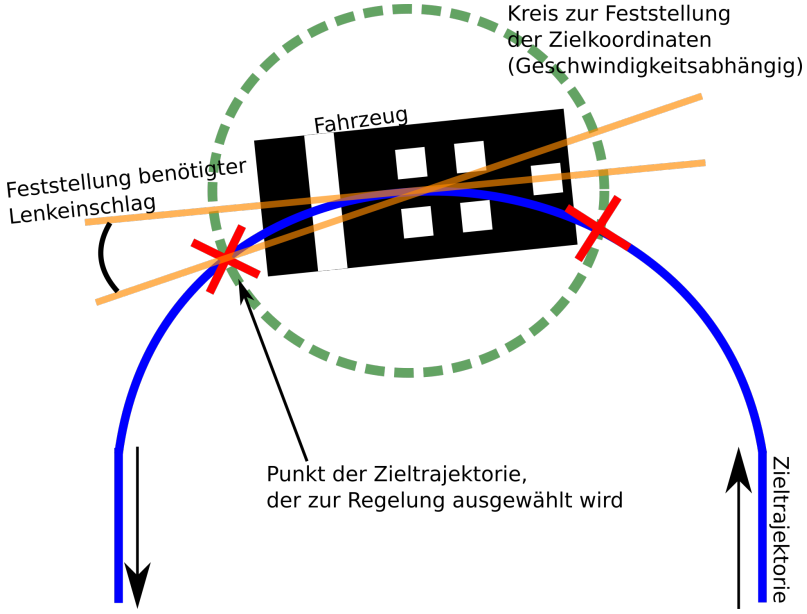


Abbildung 10.8: Die Abbildung veranschaulicht die Funktion c, welche einen Kreis um das Fahrzeug legt und Null bis zwei Schnittpunkte mit der Trajektorie sucht.

Die Funktion `findIntersections()` realisiert die Funktion des grünen Kreises um das Fahrzeug. Der Radius des Kreises ist abhängig von der Geschwindigkeit des Fahrzeugs (siehe Kapitelabschnitt 10.3.3.7) und beschreibt sich über die Formel 10.4. Der vorgegebene Radius ($R_{default}$) kann in der Konfigurationsdatei der Control-Unit eingestellt werden.

$$R = v \cdot R_{default} \tag{10.4}$$

Die obere sowie untere Grenze des Radius R kann ebenfalls in der Konfigurationsdatei (siehe Tabelle 14.3 in Kapitelabschnitt 14.5) der Control-Unit definiert werden. So wird vermieden, dass durch eine sehr hohe oder sehr niedrige Geschwindigkeit des Fahrzeugs (beispielsweise während den Einführungsrunden (siehe Kapitelabschnitt 10.3.3.9)) der Kreis zu klein oder zu groß wird. Ein zu kleiner Kreis sorgt besonders in den Kurven durch die geringe Auflösung der Rennstrecke von 1x1 cm zu starken Peaks in der Querregelung des Fahrzeugs, da die Trajektorie dort nicht mehr gerade verläuft. Ein zu großer Kreis kann ein zu frühes Einlenken in die Kurven verursachen.
Um die Schnittpunkte des Kreises mit der Trajektorie festzustellen sucht die Funktion `findIntersections()` zu Beginn alle im Umfeld befindlichen Trajektorienpunkte. Im Anschluss daran wird jeder gefundene Trajektorienpunkt durch Formel 10.8 geprüft, ob dieser gleich weit oder weiter von dem Fahrzeug entfernt ist, als der Kreis um das Fahr-

zeug groß ist.

$$R \leq \sqrt{(X_{Trajectory} - X)^2 + (Y_{Trajectory} - Y)^2} \tag{10.5}$$

Danach wird die Funktion `findNearestIntersections()` aufgerufen, welche aus den zuvor gefundenen Trajektorienpunkten, welche gleich weit oder weiter vom Fahrzeug entfernt sind als der Kreis groß ist, die Trajektorienpunkte sucht, welche von den zuvor gefundenen Punkten am nächsten am Fahrzeug liegen. Im Idealfall sind diese genau so groß, wie der Radius R des Kreis. Durch Abweichungen schneiden diese Punkte jedoch nicht immer genau den Kreis.
Die ein bis zwei gefundenen Schnittpunkte werden im Anschluss in die entsprechenden Eigangsvariablen des *SCADE* Modells geschrieben und dort weiter verarbeitet. Sollte kein Schnittpunkt gefunden werden, wird wie zuvor beschrieben abgebrochen. Innerhalb des *SCADE* Modells (siehe Kapitelabschnitt 10.3.3.12) wird geprüft, welcher der (möglichen) zwei Schnittpunkt der Schnittpunkt ist, der den Zieltrajektorienpunkt darstellt.

Damit fortlaufend neue Trajektorienpunkte ermittelt werden können, wird die Funktion `findIntersections()` in einer While-Schleife ausgeführt.

10.3.1.5 Ausführung des SCADE Modells

Um das *SCADE* Modell ausführen zu können, müssen die Eingangsvariablen des *SCADE* Modells vom Datentypen `inC_RootNode` belegt werden. Voraussetung für den Zugriff auf diese Struktur ist, dass die Header-Datei `RootNode.h` im Projekt inkludiert wurde. Diese Struktur enthält sämtlichen Eingangsvariablen des *SCADE* Modells. Hierzu zählen unter anderem der Zeitstempel, die X-Koordinate, die Y-Koordinate, der Ausrichtungswinkel und die Information, ob das Fahrzeug detektiert wurde. Diese Informationen werden aus dem Nachrichtenspeicher ausgelesen. Zusätzlich werden einige Eingangsvariablen mit Parametern aus der Konfigurationsdatei für direkte Einstellungen des *SCADE* Modells belegt (siehe Tabelle 14.3 in Kapitelabschnitt 14.5).

Das *SCADE* Modell wird über die von *SCADE* generierte Hauptfunktion `RootNode()` ausgeführt. Es berechnet anhand der anliegenden Fahrzeugpose an den Eingangsvariablen und der Einstellungen die entsprechenden Regelungswerte für die Längs- und Querführung und legt diese an die Ausgabeschnittstellen des *SCADE* Modells vom Datentypen `outC_RootNode`. Dieser Datentyp ist eine Struktur mit sämtlichen Ausgangsvariablen

des *SCADE* Modells. Wird intern im *SCADE* Modell während der Ausführung ein Fehler festgestellt, so wird die importierte Funktion `LogError()` mit einem internen Fehlercode ausgeführt. Diese Funktion führt sofort einen Not-Halt des Fahrzeuges aus und ruft die Funktion `handleToFailedSCADEModell()` mit dem internen Fehlercode als Parameter aus. Damit das *SCADE* Modell fortlaufend neue Regelungswerte berechnen kann, wird die Funktion `RootNode()` in einer While-Schleife ausgeführt.

Wichtig ist, dass die Funktion `RootNode_reset()` einmalig vor dem ersten Ausführen der Funktion `RootNode()` ausgeführt wird, damit das *SCADE* Modell korrekt ausgeführt werden und arbeiten kann. Mit dieser Funktion wird das *SCADE* Modell initialisiert bzw. zurückgesetzt. Weiteres zum Funktionsumfang des *SCADE* Modells kann dem Kapitelabschnitt 10.3.3 entnommen werden. Weiteres zur Fehlerbehandlung kann dem Kapitelabschnitt 10.3.1.11 entnommen werden.

10.3.1.6 Serielle Kommunikation

Um die vom *SCADE* Modell (siehe Kapitelabschnitt 10.3.1.5) berechneten Regelungswerte an die CarControl zu übermitteln bzw. mit dieser Kommunizieren zu können, wurde der *CarControlMessenger* - basiert auf dem *SerialMessenger* implementiert. Der *CarControlMessenger* wurde für die Kommunikation zwischen dem Wrapper Code und der CarControl implementiert und stellt die Funktionalitäten für das Initialisieren der seriellen Schnittstelle, das Prüfen der Verbindung, das Wechseln des Modus der CarControl, das Senden der Regelungswerte für die Längs- und Querführung und das Senden des Not-Halt Befehls an die CarControl.
Nachrichten, welche über die serielle Schnittstelle gesendet werden sollen, müssen in einem bestimmten Format vorliegen. Die Nachricht beginnt mit einem Flag, welches den Typen der Nachricht angibt und einer Endekennung. Die Endekennung besteht aus einem *Line Feed*, gefolgt von einem *Carriage Return* (kurz: LFCR bzw. ”\n\r”). Mögliche Nachrichten bzw. detailliertere Informationen können der Tabelle 10.2 in Kapitelabschnitt 10.3.2.1 entnommen werden.

Initialisierung
Bevor Nachrichten über die serielle Schnittstelle an die CarControl gesendet werden können, muss einmalig die serielle Schnittstelle initialisiert und geöffnet werden. Hierzu wird ein Exemplar des Datentyps `Connection` erstellt und als Parameter sämtlicher Funktionen des *CarControlMessengers* übergeben. Zusätzlich wird die Funktion `initializeSerialConnection()` zum Öffnen der seriellen Schnittstelle und der Ausführung von benötigten Einstellungen für eine asynchrone Kommunikation und der entsprechenden

Baudrate vorgenommen und die Funktion `initializeSerialMessages()` zum Initialisieren von Nachrichtenpuffern ausgeführt. Die Nachrichtenpuffer werden von internen Funktionen bei dessen Aufruf verwendet und müssen nicht weiter beachtet werden.

Prüfung der Verbindung

Um die Verbindung zur CarControl zu überprüfen, wurde die Funktion `checkConnection()` implementiert. Diese Funktion sendet ein Prüf-Flag an die CarControl und wartet mittels der Funktion `receiveFeedbackmessage()` auf eine definierte Rückantwort. Wenn die empfangene Rückantwort korrekt empfangen wurde, ist die Verbindung erfolgreich geprüft und der Funktionsrückgabewert 1. Wenn die Prüfung nicht durchgeführt werden konnte, weil die Verbindung der seriellen Schnittstelle nicht mehr besteht, ist der Funktionsrückgabewert -1. Wenn eine Rückantwort empfangen wurde, jedoch nicht der definierten Rückantwort entspricht, ist der Funktionsrückgabewert 0. Die Prüfung der Verbindung muss zwingend im Rahmen der Initialisierung der Control-Unit ausgeführt werden.

Moduswechsel

Um den Modus der CarControl zu wechseln, kann die Funktion `changeCarControlMode()` ausgeführt werden. Hierzu wird der jeweilige Modus als Integer-Wert als Parameter übergeben. Die Funktion sendet einen entsprechenden Flag für den Moduswechsel, gefolgt vom gewünschten Modus an die CarControl.

Senden von Regelungswerten

Um Regelungswerte an die CarControl senden zu können, wurden die Funktionen `sendSteeringValue()` und `sendThrottleValue()` implementiert. Diesen Funktionen wird der jeweilige Regelungswert für die Längs- oder Querführung als Parameter übergeben. Die Funktion prüft, ob der Regelungswert sich in einem gültigen Wertebereich befindet, versieht diesen mit dem benötigten Flag und der Endekennung und versendet diesen über die serielle Schnittstelle. Um die serielle Schnittstelle mit zwei separat versendeten Regelungswerten nicht zu überlasten, wurde die Funktion `sendSteeringAndThrottleValue()` implementiert, welche aktuell hauptsächlich verwendet wird und es ermöglicht sowohl den Regelungswert für die Längs-, als auch die Querführung gemeinsam in einer Nachricht zu übertragen. Die drei Funktionen übertragen den Regelungswert nur, wenn dieser nicht gleich dem zuletzt gesendeten Regelungswert entspricht. Ändert sich nur einer der beiden Regelungswerte, so kann die Funktion zum Senden eines einzelnen Regelungswertes verwendet werden. Auch dies ist eine Maßnahme um die serielle Schnittstelle nicht zu überlasten.

Damit fortlaufend Regelungswerte an die CarControl bzw. das Fahrzeug gesendet werden, wird die Funktion zum Senden von Regelungswerten in einer While-Schleife ausgeführt.

Not-Halt auslösen
Damit seitens des Wrapper Codes ein Not-Halt ausgeführt werden kann, wurde die Funktion `sendEmergencyStop()` implementiert. Diese Funktion sendet einen entsprechenden Flag an die CarControl, damit diese in den sicheren Fehlerzustand überführt und die Regelungswerte für die Längs- und Querführung auf die Neutralstellung gesetzt werden.

SoftReset
Um die CarControl zurücksetzen zu können bzw. einen SoftReset durchführen zu können, wurde die Funktion `resetCarControl()` implementiert. Diese Funktion sendet einen entsprechenden Flag für den SoftReset an die CarControl und kann direkt nach der Initialisierung der Control-Unit ausgeführt werden.

Weiteres zum Funktionsumfang der CarControl kann dem Kapitelabschnitt 10.3.2 entnommen werden.

10.3.1.7 Watchdog

Um die Realzeitanforderungen und Plausibilitätsüberprüfung durchzuführen wurde ein Watchdog implementiert. Dieser besteht aus einer Prüfroutine, welche die zuletzt empfangenen Fahrzeug-, Kontroll- und Fehlerdaten überwacht. Es werden sowohl die Zeitspannen zwischen dem Empfangszeitstempel und dem Zeitstempel der empfangenen Fahrzeug- und Kontrolldaten geprüft, als auch die Plausibilität der Inhalte der Fahrzeug-, Kontroll- und Fehlerdaten überprüft. Folgende Prüfungen werden im aktuellen Entwicklungszustand vorgenommen:

- **Fahrzeugdaten**
  - Prüfung, wann die letzten Fahrzeugdaten empfangen wurden.
  - Prüfung, ob die Fahrzeugdaten von der Positionsbestimmung verschickt wurden.
  - Prüfung, ob der Zeitstempel der Fahrzeugdaten älter ist als der Empfangs-Zeitstempel.
  - Prüfung, ob die letzten Fahrzeugdaten älter sind, als die aktuellen Fahrzeugdaten.
  - Prüfung, ob die Fahrzeugdaten eine Fahrzeugpose für das eigene Fahrzeug enthalten.

  - Prüfung, ob die erlaubte Differenz des Fahrzeuges auf der X-Achse zwischen den letzten beiden Fahrzeugdaten nicht überschritten wurde.
  - Prüfung, ob die erlaubte Differenz des Fahrzeuges auf der Y-Achse zwischen den letzten beiden Fahrzeugdaten nicht überschritten wurde.
  - Prüfung, ob die erlaubte Differenz der Ausrichtungswinkel des Fahrzeuges zwischen den letzten beiden Fahrzeugdaten nicht überschritten wurde.
  - Prüfung, ob das Fahrzeug sich außerhalb des Systemzustandes *Rennbetrieb* bewegt.

- **Kontrolldaten**
  - Prüfung, wann die letzten Kontrolldaten empfangen wurden.
  - Prüfung, ob der letzte Kontrollbefehl (außer dem "ReadySignal") von der Systemsteuerung versendet wurde.
  - Prüfung, ob der Zeitstempel der Kontrolldaten älter ist, als der Empfangs-Zeitstempel.
  - Prüfung, ob die letzten Kontrolldaten älter sind, als die aktuellen Kontrolldaten.
  - Prüfung, ob der angeforderte Zustandswechsel erlaubt bzw. plausibel ist.

- **Fehlerdaten**
  - Prüfung, ob der Not-Aus Befehl von der Systemsteurung versendet wurde.

In der Konfigurationsdatei der Control-Unit (siehe Tabelle 14.3 in Kapitelabschnitt 14.5) kann der Watchdog komplett oder nur Teile ein- oder abgeschaltet werden. Auch die Toleranzen einiger Prüfungen können eingestellt werden. Sollte eine Prüfung einen Fehler feststellen, wird je nach Prüfung eine der folgenden Funktion ausgeführt:

- `handleToDetectedVehicledataDelay()`
- `handleToFailedVehicledataPlausibilityCheck()`
- `handleToDetectedControldataDelay()`
- `handleToFailedControldataPlausibilityCheck()`
- `handleToFailedFaultdataPlausibilityCheck()`

Damit die Prüftroutine des Watchdogs fortlaufend ausgeführt wird, wird diese in einer While-Schleife ausgeführt. Weiteres zur Fehlerbehandlung kann dem Kapitelabschnitt 10.3.1.11 entnommen werden.

10.3.1.8 Multithreading

In der Programmiersprache C ist es üblich, dass Funktionen sequenziell ausgeführt werden. Dies würde bedeuten, dass beispielsweise neue Fahrzeugdaten während der Berechnung und dem Senden von Regelungswerten nicht empfangen werden können und ggf. verloren gehen. Für dieses Problem bietet sich die Bibliothek *Pthread* an. Diese Bibliothek muss auf dem Betriebssystem installiert und in ein Softwareprojekt eingebunden werden. Mithilfe von *Pthread* ist es dann möglich, einzelne Funktionen in eigene Sub-Threads auszulagern und diese so parallel auszuführen.

Mit der Funktion `pthread_create()` kann eine Funktion in einem separaten Thread ausgeführt werden. Dazu wird ein Exemplar des Datentypen `pthread_t` erstellt und zusammen mit der auszuführenden Funktion als Parameter übergeben. Das Exemplar des Datentypen `pthread_t` stellt die Identität des Sub-Threads dar und wird für die weitere Steuerung dieses Sub-Threads benötigt. Ab diesem Zeitpunkt läuft das Programm im Main-Thread parallel zu der Funktion im Sub-Thread weiter. Wenn der Haupt-Thread zum Programmende gekommen ist, wird automatisch auch der Sub-Thread - unabhängig von dessen Zustand - beendet. Damit dies nicht passiert, kann die Funktion `pthread_join()` vor Ende des Programmablaufs im Main-Thread gesetzt werden. Diese Funktion zwingt den Main-Thread zum Warten auf das Ende des Programmablaufs des im Parameter übergebenen Sub-Threads. Die Funktion `pthread_exit()` hingegen beendet sofort den als Parameter übergebenen Sub-Thread. Dies ist zu jedem Zeitpunkt des Main-Threads möglich.

Wie bereits in den Kapitelabschnitten 10.3.1.3, 10.3.1.4, 10.3.1.5, 10.3.1.6 und 10.3.1.7 beschrieben, werden die Funktionalitäten des Empfangens von Netzwerkdaten, die Trajektorienfindung, die Ausführung des *SCADE* Modells, das Senden von Regelungswerten an die CarControl und die Prüfroutine des Watchdogs in einer While-Schleife ausgeführt, um deren kontinuierliche Funktionalität zu gewährleisten. Dies würde bei einer sequenziellen Ausführung dazu führen, dass nur eine Funktion zur Zeit aktiv ist. Während beispielsweise das *SCADE* Modell die Regelungsdaten berechnet, können keine neuen Fahrzeugdaten empfangen werden und können verloren gehen. Daher wurden die Funktionalitäten, wie in Abbildung 10.9 dargestellt, auf insgesamt fünf While-Schleifen aufgeteilt. Diese While-Schleifen werden jeweils in einem eigenen (Sub-)Thread ausgeführt.

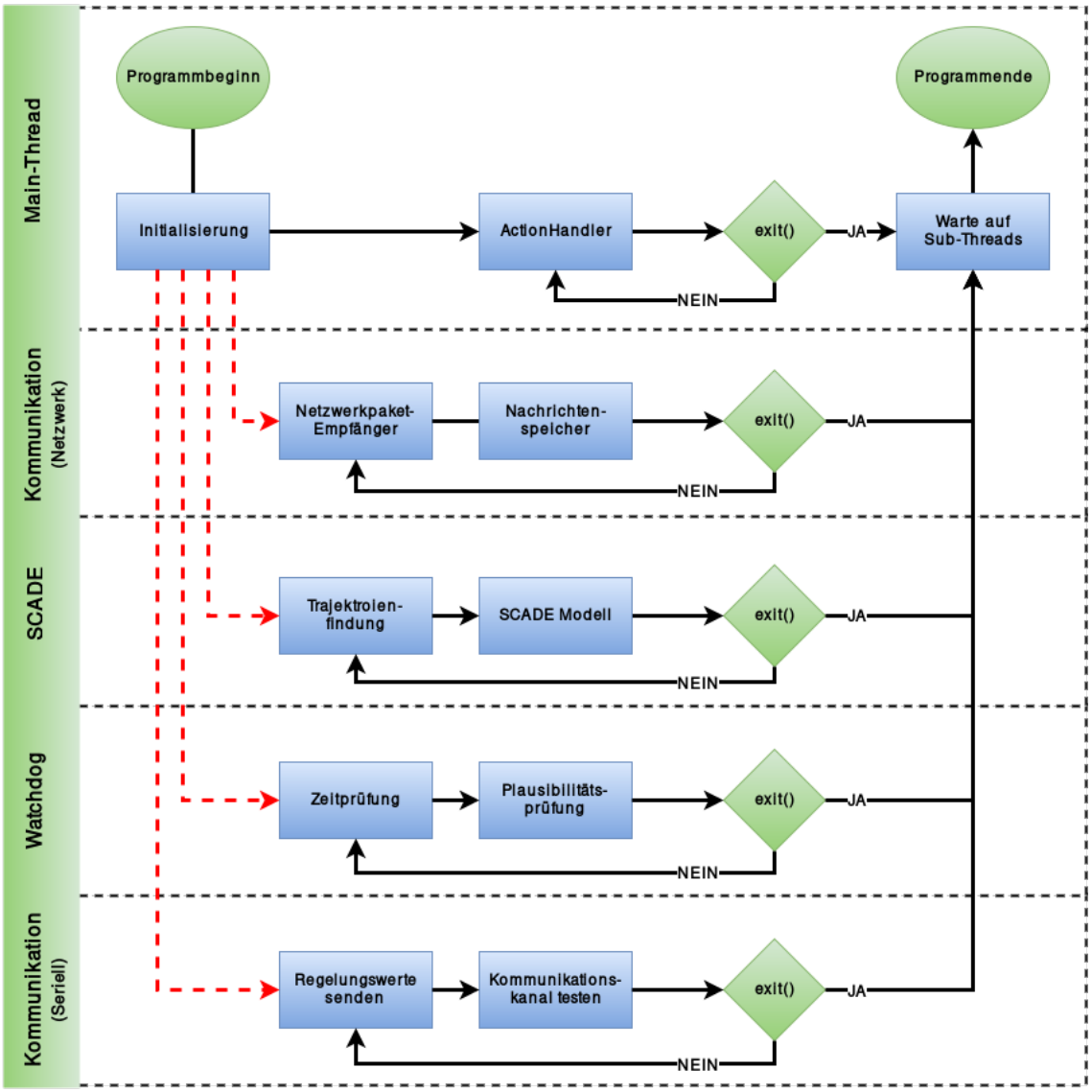


Abbildung 10.9: Programmablauf des Wrapper Codes.

Thread: Main-Thread

Der Main-Thread besitzt lediglich zwei Aufgaben. Zu Beginn wird die Initialisierung (siehe Kapitelabschnitt 10.3.1.2) der Control-Unit ausgeführt. Im Rahmen dieser Initialisierung werden vier Sub-Threads für die Kommunikation über das Netzwerk, die Ausführung des *SCADE* Modells und die Trajektorienfindung, die Prüfroutinen des Watchdogs und die Kommunikation über die serielle Schnittstelle erstellt und gestartet. Anschließend wird die Funktion `startActionHandler()` in einer While-Schleife ausgeführt, deren Funktionsweise in Kapitelabschnitt 10.3.1.10 beschrieben wird.

Thread: Kommunikation (Netzwerk)

Die Funktion `receiveNetworkpackage()` wird in einer While-Schleife ausgeführt, um

keine Netzwerkdatenpakete zu verpassen. Da im integrierten Nachrichtenspeicher des *NetzworkMessengers* ein System aus Ringpuffern eingesetzt wird, werden in anderen Sub-Threads parallel ausgeführte Funktionen des Wrapper Codes nicht negativ beeinflusst, da die Funktion `receiveNetworkpackage()` als einzige Funktion in den Nachrichtenspeicher schreibt.

Thread: SCADE

Da die Funktionen `findIntersections()` (siehe Kapitelabschnitt 10.3.1.4) und `RootNode()` (siehe Kapitelabschnitt 10.3.1.5) in Abhängigkeit voneinander ausgeführt werden, wurden diese in einer gemeinsamen While-Schleife zusammengefasst und in einem Sub-Thread ausgeführt. Wichtig ist hierbei, dass die Funktion `RootNode_reset()` einmalig vor dem Betreten der While-Schleife bzw. vor dem ersten Ausführen der Funktion `RootNode()` ausgeführt wird, damit das *SCADE* Modell korrekt ausgeführt werden und arbeiten kann. Die Funktion `findIntersections()` muss mindestens ein Mal vor dem *SCADE* Modell selbst ausgeführt werden.

Thread: Kommunikation (Seriell)

Um eine ständige Kontrolle des Fahrzeuges zu behalten werden die Funktionen `sendSteeringAndThrottleValue()` und `checkConnection()` in einer gemeinsamen While-Schleife ausgeführt. Die Funktion `sendSteeringAndThrottleValue()` sendet in einem festgelegten Takt ständig die vom *SCADE* Modell zuletzt berechneten Regelungswerte an die CarControl. Zusätzlich wird mit der Funktion `checkConnection()` in einem festgelegten Takt die Verbindung zur CarControl geprüft.

Thread: Watchdog

Die in Kapitelabschnitt 10.3.1.7 beschriebenen Prüfroutinen wurden in einer While-Schleife zusammengefasst und werden in einem eigenen Sub-Thread parallel ausgeführt. So werden unabhängig von den anderen Threads dessen Eingabe- und Ausgabevariablen ständig überwacht.

10.3.1.9 Management der Systemzustände

Wie bereits in Kapitelabschnitt 10.3.1.8 beschrieben, arbeitet der Wrapper Code mit 5 insgesamt (Sub-)Threads und damit zueinander parallel ausgeführten Funktionsroutinen, welche in While-Schleifen realisiert werden. Um die Routinen anhalten oder starten zu können, wurde ein Management für Systemzustände entwickelt. Das Management besteht aus Funktionen zum Initialisieren und wechseln der Systemzustände. Die Systemzustände (siehe Abbildung 4.6 in Kapitelabschnitt 4.6) wurden in einer Struktur vom Datentypen

Systemstate abgebildet (siehe Abbildung 10.10).

```
struct Systemstate {
    int inspection;
    int initialization;
    int standby;
    int race;
    int error;
    int off;
};
```

Abbildung 10.10: Struktur vom Datentypen Systemstate.

Diese Struktur besteht aus sechs Integer-Variablen, welche den Wert 1 (aktiv) oder 0 (inaktiv) annehmen können. Die Variable inspektion steht hierbei für den Systemzustand *Inspektion*. Die Variable initialization für den Systemzustand *Initialisierung*. Die Variable standby für den Systemzustand *Standby*. Die Variable race für den Systemzustand *Rennbetrieb*. Die Variable error für den Systemzustand *Fehler*. Die Variable off für den Systemzustand *Aus*. Nur eine dieser Variablen kann den Wert 1 annehmen, wenn der jeweilige Systemzustand aktuell aktiv ist. Die fünf übrigen Variablen müssen den Wert 0 annehmen.

Damit die Systemstzustände korrekt verwaltet bzw. gewechselt werden können, muss die Funktion initializeSystemstate() während der Initialisierung ausgeführt werden. Diese Funktion initialisiert erstmalig sämtliche Zustandsvariablen mit dem Wert 0. Anschließend kann der Systemzustand mit den folgenden Funktionen gewechselt werden:

- switchSystemstateToInspection()
- switchSystemstateToInitialization()
- switchSystemstateToStandby()
- switchSystemstateToRace()
- switchSystemstateToError()
- switchSystemstateToOff()

Bei der Ausführung dieser Funktionen wird zuerst der aktuelle Systemzustand abgefragt und geprüft, ob der Wechsel des Systemzustandes gemäß des Zustandsautomaten

in Abbildung 4.6 in Kapitelabschnitt 4.6 gewechselt werden darf. Wenn der Wechsel des Systemzustandes nicht erlaubt ist, wird die Funktion `handleToFailedSystemstate-Switching()` ausgeführt.

Sämtliche, in Kapitelabschnitt 10.3.1.8 beschriebenen, While-Schleifen tragen als Bedingung eine Abfrage der jeweiligen Variable des Systemzustandes, in welchem sie ausgeführt werden dürfen. Zusätzlich die einzelnen Threads eine eigene While-Schleife, welche den Zustand der Variable `off` für den Systemzustand *Aus* abfragt. Steht der Wert auf 0, wird die While-Schleife ausgeführt. Wechselt dieser Wert zu 1 bzw. damit der Systemzustand zu *Aus*, so werden diese While-Schleifen verlassen und sämtliche Threads terminieren. Dieses Prinzip wird auch für einzelne Funktionen genutzt. Die Trakektorienfindung und die Ausführung des *SCADE* Modells befinden sich in einer gemeinsamen While-Schleife im *SCADE*-Thread, welche prüft, ob der Systemzustand *Rennbetrieb* aktiv ist. Nur im Falle des Rennbetriebs wird das *SCADE* Modell ausgeführt. Genau so werden nur Regelungswerte an das Fahrzeug im Kommunikations-Thread versendet, wenn der Systemzustand *Rennbetrieb* aktiv ist. Den Wechsel der Systemzustände übernimmt die Systemsteuerung mittels gesendeter Kontrollbefehle. Für die Auswertung und Auslösung der Systemzustandswechsel ist der *ActionHandler* zuständig, welcher in Kapitelabschnitt 10.3.1.10 beschrieben wird. Weiteres zur Fehlerbehandlung kann dem Kapitelabschnitt 10.3.1.11 entnommen werden.

10.3.1.10 ActionHandler

Der *ActionHandler* ist für die Auswertung erhaltener Kontroll- und Fehlerdaten bzw. für die Auslösung der jeweiligen Systemzustandswechsel und der entsprechenden Handlungen zuständig. Hierzu wurde die Funktion `startActionHandler()` implementiert und wird in einer While-Schleife ausgeführt.

Die Funktion `startActionHandler()` enthält eine Prüfroutine, welche auf empfangene Kontrollbefehle der Systemsteuerung oder auf Fehlerdaten sämtlicher Komponenten reagiert und entsprechende Aktionen bzw. Funktionen ausführt.

Wird ein Kontrollbefehl von der Systemsteuerung empfangen, so wird dieser zuerst ausgewertet. Lautet der Wert 0, so wird das System mittels der Funktion `switchSystem-stateToStandby()` in den Zustand *Standby* überführt. Beim Wert 1 mit der Funktion `switchSystemstateToRace()` in den Zustand *Rennbetrieb*, beim Wert 2 mit der Funktion `switchSystemstateToInitialization()` in den Zustand *Initialisierung*, beim

Wert 3 mit der Funktion `switchSystemstateToInspection()` in den Zustand *Inspektion*, und beim Wert 4 mit der Funktion `switchSystemstateToOff()` in den Zustand *Aus* überführt. Werden Fehlerdaten empfangen, so wird zunächst analysiert, ob es sich um einen Not-Aus Befehl der Systemsteuerungssoftware oder einen Fehlercode handelt. Im Falle des Not-Aus Befehls, wird das System mit der Funktion `switchSystemstateToError()` in den sicheren Fehlerzustand überführt und ein Not-Halt des Fahrzeuges ausgeführt. Im Falle eines Fehlercodes, wird die Funktion `handleToReceivedFaultcode()` ausgeführt.

Um das System auch von der Control-Unit aus abschalten zu können, wurde in der Prüfroutine eine Tastaturabfrage implementiert. Sobald die Taste 'q' gedrückt wird, wird die Control-Unit - wie beim Systemzustand *Aus* - über die Funktion `shutdownSequence()` abgeschaltet. Die Funktion `shutdownSequence()` enthält eine Sequenz von Befehlen um das System in den Systemzustand *Aus* zu überführen, eine korrekte Deinitialisierung der Control-Unit vorzunehmen und das Programm zu beenden.

Weiteres zu den Systemzuständen und zur Fehlerbehandlung kann den Kapitelabschnitten 4.6 und 10.3.1.11 entnommen werden.

10.3.1.11 Fehlerbehandlung

Für die Fehlerbehandlung wurde das Modul *ErrorHandler* implementiert. Es enthält sämtliche Funktionen zum Handeln auf einen Fehlerfall. Die Funktion `handleToReceivedEmergencystop()` wurde beispielshaft aufgeführt.

```
/**
* @brief Handlung bei einem empfangenen Not-Aus Befehl.
*
* Die Funktion handleToReceivedEmergencyStop handelt bei einem
* empfangenen Not-Aus Befehl. Sie fuehrt einen Not-Halt des
* Fahrzeiges aus, gibt den Fehlercode auf der Konsole aus, sendet
* einen eigenen Fehlercode und beendet das System.
*/
void handleToReceivedEmergencystop() {
        int faultcode = 130056 + (parameterCU.carID * 100);
        sendBrakeSequence(&CarControl);
        sendEmergencyStop(&CarControl);
        printf("NetworkReceiver: Not-Aus vom Subsystem '
           Systemsteuerung' empfangen (#%6.6d).\n\n", faultcode);
        sendFaultcode(faultcode);
```

```
        shutdownFailedSystem();
}
```

Listing 10.1: Funktion `handleToReceivedEmergencystop()`

Die Funktion `handleToReceivedEmergencystop()` wird ausgeführt, wenn der *Action-Handler* (siehe Kapitelabschnitt 10.3.1.10) festgestellt hat, dass ein Not-Aus Befehl von der Systemsteuerung empfangen wurde. Er legt einen Fehlercode fest, führt eine Bremssequenz und einen Not-Halt des Fahrzeuges aus, gibt den Fehlercode inkl. einer Beschreibung auf der Konsole aus, senden den Fehlercode und schaltet das System ab.

Sämtliche Fehlerfälle innerhalb des Wrapper Codes haben eine eigene Funktion, welche - wie die Funktion `handleToReceivedEmergencystop()` - im eingetretenen Fehlerfall entsprechende Maßnahmen vornehmen. Bei einem nicht-sicherheitskritischen Fehler wird ausschließlich die Meldung über den aufgetretenen Fehler auf der Konsole ausgegeben und ein nicht-sicherheitskritischer Fehler gesendet. Bei einem sicherheitskritischen Fehler wird sofort ein Not-Halt des Fahrzeuges durchgeführt, eine Meldung auf der Konsole ausgegeben, der entsprechende Fehlercode an die anderen Systemkomponenten versendet und das System beendet. Sollten Fehlercodes oder Maßnahmen zu bestimmten Fehlerfällen angepasst werden müssen, so kann dies an einer zentralen Stelle - dem *ErrorHandler* - getan werden. Sämtliche Fehlercodes und der dazugehörigen Fehlerbeschreibungen sind in Tabelle 14.1 in Kapitelabschnitt 14.1 zu finden.

10.3.2 CarControl

Die CarControl bildet die Schnittstelle zwischen der Control-Unit und dem Fahrzeug. Sie empfängt Regelungswerte, Kontrollbefehle und den Not-Halt Befehl des Wrapper Codes und wandelt die digitalen Regelungswerte in analoge Spannung um. Die Software - welche auf der CarControl ausgeführt wird - ist in der Programmiersprache C geschrieben. Im folgenden werden einzelne Funktionalitäten der CarControl und deren Umsetzung beschrieben.

10.3.2.1 Nachrichtenempfang

Die CarControl ist in der Lage über eine serielle Schnittstelle Nachrichten zu empfangen. Nachrichten können hierbei Regelungswerte, Kontrollbefehle oder ein Not-Halt Signal sein. Die Nachrichten werden Byte-für-Byte auf die serielle Schnittstelle geschrieben und von der CarControl ausgelesen. Die Byte-Folge wird hierbei kontinuierlich von der seriellen Schnittstelle gelesen. Eine Endekennung (*line feed* und *carriage return*; kurz: LFCR)

beschreibt das Ende der Nachricht und signalisiert der CarControl, dass die Nachricht abgeschlossen ist und die folgenden Bytes - bis zur nächsten Endekennung - zur nächsten Nachricht gehören. Um diese Funktionalität realisieren zu können, wurde die Bibliothek *ub_uart* [STMe] verwendet.

Um der CarControl mitzuteilen, um was für eine Nachricht es sich handelt, wird zu Beginn der Nachricht ein Identifikator gesetzt. Diesem Identifikator folgen - bis zur Endekennung - die eigentlichen Daten. Die möglichen Nachrichten, deren Wertebereiche und eine Kurzbeschreibung sind in Tabelle 10.2 beschrieben. Empfangene Regelungswerte werden aus der Nachricht geparsed und anschließend in eine globale Variable geschrieben. Selbes gilt für weitere, empfangenen Nachrichten.

10.3.2.2 Betriebsmodi

In der ersten Version der CarControl wurde die Fernsteuerung mit dem Entwicklungsboard über Digital-Analog-Converter verbunden, um die abgetrennten Potentiometer zu ersetzen. In der zweiten Version der CarControl wurden die abgetrennten Potentiometer über Analog-Digital-Converter mit dem Entwicklungsboard verbunden und die Fernsteuerung wieder zusammengebaut. Somit wird das Auslesen der Potentiometer ermöglicht. Die CarControl liest die Potentiometer fortlaufend aus und speichert die sonst an die Fernsteuerung übertragenen Regelungswerte in globalen Variablen ab. Um diese Funktionalität realisieren zu können, wurde die Bibliothek *ub_adc_dma* [STMa] verwendet.

Für die Nutzung der Fernsteuerung stehen an dieser Stelle die ausgelesenen Regelungswerte der Potentiometer, die empfangenen Regelungswerte des Wrapper Codes (siehe Kapitelabschnitt 10.3.2.1) und die Regelungswerte der Neutralstellung zur Verfügung. Es wurden fünf Betriebsmodi implementiert, welche mittels Kontrollbefehl (siehe Kapitelabschnitt 10.3.2.1) oder über einen Taster (siehe Kapitelabschnitt 10.3.2.4) gewechselt werden können und im folgenden erläutert werden.

Deaktiviert
Im deaktivierten Modus werden die Regelungswerte der Neutralstellung für die Längs- und Querführung des Fahrzeuges am Digital-Analog-Converter bereitgestellt. Empfangene Regelungswerte des Wrapper Codes und ausgelesene Regelungswerte der Potentiometer haben keinen Einfluss auf das Fahrzeug. Das Fahrzeug bewegt sich nicht.

Vollständig manueller Modus
Im vollständig manuellen Modus werden die von den Potentiometern ausgelesenen Rege-

Nachrichtentyp	Nachricht / Wertebereich	Beschreibung
Kontrollbefehl	**C**	Herstellung bzw. Prüfung der Verbindung. Die CarControl antwortet mit einem CC.
	M#, wobei # dem Wert 0… 4 entspricht	Wechsel der Modi der CarControl.
	R	Auslösen eines SoftReset der CarControl.
Regelungswert	**S#**. wobei # dem Regelungswert zwischen 0-4095 entspricht	Übertragung des Regelungswertes für die Querführung.
	T#, wobei # dem Regelungswert zwischen 0-4095 entspricht	Übertragung des Regelungswertes für die Längsführung.
	XS#T#, wobei # dem Regelungswert zwischen 0… 4095 entspricht	Übertragung der Regelungswerte für die Quer- (S#) und Längsführung (T#).
Not-Halt	**N**	Auslösen eines Not-Halts. Längs- und Querführung werden auf Neutralposition gestellt und die CarControl in den sicheren Fehlerzustand überführt.

Tabelle 10.2: Auflistung möglicher Nachrichten, welche an die CarControl gesendet werden können.

lungswerte für die Längs- und Querführung des Fahrzeuges am Digital-Analog-Converter bereitgestellt. Empfangene Regelungswerte des Wrapper Codes haben keinen Einfluss auf das Fahrzeug. Das Fahrzeug bewegt sich - wie im originalen Zustand - den händischen Steuerungen über die Fernsteuerung entsprechend.

Hybrider Modus - Längsführung manuell
Im hybriden Mudus mit manueller Längsführung stehen die von dem Potentiometer ausgelesenen Regelungswerte für die Längsführung und die empfangenen Regelungswerte für die Querführung des Fahrzeuges am Digital-Analog-Converter bereitgestellt. Die jeweils ausgelesenen Regelungswerte für die Querführung und empfangenen Regelungswerte für die Längsführung haben keinen Einfluss auf das Fahrzeug. Das Fahrzeug wird entsprechend der empfangenen Regelungswerten für die Querführung gesteuert. Die Steuerung der Längsführung wird händisch über die Fernsteuerung durchgeführt.
Hybrider Modus - Querführung manuell
Im hybriden Mudus mit manueller Querführung stehen die von dem Potentiometer ausgelesenen Regelungswerte für die Querführung und die empfangenen Regelungswerte für die Längsführung des Fahrzeuges am Digital-Analog-Converter bereitgestellt. Die jeweils ausgelesenen Regelungswerte für die Längsführung und empfangenen Regelungswerte für die Querführung haben keinen Einfluss auf das Fahrzeug. Das Fahrzeug wird entsprechend der empfangenen Regelungswerten für die Längsführung beschleunigt oder gebremst. Die Steuerung über der Querführung wird händisch über die Fernsteuerung durchgeführt.

Autonomer Modus
Im autonoment Modus werden die empfangenen Regelungswerte des Wrapper Codes für die Längs- und Querführung des Fahrzeuges am Digital-Analog-Converter bereitgestellt. Die ausgelesenen Regelungswerte für die Längs- und Querführung haben keinen Einfluss auf das Fahrzeug. Das Fahrzeug wird entsprechend der empfangenen Regelungswerte automom von der Control-Unit gesteuert.

10.3.2.3 Fehlerzustand

Bei einem empfangenem Not-Halt Befehl oder einer abgebrochenen Verbindung - zwischen dem Rechner und der CarControl - wechselt die CarControl sofort in den sicheren Fehlerzustand. Hierzu werden die Regelungswerte für die Längs- und Querführung auf die Neutralstellung gestellt, sodass das Fahrzeug ausrollt, wenn es nicht nicht bereits steht, weil noch keine anderweitigen Regelungswerte vorlagen oder eine Bremssequenz

seitens des Wrapper Codes ausgeführt wurde. Ausgelesene oder empfangene Regelungswerte haben keine Auswirkung mehr auf das Fahrzeug, solange sich dies im sicheren Fehlerzustand befindet. Der sichere Fehlerzustand kann nur über einen manuellen Reset der CarControl durch den Administrator verlassen werden. Anschließend befindet sich die CarControl im Betriebsmodus *Deaktiviert*.

10.3.2.4 Tasterauswertung

Um die CarControl unabhängig vom Wrapper Code unter Kontrolle zu behalten, wurden die auf dem Entwicklungsboard integrierten Taster um weitere Taster auf der Lochrasterplatine erweitert, auf welchem das Entwicklungsboard befestigt ist. Insgesamt stehen dem Administrator vier Taster zur Verfügung, welche in Abbildung 10.11 abgebildet sind. Diese Taster werden fortlaufend ausgelesen. Um diese Funktionalität realisieren zu können, wurde die Bibliothek *ub_button* [STMb] verwendet. Der Taster *Not-Aus* löst einen Not-Halt des Fahrzeuges aus und überführt die CarControl in den sicheren Fehlerzustand (siehe Kapitelabschnitt 10.3.2.3). Der Taster *SoftReset* setzt die CarControl zurück. Hierbei werden sämtliche Variablen zurückgesetzt und in den Betriebsmodus *Deaktiviert* gewechselt. Der Taster *HardReset* führt einen vollständigen Reset des Entwicklungsboards durch. Es wird während des Tasterdruckes dem Mikrocontroller die Betriebsspannung entzogen. Hierbei handelt es sich um eine hardwareseitige Funktion, welche sofwareseitig nicht beeinflussbar ist. Der Taster *Modus-Wechsel* wechselt die Betriebsmodi in der Reihenfolge *Deaktiviert - vollständig Manuell - Hybrider Modus Längsführung - Hybrider Modus Querführung - Autonomer Modus*.

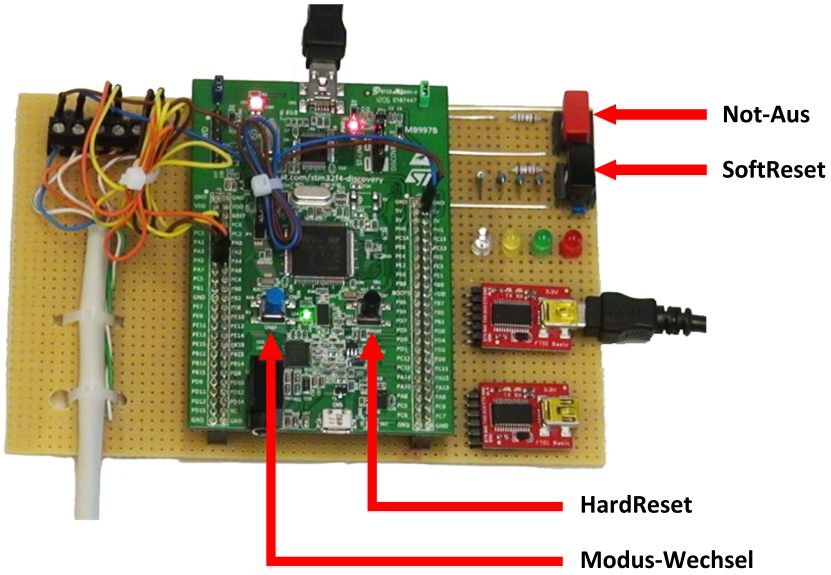


Abbildung 10.11: Taster der CarControl

10.3.2.5 Diagnose

Da das Entwicklungsboard über kein Display verfügt, kann der Administrator nur dessen Arbeitsweise am Fahrzeug erkennen. Ob die Regelungswerte korrekt empfangen wurden, ob der richtige Modus aktiv ist oder sich die CarControl im sicheren Fehlerzustand befindet, kann ohne Weiteres nicht erkannt werden. Hierzu wurden Diagnosemöglichkeiten entworfen und implementiert, welche die Überwachung der CarControl über den Administrator ermöglicht. Hierzu dienen integrierte LED's des Entwicklungsboards, zusätzlich angebrachte LED's auf der Lochrasterplatine und eine textuelle Überwachung über eine zusätzliche serielle Schnittstelle. Im folgenden wird die Funktionalität der Diagnose über die LED's und die serielle Schnittstelle erläutert.

Status LED's

Das Entwicklungsboard verfügt über vier integrierte und ansteuerbare LED's in den Farben grün, blau, rot und orange. Diese LED's können über die Bibliothek *ub_led* [STMd] angesteuert werden. Diese Bibliothek wurde erweitert um vier zusätzliche LED's ansteuern zu können, welche auf der Lochrasterplatine befestigt wird. Die Zusätzlichen LED's sind in den Fahrben gelb, grün und rot. Zusätzlich befindet sich eine IR-LED auf der Lochrasterplatine.

Die grüne LED blinkt beim Einschalten der CarControl bzw. nach einem *Hard-Reset*. Diese LED leuchtet permanent, wenn die Verbindung zwischen dem Wrapper Code und der CarControl hergestellt wurde.

Die blaue, rote und orangene LED signalisieren den akuell gewählten Betriebsmodus. Leutet nur die blaue LED, so befindet sich die CarControl im Modus *vollständig Manuell*. Leuchtet die rote LED, so befindet sich die CarControl im Modus *Hybrid - Längsführung manuell*. Leuchtet die orangene LED, so befindet sich die CarControl im Modus *Hybrid - Querführung manuell*. Leuchten die blaue, rote und orangene LED, so befindet sich die CarControl im Modus *Automom*. Weiteres zu den Betriebsmodi der CarControl kann dem Kapitelabschnitt 10.3.2.2 entnommen werden.

Blinken sämtliche LEDs des Entwicklungsboards, so befindet sich die CarControl im sicheren Fehlerzustand. Weiteres zum sicheren Fehlerzustand der CarControl kann dem Kapitelabschnitt 10.3.2.3 entnommen werden.

Debug-Schnittstelle

Da das Entwicklungsboard kein Display besitzt und die LED's nur bis zu einem gewissen Grad die Diagnose erlauben, wurde eine zweite serielle Schnittstelle erstellt. Über diese werden - unabhängig von der seriellen Hauptschnittstelle zum Nachrichtenempfang - dauerhaft Debug-Ausgaben gesendet oder Befehle zur Steuernung der Debug-Ausgaben empfangen. Hierzu kann das ein Tool zur Kommunikation über eine serielle Schnittstelle, wie *Hterm*, am Rechner genutzt werden.

Mögliche Debug-Ausgaben,welche über die zweite serielle Schnittstelle ausgelesen werden können, sind Einfache Statusausgaben und eine ausführlichere Ausgabe (Betriebsmodus, Status, empfangene Regelungswerte, ausgelesene Regelungswerte). Beispielausgaben können der Abbildung 10.12 entnommen werden.

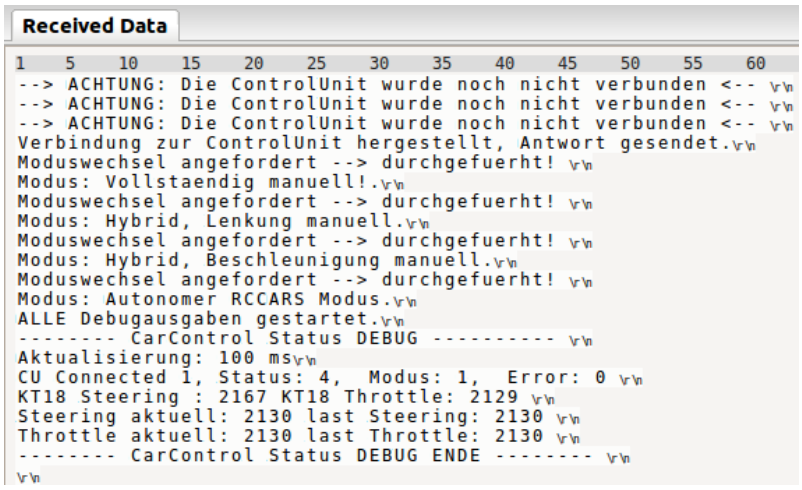


Abbildung 10.12: Beispielhafte Debug-Ausgabe der CarControl.

Um die Debug-Ausgaben einzustellen, werden Befehle - gefolgt von einer Endekennung (LFCR) - an die CarControl über die Debug-Schnittstelle gesendet. Mittels des Befehls ”A” können sämtliche Debugausgaben aktiviert werden. Der Befehl ”B” aktiviert nur die einfachen Statusausgaben. Der Befehl ”C” aktiviert die ausführtlichere Ausgabe. Der jeweilige Befehl als kleiner Buchstabe deakiviert die jeweilige Ausgabe. Ein erneutes Senden des jeweiligen Befehls reduziert die Sendegeschwindigkeit der Ausgabe.

10.3.2.6 Scheduling

In der Regel wird in der Mikrocontrollerprogrammierung die gesamte Funktionsroutine sequenziell in einer unendlichen While-Schleife ausgeführt. Das Problem hierbei ist jedoch, dass Funktionen erst ausgeführt werden, wenn die Funktionsroutine bei dieser angekommen ist. Nachrichten, welche über die serielle Schnittstelle empfangen werden, stellen kein Problem dar. Diese liegen so lange an, bis sie von der Schnittstelle gelesen werden. Sämtliche Funktionen werden mit der gleichen Priorität ausgeführt. Wenn jedoch mehrere Funktionalitäten ”abgearbeitet” werden müssen, bevor das Fahrzeug gesteuert bzw. die Regelungswerte über den Digital-Analog-Converter bereitgestellt werden können, stellt dies keine optimale Lösung dar. Um die wichtigen Funktionalitäten - welche das Fahrzeug direkt betreffen - höher zu priorisieren, wird ein Realzeitbetriebssystem auf der CarControl ausgeführt. Hierzu wird die Bibliothek *FreeRTOS* [STMc] verwendet. Mittels dieser Bibliothek lassen sich Funktionen in Gruppen bzw. Task’s zusammenfassen und dem jeweiligen Task eine Priorität zugewiesen werden. Der sich daraus ergebende Programmablauf wird in Abbildung 10.13 dargestellt. Im folgenden werden die Funktionsroutinen der Main-Datei und der einzelnen Tasks erläutert.

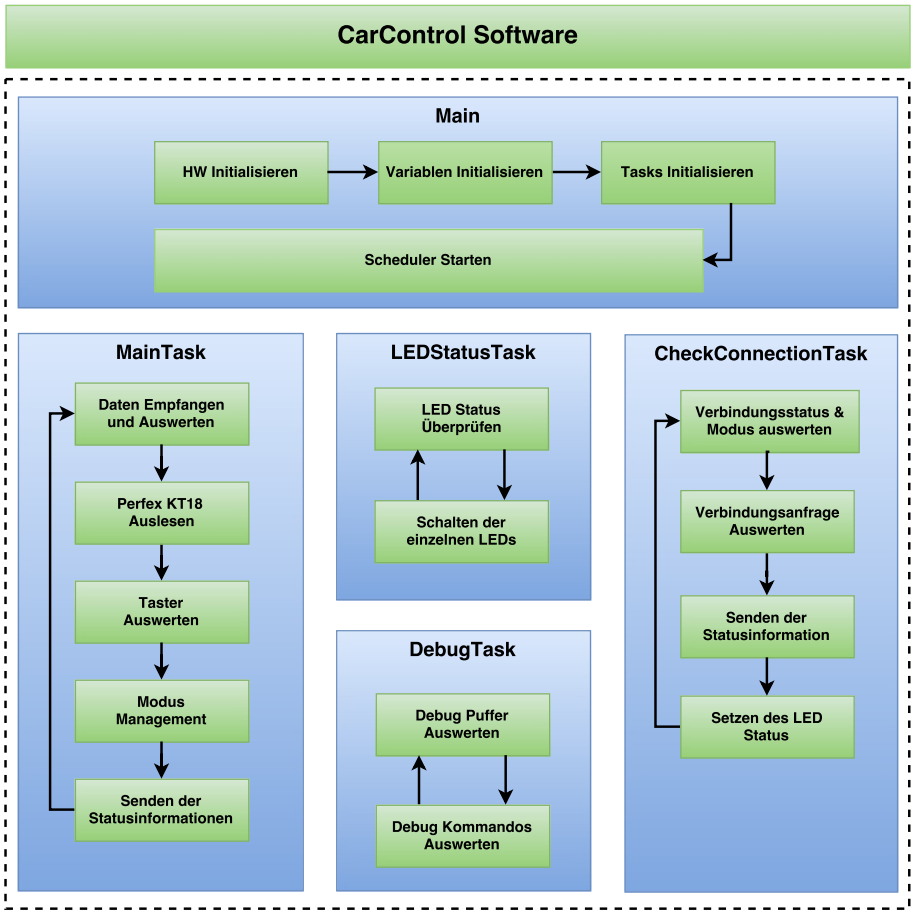


Abbildung 10.13: Programmablauf der Software der CarControl

Main-Datei

In der Main-Datei wird lediglich eine Main-Funktion ausgeführt. Diese initialisiert die Hardware-Schnittstellen, die benötigten Variablen und die Tasks. Die Priorisierung der Tasks wird ebenfalls in der Initialisierung vorgenommen und ist wie folgt: *MainTask*, *DebugTask*, *CheckConnectionTask* und *LEDStatusTask*, wobei dem *MainTask* die höchste und dem *LEDStatusTask* die niedrigste Priorität zugewiesen wird. Anschließend startet der Scheduler des Realzeitbetriebssystems und führt die Task's bzw. dessen Funktionen der Priorität nach aus. Das hier eingesetzte FreeRTOS setzt auf ein auf preemptiven festen Prioritäten basiertes Schedulingverfahren.

MainTask

Der *MainTask* liest die serielle Schnittstelle zwischen dem Wrapper Code und der CarControl und die aktuell am Analog-Digital-Converter anliegenden Regelungswerte der Potentiometer der Fernsteuerung aus. Anschließend werden die Taster (Not-Aus, Soft-Reset und Modus-Wechsel) ausgelesen. Je nach Eingabe bzw. Tastendruck reagiert das Modus Management. Dieser schreibt eine Debug-Ausgabe in einen Puffer, setzt Flags

für das Schalten der jeweiligen LED, wechselt den Betriebsmodus der CarControl und setzt die dem Betriebsmodus entsprechenden Regelungswerte an den Digital-Analog-Converter (siehe Kapitelabschnitt 10.3.2.2). Zum Schluss sendet der *MainTask* die Statusinformationen aus.

LEDStatusTask

Der LEDStatusTask sorgt für die optischen Feedbacks der CarControl. Er prüft den Status aller LED's, welcher von den anderen Tasks für jede LED aktualisiert wird, und schaltet dem Status jeder LED entsprechend diese ein, aus oder lässt diese in einem bestimmten Takt blinken.

DebugTask

Der DebugTask sortg für das textuelle Feedback, welches über die zusätzliche serielle Schnittstelle ausgelesen werden kann. Er prüft den Inhalt eines Puffers, welcher von den anderen Tasks zum Senden beschrieben wird, und sendet den Inhalt über die serielle Schnittstelle aus. Zusätzlich wertet dieser Task die Befehle aus, welche über die Debug-Schnittstelle gesendet werden können, um die Debug-Ausgaben (siehe Kapitelabschnitt 10.3.2.5) konfigurieren zu können.

CheckConnectionTask

Der CheckConnectionTask ist für die Auswertung von Verbindungsanfragen bzw. die Prüfung der Verbindung zuständig. Er prüft den Verbindungsstatus, wertet die Verbindungsanfrage aus, antwortet entsprechend und setzt ggf. den LED Status.

10.3.3 SCADE Modell

Dieser Kapitelabschnitt beschreibt den Aufbau und die Funktionsweise des *SCADE* Modells, welches durch den Wrapper Code (siehe Kapitelabschnitt 10.3.1.5) aufgerufen wird. Das Modell wird zur Zeit ein Mal je Millisekunde vom Wrapper getaktet .

Zu Beginn wurde ein Hauptoperator in *SCADE* erstellt, der die oberste Ebene des Modells repräsentiert und auf dem alle folgenden zu entwickelnden Operatoren aufbauen. Dieser Operator heißt *RootNode* und stellt die Schnittstelle zum Wrapper Code dar (siehe Kapitelabschnitt 10.3.1). Der *RootNode* Operator definiert somit alle Ein- und Ausgänge, die mit dem Wrapper Code kommunizieren können.
Im Weiteren stellt das aufgebaute Modell einen selbst definierten Sensor `racetrack` vom Datentypen `mapArray` zur Verfügung, welcher durch den Wrapper Code im Rahmen der Initialisierung mit den benötigten Rennstreckendaten beschrieben werden muss (siehe Kapitelabschnitt 10.3.1.1). Nähere Informationen zu der Schnittstelle Wrapper Code ↔

SCADE Modell sind in Kapitelabschnitt 8.4.4.2 beschrieben.

Abbildung 10.14 zeigt den *RootNode* Operator, welcher die oberste Ebene des Modells darstellt. Dieser Operator beinhaltet die State-Machine *OnOff*, welche kontrolliert, ob der Eingang *VehicleCheck* true ist. Der Eingang beschreibt, ob in den empfangenen Fahrzeugdaten eine Pose für das Fahrzeug der Control-Unit enthalten ist beziehungsweise das Fahrzeug von der Positionsbestimmung detektiert wurde. Ist dieser Eingang unwahr, wird das Modell in einen sicheren Fehlerzustand überführt, die Fehlerfunktion *LogError* aufgerufen und ein Fehlercode übergeben. Die Zusammensetzung der Fehlercodes, sowie eine vollständige Fehlercodeliste ist in Kapitelabschnitt 14.1 zu finden.
Bei erkanntem zugehörigen Fahrzeug befindet sich das Modell im Zustand *On*. Innerhalb dieses Zustands wurde die gesamte Funktionalität des *SCADE* Modells, welche in den Folgenden Abschnitten beschrieben wird, implementiert.

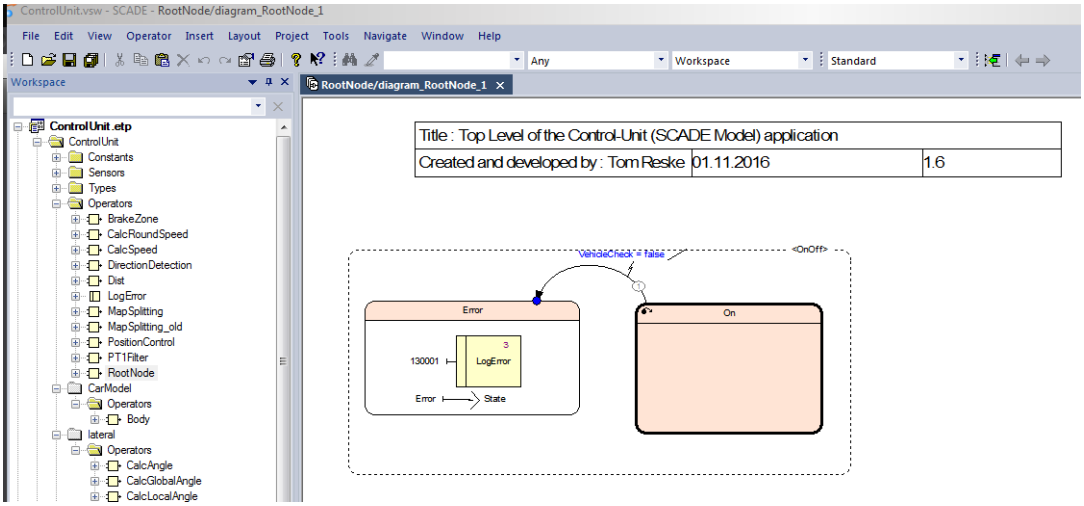


Abbildung 10.14: Der *RootNood* Operator ist der Operator in *SCADE*, der auf der obersten Ebene liegt und die Schnittstelle zum Wrapper Code bildet. Innerhalb des *On* Zustands befindet sich die gesamte weitere Funktionalität inklusive der Regelung des Modells.

10.3.3.1 Sicherheit

Innerhalb des *SCADE* Modells wurden bei der Entwicklung diverse Sicherheitsmechanismen implementiert. Dazu gehört der *LogError* Operator, welcher sich aus importiertem Programmcode des Wrapper Codes zusammen setzt. Der *LogError* Operator wird entweder in einem Fehlerzustand eines Zustandsautomaten (State-Machine) implementiert oder über die Funktion *Boolean-Activate* aktiviert. Beide Implementierungsverfahren führen dazu, dass bei Aktivierung des *LogError* Operators die extern implementierte Fehlerrou-

tine des Wrapper Codes aufgerufen wird. Zusätzlich wird bei jedem Aufruf ein individueller Fehlercode überreicht. Mehr zur Fehlerroutine des Wrapper Codes ist in Kapitelabschnitt 10.3.1.5 und 10.3.1.11 zu finden. Um Divisionen durch Null und so ungewolltes Verhalten zu vermeiden wird vor jeder Division der Teiler überprüft. Ist der Teiler gleich Null, so wird die Division nicht ausgeführt.

Zusammenfassend kann also festgehalten werden, dass sobald ein Fehler im *SCADE* Modell festgestellt wird, sich dieses in einen sicheren Zustand überführt, welcher nicht mehr verlassen werden kann. Im Weiteren wird die Fehlerroutine des Wrapper Codes durch den *LogError* Operator ausgeführt wobei darauf folgend der Wrapper Code die gesamte Control-Unit in einen sicheren Fehlerzustand versetzt und einen entsprechenden Fehlercode aussendet.

10.3.3.2 Positionsüberprüfung

In Abbildung 10.15 ist der *PositionControl* Operator dargestellt. Dieser ist in der Lage die Position des Fahrzeugs auf der Rennstrecke zu überprüfen. Dabei bietet er die Funktionalität anhand des Rennstreckenarrays festzustellen, ob eine Kollision mit der Rennstreckenbegrenzung vorliegt. Im Weiteren wird erkannt, ob das zugehörige Fahrzeug den befahrbaren Bereich verlassen hat, oder gar vollständig von der Rennstrecke abgewichen ist, also eine Indizierung außerhalb des Rennstreckenarrays vorliegt. Dieser Fall könnte entstehen, wenn das Fahrzeug von der Kamera noch detektiert wird, sich aber außerhalb der Rennstreckenkoordinaten befindet. Wenn das Fahrzeug nicht mehr detektiert wird, fängt dies die State-Machine im *RootNode* Operator ab und das Modell wechselt in einen sicheren Fehlerzustand *Error* (siehe Kapitelabschnitt 10.14).
Der *PositionControl* Operator ist somit in der Lage Fehlercodes - unter den eben genannten Umständen - zu erzeugen. Wird ein Fehler innerhalb des *PositionControl* Operators erzeugt, wird von dem *Indexing* Zustand in den *Error* Zustand gewechselt, welcher nicht mehr verlassen werden kann und so einen sicheren Zustand darstellt. Innerhalb des *Error* Zustands können je nach Fehlerart verschiedene Fehlercodes erzeugt werden. Mehr Informationen zu den jeweiligen Fehlercodes sind in Kapitelabschnitt 14.1 zu finden.

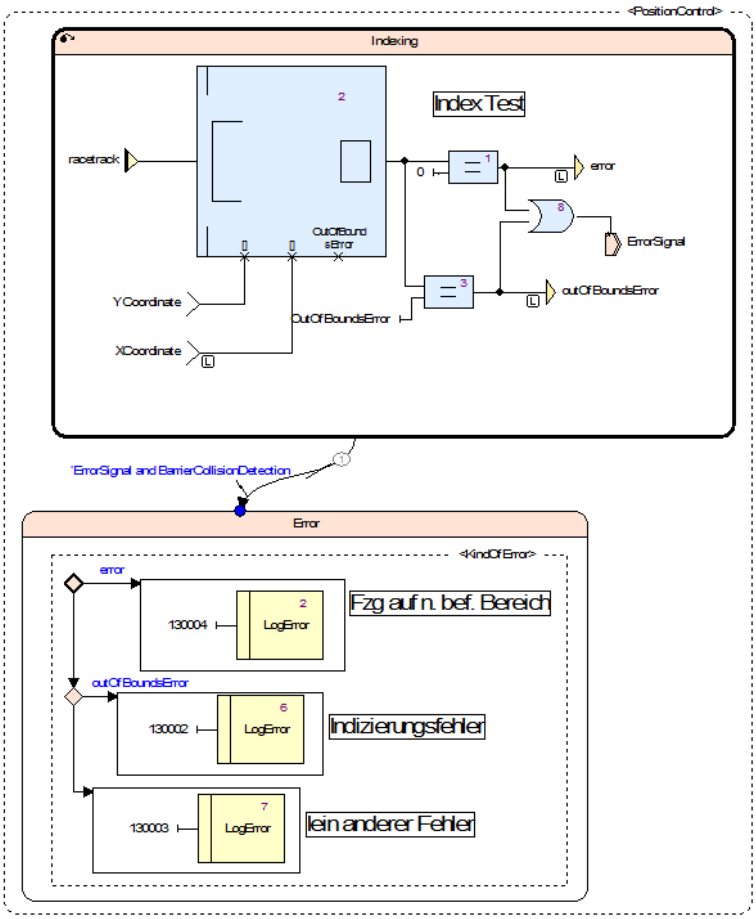


Abbildung 10.15: Der *PositionControl* Operator ist Teil des *RootNode* Operators und beinhaltet die Positionsüberprüfung des zugehörigen Fahrzeugs.

10.3.3.3 Bandenkollisionserkennung

In der aktuellen Ausbaustufe agiert die Bandenkollisionserkennung ebenso wie die Positionsüberprüfung. Es wird geprüft, ob sich die Koordinaten des Fahrzeugs auf dem befahrbaren beziehungsweise nicht befahrbaren Bereich befinden. Da sich das Fahrzeug durch mehr Fläche als durch den Punkt der Fahrzeugkoordinaten beschreibt, wurde die reale Fahrbahnbegrenzung um 5 cm erweitert. Die äußeren Kanten der vorderen Stoßstange des Fahrzeugs sind die Punkte, welche am weitesten vom Punkt, der durch die Koordinaten des Fahrzeugs beschrieben wird, entfernt sind. Die Entfernung liegt bei rund 4 cm. Zusätzlich wurde 1 cm Toleranz in die Entfernung einkalkuliert. So wird sichergestellt, dass auch bei rauschenden Koordinaten eine Bandenkollision detektiert werden kann. Diese Erweiterung ist ein gängiges Verfahren in der Robotik. Zukünftig sollte dennoch darüber nachgedacht werden, dieses Verfahren durch ein verbessertes mathematisches Verfahren zu ersetzen. Dadurch bedingt, dass im aktuellen Verfahren die Bande statisch verbreitert

wurde, wird teilweise eine Bandenkollision ausgelöst, auch wenn das Fahrzeug die Bande nicht berührt hat. Das hängt damit zusammen, dass die Seite des Fahrzeugs lediglich rund 1 cm von den Koordinatenpunkt des Fahrzeugs entfernt ist. Bei seitlicher Näherung an die Fahrbahnbegrenzung kann deshalb vorzeitig eine Kollision detektiert werden. Ein möglicher Lösungsansatz ist in Kapitelabschnitt 13.3.3.5 beschrieben.

10.3.3.4 Rennstreckeneinteilung

Die Rennstreckendatei wurde in einem Tabellenkalkulationsprogramm entwickelt, liegt im .odp Format vor und ist in Abbildung 10.16 zu sehen. Der befahrbare Bereich ist in grün dargestellt und wird von der virtuell verbreiterten Fahrbahnbegrenzung in gelb und der real existierenden Fahrbahnbegrenzung in orange vom nicht befahrbaren Bereich abgegrenzt. Die virtuelle, sowie die reale Fahrbahnbegrenzung stellt ebenfalls einen nicht befahrbaren Bereich dar. Die grauen Bereiche innerhalb des grünen Bereichs visualisieren Bremszonen. Im Weiteren sind Zahlen dargestellt, welche die Rennstreckensegmente nummerieren. Die Kurven sind mit schwarzen Linien von den restlichen Streckensegmenten abgegrenzt.

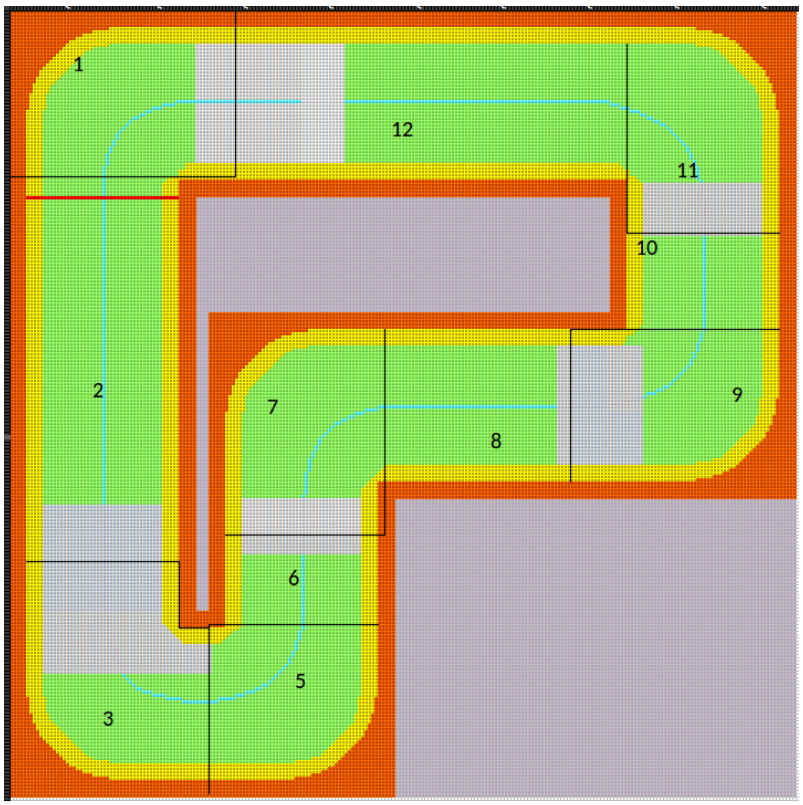


Abbildung 10.16: Die Grafik zeigt einen Screenshot der Rennstreckendatei im .odp Format.

Der Operator *MapSplitting* teilt die Rennstrecke in insgesamt elf verschiedene Rennstreckensegmente (`MapParts`) ein. Diese Abschnitte dienen dazu, um an einem anderem Ort im Modell die Längs- und Querregelung durchzuführen zu können.
Bei der Einteilung wird überprüft, ob sich die aktuelle Fahrzeugposition innerhalb eines bestimmten Koordinatenrahmens befindet. Die geraden Streckenabschnitte werden dabei durch gerade Zahlen beschrieben. Kurven durch ungerade Zahlen. Dadurch kann zu einem späteren Zeitpunkt durch eine einfache Modulo Operation des *MapPart*s festgestellt werden, ob sich das Fahrzeug auf einer Geraden oder in einer Kurve befindet.

10.3.3.5 Fahrtrichtungsüberprüfung

Die Fahrtrichtungsüberprüfung wird anhand der Fahrzeugkoordinaten und dem Rennstreckensegment (siehe Kapitelabschnitt 10.3.3.4) durchgeführt. Es wird kontrolliert, ob das Fahrzeug in die vorgegebene Richtung fährt (gegen den Uhrzeigersinn, siehe Anforderung SW2.1.5). Dazu wurde eine State-Machine erstellt, welche in einem Startzustand beginnt, um den Fahrzeugkoordinaten einen Takt Vorlauf zu geben. Dies ist wichtig, da immer die aktuelle mit der letzten X- oder Y-Koordinate verglichen wird. Im ersten Takt, wäre auf dem letzten Wert der jeweiligen Koordinaten nicht der reale letzte Wert, sondern ein zuvor definierter Wert enthalten. Im zweiten Takt des Modells wird direkt in den *DirectionControl* Zustand gewechselt. Innerhalb diesen Zustands wird anhand des Rennstreckensegments die aktuellen sowie die letzten Fahrzeugkoordinaten verglichen. Je nach Rennstreckensegment wird in der gleichnamigen bedingten Anweisung in die passende Bedingung gewechselt. Innerhalb dieser wird je nach Streckenabschnitt verglichen, ob die aktuelle X- oder Y-Koordinate größer oder kleiner als die letzte Koordinaten inklusive einer Toleranz ist. Beispielsweise muss auf dem Rennstreckensegmet 2 die aktuelle Y-Koordinate größer als die letzte Y-Koordinate inklusive einer Toleranz sein. Die Toleranz ist wichtig, da die Fahrzeugkoordinaten im Stand des Fahrzeugs Rauschen, welches vom Modell ohne Toleranz zu einem sicherheitskritischen Fehler führen kann. Das Modell interpretiert das Rauschen ansonsten als Fahren in die falsche Richtung.
Die Fahrtrichtungsüberprüfung funktioniert ausschließlich auf geraden Segmenten, denn innerhalb einer Kurve ist eine Veränderung der X- sowie Y-Koordinate in alle Richtungen möglich, ohne dass das Fahrzeug in die falsche Richtung fährt.

10.3.3.6 Geschwindigkeitsfeststellung

Innerhalb des *SCADE* Modells wird die Geschwindigkeit des Fahrzeugs festgestellt. Diese wird anhand von der Differenz der Zeitstempel der Fahrzeugkoordinaten und der zurückgelegten Strecke berechnet. Die Control-Unit erhält von der Positionsbestimmung

in jedem Fahrzeugdatenpaket einen Zeitstempel, welcher das Alter der Fahrzeugkoordinaten beschreibt. Anschließend wird von zwei aufeinander folgenden Zeitstempeln aus dem Fahrzeugdatenpaket die Differenz gebildet. Innerhalb des Modells wird der Zustand gewechselt, sobald erkannt wird, dass der aktuelle Zeitstempel sich von dem letzten Zeitstempel unterscheidet. Dadurch bedingt, dass lediglich alle circa 6,7 ms neue Fahrzeugdaten von der Positionsbestimmung empfangen werden (siehe Kapitelabschnitt 11.1) , ist nötig, die Geschwindigkeit nur dann zu berechnen, wenn sich die Fahrzeugdaten erneuert haben.Wenn das deutlich schneller taktende *SCADE* Modell (Taktfrequenz: 1 ms) in jedem Ausführungsschritt die Geschwindigkeit berechnet, wäre diese gleich Null, sobald in einem Step keine neuen Daten empfangen wurden. Die Geschwindigkeit wird durch die Wegdifferenz der Fahrzeugkoordinaten zu den letzten Fahrzeugkoordinaten über die Formel 10.6 gebildet.

$$s = \sqrt{(X - last'X)^2 + (Y - last'Y)^2} \tag{10.6}$$

Im Anschluss an die Wegdifferenzbildung wird anhand der Formel 10.7 die Geschwindigkeit festgestellt. Zusätzlich wird der Weg (s) von Zentimeter in die Einheit Meter und die Zeitdifferenz (t_{diff}) von Millisekunden in Sekunden konvertiert. Dies ist wichtig um die berechnete Geschwindigkeit (v) in der Einheit $\frac{m}{s}$ vorliegen zu haben.

$$v = \frac{\frac{s}{100}}{\frac{t_{diff}}{1000}} \tag{10.7}$$

Im Weiteren wird vor der Geschwindigkeitsberechnung anhand einer bedingten Anweisung überprüft, ob die Zeitdifferenz der Zeitstempel (t_{diff}) tatsächlich ungleich Null ist. Sollte die Zeitdifferenz Null werden, wird keine neue Geschwindigkeit berechnet.

Nach einem Takt, und somit einer neuen Berechnung der Geschwindigkeit, wird die berechnete Geschwindigkeit gehalten, bis neue Fahrzeugdaten mit einem neuem Zeitstempel von der Control-Unit empfangen werden.

Filter

Um eventuelles Rauschen der Koordinaten, und daraus resultierend ein massives Rauschen in der Geschwindigkeitsfeststellung zu glätten wird die zuvor berechnete Geschwindigkeit anhand eines PT1-Glieds mit dem Faktor 2 gefiltert. Dadurch entsteht ein geringes Maß an Totzeit, welches jedoch durch die sehr hohe Taktrate des Modells gegenüber

der Positionsbestimmung vernachlässigt werden kann. Die Filterung findet in jedem Takt statt, nicht nur, wenn neue Fahrzeugdaten mit einem neuem Zeitstempel von der Control-Unit empfangen werden.

10.3.3.7 Rundenzeitfestellung

In dem *CalcRoundSpeed* Operator bietet das Modell die Funktion zur Feststellung der Rundenzeit sowie der Durchschnittsgeschwindigkeit pro Runde. Die Rundezeitfeststellung sowie die Feststellung der Rundendurchschnittsgeschwindigkeit beginnen erst nachdem die Einführungsrunden (siehe Kapitelabschnitt 10.3.3.9) abgeschlossen wurden.

Die Funktionalität der Rundenzeitfeststellung wird durch eine State-Machine realisiert. Innerhalb dieser State-Machine wird beim ersten Überfahren eines fest definierten Punktes, welcher sich kurz vor der Ziellinie befindet der aktuelle Zeitstempel festgestellt. Im Anschluss wird in einen Zustand gewechselt, in dem auf den Ausgang der Rundenzeit und der Rundendurchschnittsgeschwindigkeit die entsprechenden Werte gelegt werden. In der ersten Runde nach der Einführungsrunde sind beide Ausgänge gleich Null. Wird erneut der fest definierte Punkte vor der Ziellinie überfahren, so wird anhand des aktuellen und des zuletzt gespeicherten Zeitstempels die Rundenzeit festgestellt. Über die Rundenzeit und der in der Konfigurationsdatei der Control-Unit (siehe Kapitelabschnitt 14.5) definierten Länge der Zieltrajektorie wird schließlich die Rundendurchschnittsgeschwindigkeit über Formel 10.8 festgestellt.

$$v_{average} = \frac{s_{trajectory}}{t_{diff}} \tag{10.8}$$

10.3.3.8 Bremszonen

Um die Durchschnittsgeschwindigkeit zu Erhöhen wurden auf der Rennstrecke virtuelle Bremszonen im *SCADE* Modell implementiert. Durch eine Verzögerung vor den Kurven kann die angestrebte Geschwindigkeit auf den Geraden deutlich erhöht und trotzdem eine Kurve erfolgreich durchfahren werden. Diese Bremszonen sorgen dafür, dass sobald sich die Fahrzeugkoordinaten innerhalb dieses Bereichs befinden der minimal mögliche Throttle-Stellwert gesetzt wird. Der minimale Throttle-Stellwert sorgt während einer Vorwärtsfahrt für eine Vollbremsung. Würde das Fahrzeug stehen, so würde dies für eine Rückwärtsfahrt sorgen (mehr dazu im Ausblick in Kapitelabschnitt 13.3.3).
Die Bremszonen sind zum aktuellen Entwicklungsstand statisch im Modell definiert. Die Größe und Lage der Bremszonen wurden zuvor in diversen Testfahrten ermittelt und evaluiert. Die aktuellen Bremszonen sind in Abbildung 10.16 in Kapitelabschnitt 10.3.3.4

innerhalb des befahrbaren Bereichs in grau dargestellt.

10.3.3.9 Einführungsrunden

Das *SCADE* Modell bietet innerhalb des *CalcRoundSpeed* Operators (siehe Kapitelabschnitt 10.3.3.7) die Funktion, dass beim Start des Systems das Fahrzeug mit einer oder mehreren Einführungsrunden beginnt. Dazu wird eine State-Machine genutzt, welche einen boolschen Ausgangswert namens `WarmUp` besitzt, welcher wahr wird, sobald sich das Fahrzeug darin befindet.
Dadurch bedingt, dass aktuell die Bremszonen statisch und nicht dynamisch implementiert sind (siehe Kapitelabschnitt 10.3.3.8) wird mindestens eine Einführungsrunde benötigt, um das Fahrzeug auf Geschwindigkeit zu bringen, bevor die Bremszonen aktiv sind. Innerhalb der Einführungsrunden wird das Fahrzeug mit einer geringeren Zielgeschwindigkeit geregelt und gleichzeitig die Bremszonen deaktiviert. Die Anzahl der Einführungsrunden kann in der Konfigurationsdatei der Control-Unit eingestellt werden (siehe Kapitelabschnitt 14.5). Es wird immer mindestens eine Einführungsrunde gefahren. Die Anzahl der Einführungsrunden, die in der Konfigurationsdatei eingetragen werden sind somit immer die Anzahl der Runden, die zusätzlich zur ersten Einführungsrunde gefahren werden. Die Einführungsrunden simulieren sogenannte *Warm-Up* Runden sowie einen fliegenden Start.

Das Modell startet in der ersten Einführungsrunde (Runde 0) in einem Zustand, in dem die Rundenzeit, sowie die Rundendurchschnittsgeschwindigkeit statisch auf 0 gesetzt werden. Der Zustand wird gewechselt, sobald das Fahrzeug kurz vor der Ziellinie einen definierten Punkt das erste Mal überfährt. Bei jedem Überfahren dieses fest definierten Punktes wird geprüft, ob die in der Konfigurationsdatei definierte Anzahl an Einführungsrunden erreicht wurde. Wurden diese nicht erreicht, so wird in einen Inkrementierungszustand gewechselt, welcher den Zähler der gefahrenen Einführungsrunden hoch zählt. Im Anschluss wird in den Startzustand zurück gewechselt. Ist beim Überfahren des fest definierten Punktes vor der Ziellinie die gewünschte Anzahl an Einführungsrunden erreicht, so beginnt die Zeit- und Geschwindigkeitsmessung der Rundenzeitfeststellung (siehe Kapitelabschnitt 10.3.3.7).

10.3.3.10 Längsregelung

Die Projektgruppe entschied sich dafür die Längsregelung durch eine Kennfeldsteuerung mit einer Residualregelung zu realisieren. Das bedeutet, dass für jede angezielte Geschwindigkeit ein zuvor evaluierter Gasstellwert (Throttle-Stellwert) (siehe Tabelle 7.1

in Kapitelabschnitt 7.1.1.1) aus dem Kennfeld bezogen wird. Im Anschluss wird durch die zuvor genannte Geschwindigkeitsberechnung (siehe Kapitelabschnitt 10.3.3.6) überprüft, ob die Zielgeschwindigkeit erreicht wurde. Wurde diese nicht erreicht, so passt die Residualregelung den nötigen Stellwert an.

Kennfeldsteuerung

Über die Eingänge $TargetSpeed$, $TargetSpeed_{Curve}$, $TargetSpeed_{WarmUp}$ und $TargetSpeed_{WarmUpCurve}$, welche im Config File definiert werden, kann in das *SCADE* Modell eine Zielgeschwindigkeit gereicht werden, welche auf den Gerade und den Kurven, während und außerhalb des Warm-Ups, angesteuert werden sollen. Diese Zielgeschwindigkeiten sind beschrieben in $\frac{m}{s}$ und dürfen eine Nachkommestelle besitzen. Besteht die Zielgeschwindigkeit aus mehr als einer Nachkommastelle, so werden diese vernachlässigt.
Der jeweilige Zielgeschwindigkeitswert ist abhängig vom Streckensegment und ob sich das Modell im Warm-Up befindet. Anhand der jeweiligen Zielgeschwindigkeit wird dazu passend ein entsprechender Throttle-Stellwert aus dem Kennfeld abgefragt. Das Kennfeld besteht aus 35 Zeilen und zwei Spalten. Eine Zeile enthält dabei die Zielgeschwindigkeit und den passenden Throttle-Stellwert in einer Abstufung von jeweils 0,1 $\frac{m}{s}$.
Um bei einem Eingangswert von beispielsweise 1,1 $\frac{m}{s}$ den passenden Stellwert aus dem Kennfeld ziehen zu können wird die Formel 10.9 genutzt. Das Kennfeld ist so aufgebaut, dass die nachstehende Formel den korrekten Stellwert ausgibt.

$$row = (Speed_{target} - 0,1) \cdot 10,0 \tag{10.9}$$
$$row = (1,1 - 0,1) \cdot 10,0 \tag{10.10}$$
$$row = (1,0) \cdot 10,0 \tag{10.11}$$
$$row = 10,0 \tag{10.12}$$
$$\tag{10.13}$$

Das Kennfeld beinhaltet Stellwerte von 0,1 bis 3,5 $\frac{m}{s}$. Sollte eine Zielgeschwindigkeit außerhalb des zuvor genannten Rahmens angegeben werden, so wird außerhalb des definierten Kennfeld-Arrays indiziert, was einen Fehler erzeugt. Das *ErrorSignal* wird wahr und es wird in den Fehlerzustand *Error* gewechselt. Der Fehlerzustand zieht das Aufrufen der *SCADE* internen Fehlerroutine nach sich (*LogError*). Der Fehlerzustand kann nicht mehr verlassen werden.
An dieser Stelle soll jedoch angemerkt werden, dass das Kennfeld zum aktuellen Entwicklungsstand nur teilweise gefüllt wurde. Es kann aktuell nicht jede Geschwindigkeit innerhalb des genannten Rahmens, die gewünscht ist, eingetragen werden. Dazu müsste

das Kennfeld in Zukunft weiter evaluiert und gefüllt werden. Die grundlegende Funktionalität ist jedoch gegeben.

PID-Regler

Nachdem durch die jeweilige Zielgeschwindigkeit aus dem Kennfeld ein entsprechender Throttle-Stellwert abgefragt wurde, wird kontinuierlich geprüft, wie groß die Differenz zwischen Wunschgeschwindigkeit und realer Geschwindigkeit des Fahrzeugs ist. Die Differenz stellt den Regelfehler ($speed_{error}$) dar. Der Regelfehler ist die Grundlage des nachfolgenden PID-Reglers. Der PID Regler ist in Abbildung 10.17 dargestellt.

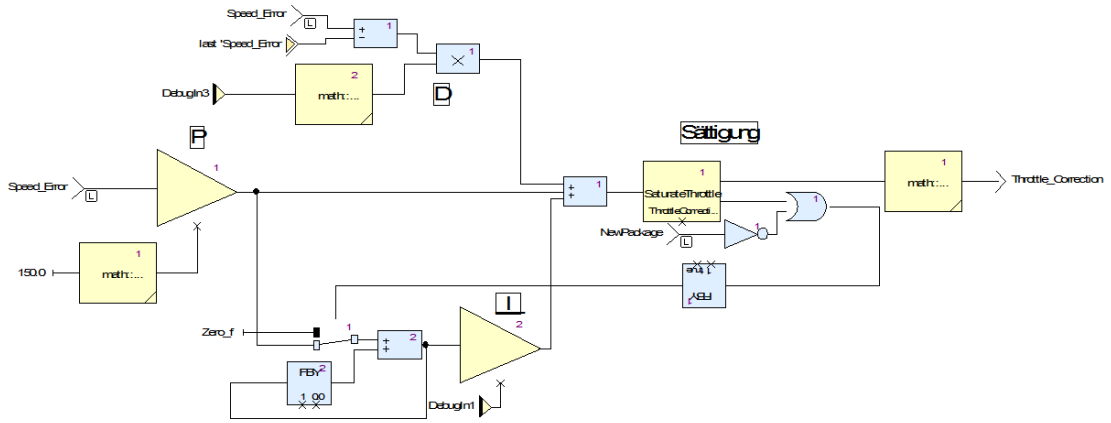


Abbildung 10.17: Die Grafik zeigt einen Screenshot des Geschwindigkeitsreglers mit einem P-Anteil ($K_p = 150.0$, $K_d = 15.0$ und einem I-Anteil ($K_i = 0.5$)).

Der Regler besitzt zudem eine Sättigungsfunktion und Einflussbegrenzungsfunktion, welche dafür sorgt, dass der Regler eine maximale Korrektur am Throttle-Stellwert von aktuell $\pm$ 150 zulässt. Dies sorgt dafür, dass insbesondere der I-Anteil des Reglers den Stellwert nicht zu hoch beziehungsweise zu niedrig korrigiert und so ein unerwünschtes Verhalten des Fahrzeugs auftritt. Ein unerwünschtes Verhalten äußert sich in diesem Fall beispielsweise durch eine hohe Trägheit des Reglers oder durch zu einflussstarke Korrekturen am Stellwert. Im Weiteren sorgt die Sättigungsfunktion dafür, dass sich ein Regelfehler nur dann aufintegrieren kann, wenn ein neues Fahrzeugdatenpaket empfangen wird. Die Abschaltung der Integration ohne neue Fahrzeugdaten verhindert ein Übersteuern des Regelfehlers bedingt durch den I-Anteil und der deutlich höheren Taktung von *SCADE* im Vergleich zum Erhalt neuer Fahrzeugdaten.

10.3.3.11 Querregelung

Im *SteeringController* des *SCADE* Modells befinden sich alle Modelle, welche für die Querführung benötigt werden. Die Querregelung wurde ebenso die Längsregelung durch eine Kennfeldsteuerung mit einer Residualregelung realisiert. Das bedeutet, dass für jede Kurve ein zuvor evaluierter Lenkstellwert (Steering-Stellwert) aus dem Kennfeld gezogen wird. Im Anschluss wird die Abweichung der Fahrzeugposition und des Ausrichtungswinkels des Fahrzeugs zur Zieltrajektorie und zur Zielausrichtung abgeglichen. Die Abweichung wird durch die Residualregelung im Anschluss ausgeglichen.

10.3.3.12 Zieltrajektorienfeststellung

Der Wrapper-Code bestimmt ein bis zwei mögliche Trajektorienpunkte, welche vom Fahrzeug angesteuert werden könnten. Diese Funktion wurde bereits in Kapitelabschnitt 10.3.1.4 beschrieben. Das *SCADE* Modell erhält diese Punkte über einen Eingang (Array) und bestimmt im Operator *FindTrajectory* letztendlich den anzusteuernden Zieltrajektorienpunkt.

Zu Beginn wird festgestellt, wie viele mögliche Trajektorienpunkte durch den Wrapper-Code festgestellt wurden. Je nachdem wie viele mögliche Zielpunkte gefunden wurden, wird in verschiedene Zustände im Modell gewechselt.
Wird genau ein möglicher Zieltrajektorienpunkt gefunden, wird dieser direkt als Zielpunkt ausgewählt und die zugehörigen X und Y Werte zur weiteren Verarbeitung gespeichert.
Erhält das Modell zwei mögliche Zieltrajektorienpunkte, muss untersucht werden, welcher dieser Punkte als letztendlicher Zielpunkt gesetzt wird. Um dies bestimmen zu können, muss betrachtet werden auf welchem Streckensegment sich das Fahrzeug befindet. Durch eine If-Bedingung wird je nach Streckensegment, auf dem sich das Fahrzeug befindet, in die entsprechende Bedingung gewechselt. Innerhalb der Bedingungen werden durch Logikoperationen die zwei möglichen Punkte miteinander verglichen. Dabei muss beispielsweise jeweils ein Punkt eine größere oder kleinere X oder Y Koordinate als der andere Punkt aufweisen um der Zielpunkt zu sein. Als Beispiel wird der Streckenabschnitt Zwei angenommen (zur Veranschaulichung siehe Kapitelabschnitt 10.3.3.4 sowie Abbildung 10.16). Da gegen den Uhrzeigersinn gefahren wird, muss innerhalb dieses Abschnitts der letztendliche Zielpunkt eine größere Y-Koordinate aufweisen als der zweite mögliche Zielpunkt. Ist diese Bedingung erfüllt, wird der entsprechende Punkt zum Zieltrajektorienpunkt und die zugehörigen Koordinaten für weitere Verarbeitungsschritte gespeichert.

Kann zum jeweiligen Streckensegment keine Zieltrajektorie ausgemacht werden, wird in einen sicheren Fehlerzustand gewechselt, welcher nicht mehr verlassen werden kann und die Fehlerroutine aufgerufen. Die Fehlerroutine wird ebenfalls aufgerufen, falls auf einen nicht indizierbaren Bereich des Eingangs zugegriffen wird, der die möglichen Zieltrajektorienpunkte beinhaltet.

Im besten Fall benötigt die Feststellung der Koordinaten des Zieltrajektorienpunkts zwei Takte, also zwei Ausführungen des Modells, da der Zustand gewechselt werden muss. Im schlimmsten Fall benötigt das Modell drei Takte um die Koordinaten festzustellen. Diese benötigte Taktzeit muss an anderen Stellen beachtet werden.

10.3.3.13 Winkelfehler

Durch die Positionsbestimmung erhält das *SCADE* Modell den aktuellen Ausrichtungswinkel (ϕ) des Fahrzeugs. Anhand der aktuellen Koordinaten und den Koordinaten des Zieltrajektorienpunkts kann der Winkelfehler berechnet werden. Der Winkelfehler stellt, ähnlich wie bei der Längsregelung den Regelfehler dar, welcher durch die Residualregelung ausgeglichen wird.
Wie bereits im Kapitelabschnitt 10.3.3.12 beschrieben, benötigen die Feststellung der Koordinaten des Zieltrajektorienpunkts im worst-case drei Takte. Um bei der Winkelfehlerberechnung korrekt arbeiten zu können, werden kontinuierlich die Fahrzeugkoordinaten, welche zur Berechnung herangezogen werden, um drei Takte verzögert, damit die Winkelfehlerberechnung zeitlich mit den Zieltrajektorienkoordinaten übereinstimmen. Dies verzögert die gesamte Berechnung des Winkelfehlers um insgesamt drei Takte. Durch die hohe Taktrate des Modells ist der dadurch entstehende zeitliche Verlust jedoch zu vernachlässigen.

Innerhalb des *CalcAngle* Operators wird zuerst der lokale Winkelfehler berechnet (*CalcLocalAngle* Operator). Dieser Winkelfehler betrachtet nicht die aktuelle Ausrichtung des Fahrzeugs, sondern geht von einem lokalen Koordinatensystem aus, in dem sich das Fahrzeug befindet. Dabei hat das Fahrzeug immer den Ausrichtungswinkel 0. Die Grafik 10.18 veranschaulicht dies. Im lokalen System steht das Fahrzeug immer in Richtung der X-Achse beziehungsweise wird der Winkelfehler immer zur Ausrichtung der X-Achse gebildet. Erst der globale Winkelfehler beschreibt den tatsächlichen Ausrichtungsfehler des Fahrzeugs zur Zieltrajektorie. Die Drehrichtung des Fahrzeugs im Hinblick auf den Ausrichtungswinkel ist im Uhrzeigersinn.

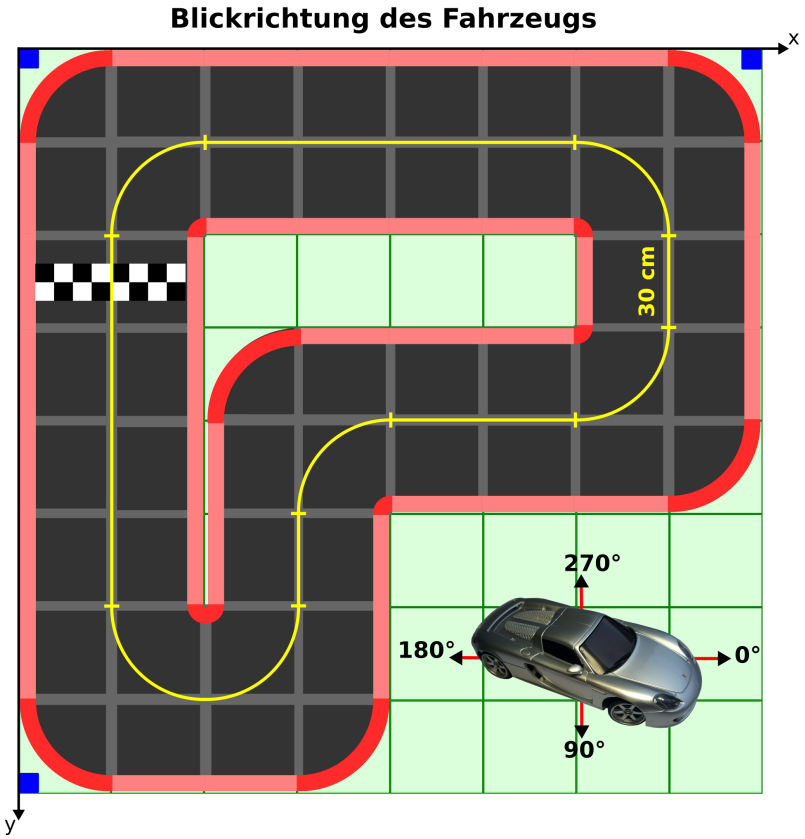


Abbildung 10.18: Die Grafik zeigt die Rennstrecke mit Koordinatensystem, sowie die Ausrichtungswinkel, welche bei diversen Blickwinkeln des Fahrzeugs entstehen.

Zur Berechnung des lokalen Winkelfehlers werden im Anschluss die realen Fahrzeugkoordinaten und die Koordinaten des Zieltrajektorienpunkts verwendet und über Formel 10.14 der lokale Winkelfehler bestimmt.

$$\phi_{localError} = atan2((Y_{Trajectory} - Y), (X_{Trajectory} - X)) \tag{10.14}$$

$$\phi_{localError} = \frac{atan2(Y,X) \cdot 180}{\pi} \tag{10.15}$$

Anhand des lokalen Winkelfehlers kann im Anschluss der globale Winkelfehler in Abhängigkeit der Ausrichtung des Fahrzeugs gebildet werden. Formel 10.16 zeigt diese Berechnung, allerdings wird der zu korrigierende Fehler immer in Richtung der Drehachse des Fahrzeugs ausgegeben.

$$\phi_{globalError} = (360 - \phi) + \phi_{localError} \tag{10.16}$$

Aus diesem Grund wird im Anschluss überprüft, ob der zu korrigierende Winkel über 180 Grad beträgt. Ist dies der Fall, so wird das Ergebnis aus Formel 10.16 entsprechend verändert. Durch diese Überprüfung korrigiert das Fahrzeug den Winkel in die Richtung, zu der am wenigsten korrigiert werden muss. Formel 10.17 fasst das Mapping von $\phi_{globalError}$ zusammen.

$$\phi_{globalError} = \begin{cases} 360 - \phi_{globalError}, & \text{wenn } \phi_{globalError} > 180 \\ \phi_{globalError}, & \text{sonst} \end{cases} \tag{10.17}$$

Kennfeldsteuerung

Ähnlich wie bei der Kennfeldsteuerung, die im Kapitelabschnitt 10.3.3.10 beschrieben wurde, wird im Operator *IndexingCharacteristikDiagram*, abhängig von der jeweiligen Kurve, ein vorher ermittelter Steering-Stellwert aus einem Kennfeld gezogen, welcher von der Neutralstellung der Lenkung korrigiert werden soll. Somit wird der aus dem Kennfeld ermittelte Wert dazu genutzt, die Neutralposition der Lenkung zu verändern. Bei jedem Zugriff auf das Kennfeld-Array wird überprüft, ob auf einen nicht indizierbaren Bereich zugegriffen wird. Sollte entsprechend ein Fehler auftreten, wird die Fehlerroutine (siehe Kapitelabschnitt 10.3.3.1) aufgerufen.

PID-Regler

Neben der Kennfeldsteuerung wurde eine Residualregelung implementiert, welche sich aus einem P, I und D Anteil zusammen setzt. Als Regelfehler gilt der in Kapitelabschnitt 10.3.3.13 festgestellt globale Winkelfehler. Zum aktuellen Entwicklungszeitpunkt wird der I-Anteil des Reglers nicht verwendet ($k_I = 0$), da der I-Anteil in ersten Versuchen dazu führte, dass der Regler zu träge wurde und ein Lenkwert, zum Beispiel nach dem Ende einer Kurve, zu lange braucht, um abgebaut zu werden.
Der D-Anteil bewährte sich, da er die Reaktionsfähigkeit des Reglers deutlich steigerte. Im Weiteren besitzt der Regler eine Sättigungsfunktion, welche dafür sorgt, dass der Lenkregler nur innerhalb fest definierter Grenzen den Winkelfehler verändern kann. Im Weiteren gilt der I-Anteil des Reglers gesättigt, sobald die zuvor genannten Grenzen überschritten wurden oder sobald im aktuellen Ausführungstakt des Modells keine neuen Fahrzeugdaten empfangen wurden.

Mappen des Winkels

Der zuvor beschriebene Lenkregler korrigiert lediglich den Winkelfehler in Grad. Diese Gradzahl wird im *MappingAngleToSteering* Operator in einen Steering-Stellwert über-

tragen. Das Mappen findet linear statt. Dazu wird die bekannte Funktion einer linearen Geraden (siehe Formel 10.18) genutzt.

$$y = m \cdot x + b \tag{10.18}$$

In diese Formel 10.18 wird entsprechend der maximale Lenkwinkel von 30 Grad zu jeder Seite, der Stellwert für die Steering-Neutralstellung und der Stellwert für den maximalen Lenkeinschlag, den man von der Neutralstellung aus einlenken kann, eingetragen.

$$Steering = \frac{Lenkwert_{Max}}{Lenkwinkel_{Max}} \cdot Lenkwinkel_{Soll} + Neutralstellung \tag{10.19}$$

In die Formel 10.19 werden dann die evaluierten Werte eingesetzt.

$$Steering = \frac{1850}{30} \cdot Lenkwinkel_{Soll} + 2130 \tag{10.20}$$

Aus Formel 10.20 ergibt sich dann Formel 10.21.

$$Steering = 61,67 \cdot Lenkwinkel_{soll} + 2130 \tag{10.21}$$

Filtern

Nach den Berechnungen wird der Kennfeldwert sowie der durch die Residualregelung berechnete Korrekturwert addiert und Low- und Highpass gefiltert. Die Grenzen bestehen dabei jeweils aus dem minimalen sowie maximalen Steering-Wert für einen Links- / Rechtseinschlag. Die ermittelten maximalen Lenkwerte in Grad und Regelungswerte können dem Kapitelabschnitt 7.1.1.3 entnommen werden.

10.3.3.14 Fahrzeugmodell

Bedingt durch Zeitmangel wurde erst zum Ende der Projektgruppe ein Fahrzeugmodell entwickelt. Dieses Fahrzeugmodell wurde in *SCADE* implementiert, jedoch nicht weiter verwendet. Die Funktion des Fahrzeugmodells wird im Folgenden beschrieben.
Nach Rücksprache mit den Universitäten in Uppsala und Zürich wurde das Unicyle-Modell näher betrachtet. Es zeichnet sich durch seine Einfachheit aus und eignet sich für langsame Geschwindigkeiten. Die Dynamik des Modell lässt sich folgendermaßen darstellen:

$$\dot{X} = v\cos(\phi) \tag{10.22}$$

$$\dot{Y} = v\sin(\phi) \tag{10.23}$$

$$\dot{\phi} = \frac{v\delta}{L} \tag{10.24}$$

$$\dot{v} = \frac{1}{m}((C_{m1} - C_{m2}v)D - C_r - C_d v^2) \tag{10.25}$$

Die Geschwindigkeit in X-Richtung $\dot{X}$ wird mittels der aktuellen Geschwindigkeit v und der Ausrichtung des Fahrzeugs ϕ modelliert. Äquivalent dazu kann auch $\dot{Y}$ dargestellt werden. Durch Integration kann der Ausrichtungswinkel mittels der Geschwindigkeit v, dem Einlenkwinkel δ und der Radstandlänge L bestimmt werden. Um die Beschleunigung der Fahrzeugs zu modellieren, muss zum Einen die Masse m des Fahrzeugs berücksichtigt werden. Darüber hinaus müssen die Motorkonstanten C_{m1} und C_{m2} identifiziert werden. Der Parameter D gibt den Tastgrad des Motors wieder und liegt typischerweise im geschlossenen Intervall [0,1]. Zum Anderen werden Roll- und Motorwiderstand mittels der Parameter C_r und C_d berücksichtigt.

10.4 Systemsteuerungssoftware

Um eine zentrale Steuerung und Überwachung des Systembetriebs zu ermöglichen, wird eine Softwarekomponente benötigt, welche dem Administrator des Systems eine Benutzeroberfläche zur Verfügung stellt, die alle erforderlichen Bedienelemente besitzt und alle relevanten Systeminformationen zusammenführt und ausgibt. Insbesondere im Hinblick auf die Erweiterbarkeit des Systems ist dies besonders wichtig, da die Positionsbestimmung und die bis zu 16 Instanzen der Control-Unit eigenständige Softwarekomponenten besitzen, welche ggf. auf unterschiedliche Hardwaresysteme verteilt sind.
Obwohl die Positionsbestimmungssoftware und die Softwarekomponenten der Control-Units eigene Funktionen zur Fehlererkennung besitzen (siehe Kapitelabschnitte 10.1.7.2, 10.1.7.3, 10.3.1.11, 10.3.3.1), ist eine unabhängige automatische Überprüfung durch die Systemsteuerungssoftware sicherheitsrelevant. Dies gilt insbesondere für das Einhalten der Realzeitanforderungen (SW1.4 und SW2.3) und für die Erkennung eines vollständigen Ausfalls der Regelungssoftware einer Control-Unit. Erkennt die Systemsteuerungssoftware einen sicherheitskritischen Fehler im System, sendet sie automatisch einen Not-Aus-Befehl in das Netzwerk.
Aufgrund des begrenzten Zeitrahmens und dem vorhandene Erfahrungsvorsprung im Bereich der Entwicklung von Java-Anwendungen, wurde der wesentliche Anteil der Sys-

temsteuerungssoftware in Java entwickelt. Um die Schnittstelle zum in C implementierten NetworkMessenger (siehe Kapitelabschnitt 10.2.2) zu realisieren wurde die Software-Bibliothek JNA (Java Native Access) [JNA] verwendet.
Zur Unterstützung der Wartbarkeit der Software, wurde der Programmcode auf unterschiedliche Pakete und Klassen verteilt. Insbesondere wurde darauf geachtet, dass die Implementierung der Benutzeroberfläche und der Netzwerkschnittstelle von der Anwendungslogik getrennt ist. Das in Abbildung 10.19 dargestellte Klassendiagramm vermittelt einen Überblick auf die Programmstruktur der Systemsteuerungssoftware. Im Folgenden wird die Funktionalität der Systemsteuerungssoftware zusammen mit einigen relevanten Aspekten der Implementierung näher beschrieben.

10.4.1 Systemsteuerung

Die Klassen der Systemsteuerungssoftware sind auf zwei Pakete aufgeteilt. Der wesentliche Teil der Funktionalität befindet sich im Paket `systemControl`, welches Klassen für die Erzeugung der grafischen Benutzeroberfläche (GUI), die Verarbeitung von Benutzereingaben und Netzwerkdaten, die Erkennung und Behandlung von Fehlern sowie die Protokollierung und Konfiguration besitzt (siehe Abbildung 10.19).

10.4.1.1 Benutzeroberfläche

Die grafische Benutzeroberfläche der Systemsteuerungssoftware wurde mit Hilfe der Java-APIs *AWT* [Orab] und *SWING* [Orac] in der Klasse `UserInterface` realisiert. Die GUI ist horizontal zweigeteilt und beide Teile sind in mehrere Tabulatoren untergliedert. Der obere Bereich besteht aus folgenden Tabulatoren:

- Der Tabulator ”System Control” (siehe Abbildung 10.20) stellt dem Administrator alle Bedienelemente zur Verfügung, welche erforderlich sind, um die im Systembetrieb vorgesehenen Zustandswechsel (siehe auch Abbildung 4.6 in Kapitelabschnitt 4.6) durchzuführen. Dazu gehört auch die erwartete Anzahl der am Rennen teilnehmenden Fahrzeuge, welche im Rahmen der Inspektion festgelegt werden muss.

- Der Tabulator ”Car Locations” ist für die Visualisierung der im Rennbetrieb empfangen Fahrzeugposen vorgesehen. Da es sich um ein optionale Funktionalität handelt und bedingt durch den Mangel an personellen Ressourcen, wurde diese Visualisierung noch nicht implementiert.

- Der Tabulator ”Settings” bietet dem Administrator die Möglichkeit, während der Inspektion bestimmte Systemparameter anzupassen (siehe Abbildung 10.21). Die

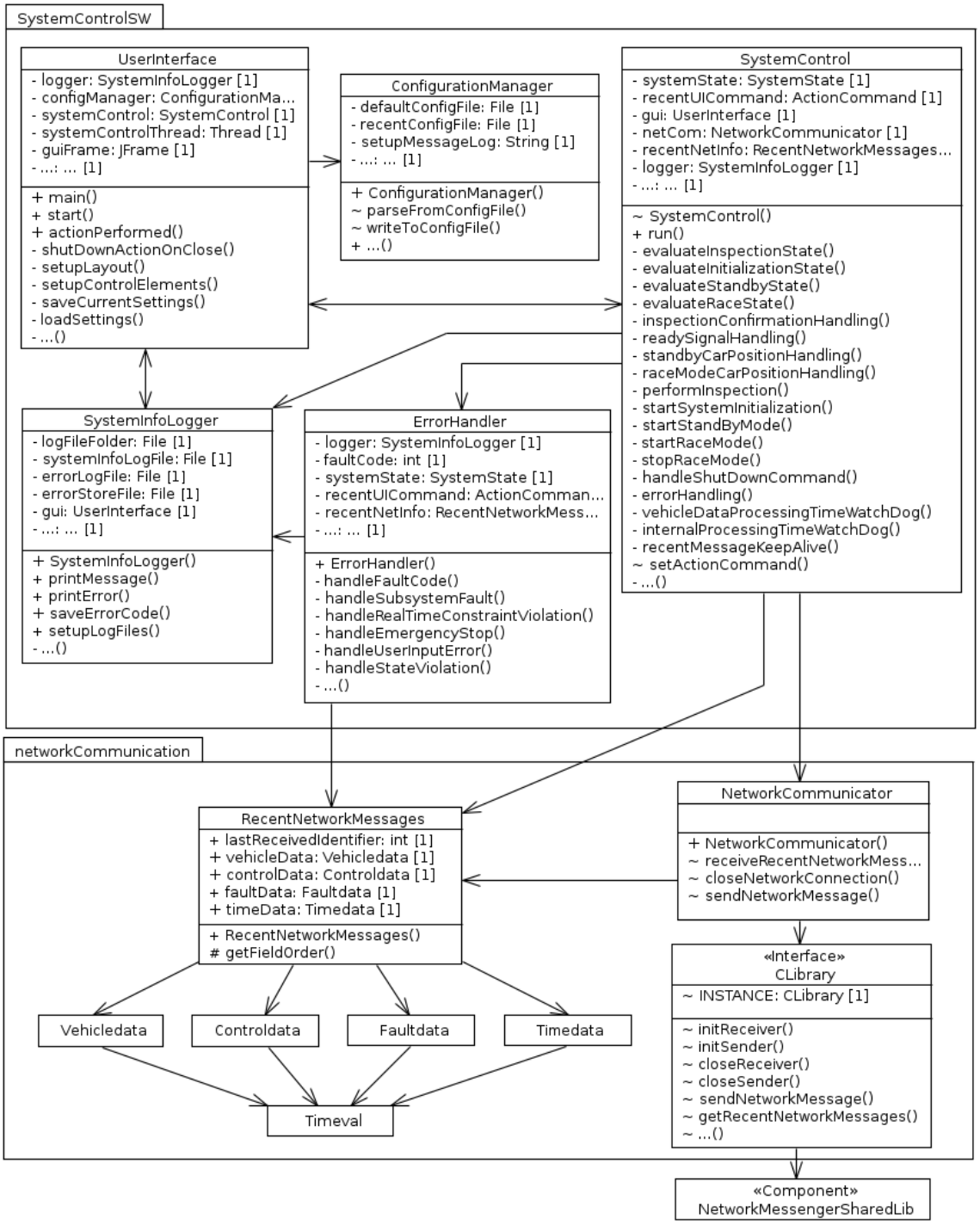


Abbildung 10.19: Klassendiagramm der Systemsteuerungssoftware. Es sind nur die wichtigsten Methoden und Klassenvariablen aufgeführt und alle Methoden sind ohne Parameter dargestellt. Die Multiplizitäten aller Assoziationen besitzen den Wert 1 und wurden daher nicht angegeben.

einzelnen Parameter werden in Anhang 14.6 näher erläutert. Die Parametereinstellungen können in eine Konfigurationsdatei gespeichert und aus dieser wieder geladen werden oder sie können wieder auf Standardwerte zurückgesetzt werden. Die

zuletzt gespeicherten Werte werden beim Start der Systemsteuerungssoftware automatisch geladen. Dies wird durch die Methoden `writeToConfigFile` und `parseFromConfigFile` der Klasse `ConfigurationManager` realisiert.
Sobald im Bereich ”System Control” die erfolgreiche Inspektion (”Confirm Inspection”) bestätigt wurde, werden die Parameter an die Klasse `SystemControl` übergeben und der ”Settings”-Tabulator wird deaktiviert. Die einzelnen Konfigurationsparameter sind in Tabelle 14.4 aus Anhang 14.6 aufgelistet und beschrieben.

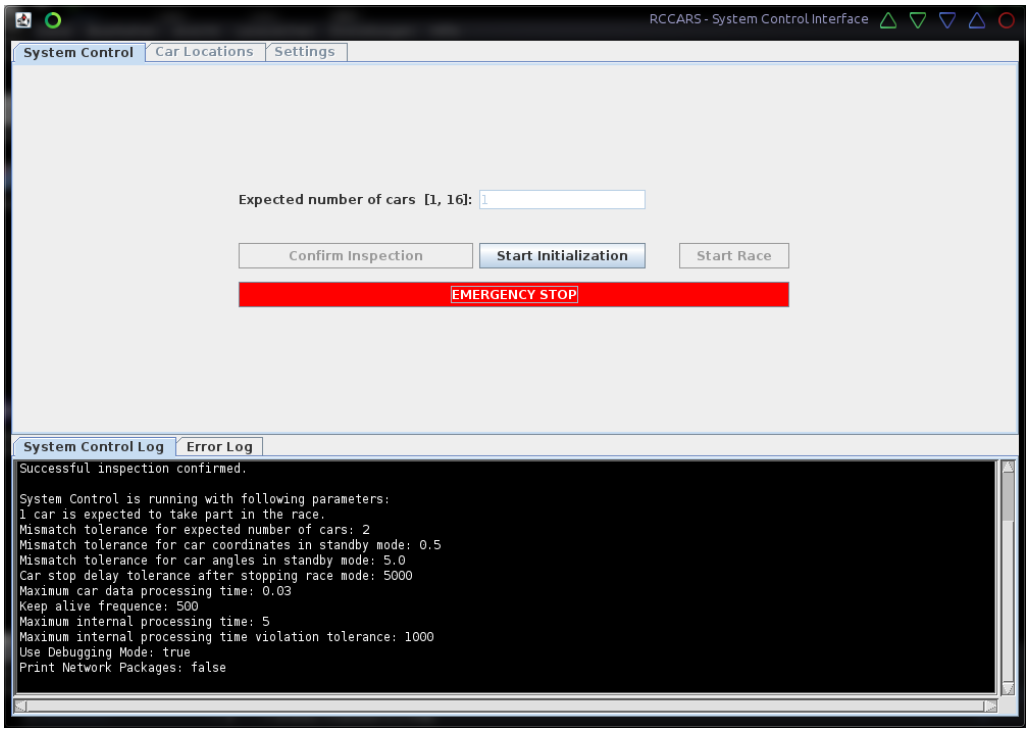


Abbildung 10.20: Screenshot des GUI-Bereichs ”System Control” nach der Bestätigung der Inspektion.

Im unteren Bereich der GUI ist ein Textbereich zu sehen, welcher in folgende zwei Tabulatoren unterteilt ist:

- Der Tabulator ”System Information Log"wird genutzt, um Systeminformationen, Fehlermeldungen und ggf. empfangene Netzwerkdaten auszugeben. Die Ausgabe von Systeminformationen und Netzwerkdaten kann ausgestellt werden, um die Performance der Software zu verbessern (siehe auch Abbildung 10.21). Fehlermeldungen werden hingeben immer ausgegeben.

- Im Bereich ”Error Log” werden Fehlermeldungen gesondert aufgeführt.

In der Klasse `UserInterface` befindet sich außerdem die `main`-Methode der Systemsteuerungssoftware. Neben der Erzeugung der grafischen Benutzeroberfläche werden hier

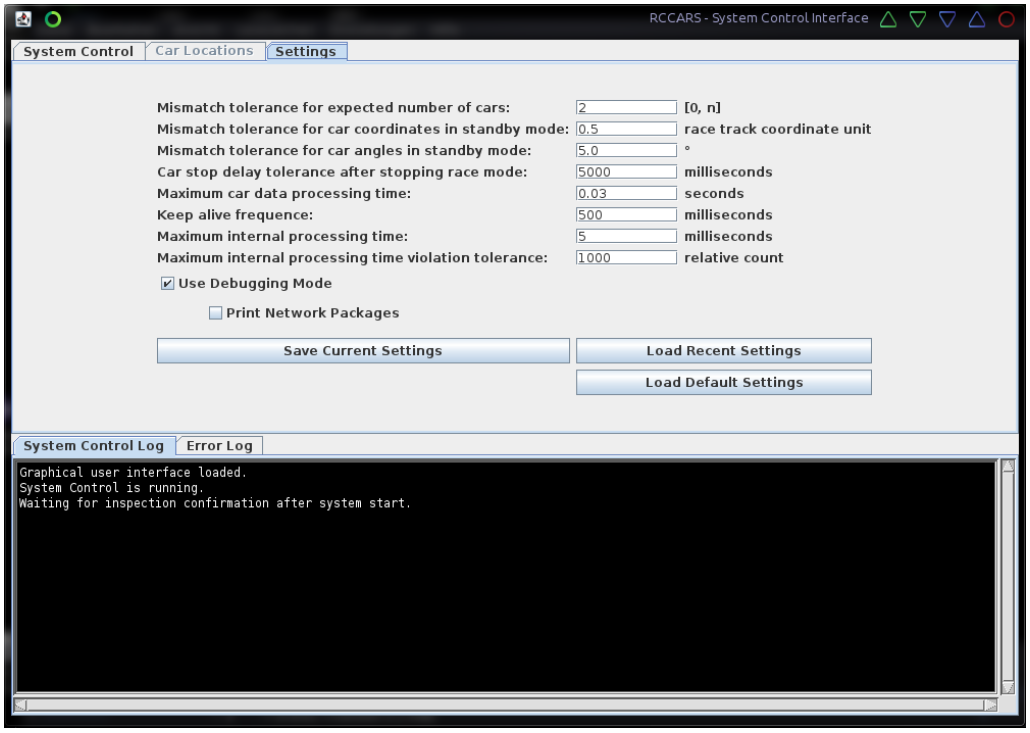


Abbildung 10.21: Screenshot der GUI-Bereichs ”Settings”.

also auch jene Klassen instanziert, welche die Anwendungslogik der Systemsteuerungssoftware realisieren.
Die Klasse `UserInterface` steht in wechselseitigem Kontakt mit der Klasse `SystemControl`. Durch die Methode `setActionCommand` werden neue Benutzereingaben an diese Klasse übergeben. Die Klasse `UserInterface` erhält wiederum Eingaben von `SystemControl`, welche abhängig vom jeweiligen Zustandswechsel bestimmte Elemente der GUI mit Hilfe der Methoden `setInitializationButtonEnabled`, `switchRaceButton` etc. aktiviert bzw. deaktiviert oder verändert. Des Weiteren werden über die Methoden `writeAtSystemInfoTextArea` und `writeAtErrorTextArea` aktuelle Systeminformationen bzw. Fehlermeldungen in die o. g. Textbereiche geschrieben.

10.4.1.2 Systemzustände

Damit der Administrator während des Systembetriebs die vorgesehen Zustandswechsel (siehe Abbildung 4.6) durchführen kann, muss die Systemsteuerungssoftware nicht nur eine Benutzeroberfläche mit den erforderlichen Bedienelementen bereitstellen. Sie muss auch sicherstellen, dass unerlaubte Zustandswechsel, welche zu Unfällen und anderen Störungen führen könnten, nicht unwissentlich oder versehentlich ausgelöst werden. Daher speichert die Systemsteuerungssoftware den aktuellen Systemzustand und überprüft, ob die empfangenen Benutzereingaben erlaubt sind. Gleiches gilt für Netzwerkdatenpa-

kete, welche von der Positionsbestimmungssoftware und der Regelungssoftware ausgegeben werden. Auch diese müssen von der Systemsteuerungssoftware überwacht werden, um Fehler und – abhängig vom aktuellen Systemzustand – unerlaubte Fahrzeugbewegungen zu erfassen.

Innerhalb der Systemsteuerungssoftware werden diese Zustände durch die Enumeration

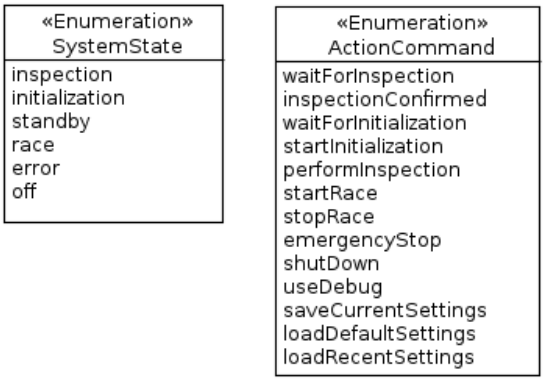


Abbildung 10.22: Grafische Darstellung der Enumerationen `SystemState` und `ActionCommand` der Systemsteuerungssoftware. Diese werden von den Klassen `UserInterface`, `SystemControl` und `ErrorHandler` verwendet und sind aus Gründen der Übersicht in Abbildung 10.19 nicht dargestellt.

`SystemState` (siehe Abbildung 10.22 links) codiert. Durch die Verwendung der Enumeration wird sichergestellt, dass sich das System aus Sicht der Systemsteuerungssoftware ausschließlich in den vordefinierten Zuständen befinden kann.

Die Verarbeitung und Überwachung von Benutzer- und Netzwerkeingaben wird durch die Klasse `SystemControl` realisiert (siehe Abbildung 10.19), welche in einem eigenen Thread ausgeführt wird und somit parallel zum Thread der Klasse `UserInterface` läuft. Dadurch wird sichergestellt, dass die grafische Benutzeroberfläche die Ausführungsgeschwindigkeit der Systemsteuerungssoftware nicht beeinträchtigt.

Solange die Systemsteuerungssoftware nicht beendet wird, wird eine Endlosschleife ausgeführt, welche diese Eingaben ausgehend vom aktuellen Zustand auswertet. Dies geschieht in den Methoden `evaluateInspectionState`, `evaluateInitializationState` etc. Wird eine Benutzereingabe oder ein Netzwerkdatenpaket im entsprechenden Zustand empfangen, wird mit Hilfe der Methoden `startSystemInitialization`, `startStandByMode`, `startRaceMode` etc. ein Zustandswechsel durchgeführt. Je nach Art des Zustandswechsels werden entsprechende Elemente des Benutzerinterfaces aktiviert bzw. deaktiviert und Systeminformationen an das Benutzerinterface weitergeleitet. Außerdem müssen bestimmte Klassenvariablen gesetzt werden. So wird beispielsweise beim Wechsel vom Rennbetrieb in den Standby-Zustand ein aktueller Zeitstempel gespeichert, wel-

cher als Referenz für einen Zeitpuffer verwendet wird. Innerhalb dieser Pufferzeit müssen alle Fahrzeuge zum Stillstand gekommen sein, da ansonsten sofort ein automatisches Not-Aus ausgelöst werden würde. Weitere Details der Implementierung können der Quellcodedokumentation der Systemsteuerungssoftware entnommen werden.

10.4.1.3 Fehlererkennung und Fehlerbehandlung

Die Fehlererkennung findet ebenfalls in der `SystemControl`-Klasse statt. Es gibt folgende Möglichkeiten einen Fehler zu erkennen:

- Der Administrator gibt den Not-Aus-Befehl, indem er den "Emergency Stop"-Button auf dem Benutzerinterface drückt.
- Die Systemsteuerungssoftware empfängt ein Fehlerdatenpaket von der Positionsbestimmungssoftware oder von einer Regelungssoftware.
- Die Systemsteuerungssoftware empfängt ein Netzwerkdatenpaket, welches darauf zurückzuführen ist, dass der Systembetrieb nicht wie vorgesehen abläuft.
  - Hierzu gehört beispielsweise der o. g. Fall, dass sich ein Fahrzeug während des Standby-Zustands bewegt.
- Die Systemsteuerungssoftware empfängt ein Zeitdatenpaket von einer Regelungssoftware, welches zeigt das die Gesamtrealzeitanforderung Sys4.6 verletzt wurde (Methode `vehicleDataProcessingTimeWatchDog`).
- Die Systemsteuerungssoftware erkennt, dass sie ihre eigene Realzeitanforderung nicht einhalten kann (Methode `internalProcessingTimeWatchDog`).
  - Die Realzeitanforderung der Systemsteuerungssoftware wird an dieser Stelle so definiert, dass sie schnell genug laufen muss, um alle von den Subsystemen Positionsbestimmung und Control-Unit empfangen Netzwerkdatenpakete zu verarbeiten. Dieser Wert variiert mit der Anzahl der am Rennbetrieb teilnehmenden Control-Units und muss vom Administrator über den "Settings"-Bereich der GUI korrekt vorgegeben werden.
- Die Systemsteuerungssoftware erhält keine Rückmeldung (in Form von Zeitdatenpaketen) von mindestens einer der am Rennbetrieb teilnehmenden Control-Units mehr (Methode `cuResponseWatchDog`).
- Es wurden Benutzereingaben vorgenommen, welche einen unerlaubten Zustandswechsel zur Folge hätten.

- Dies wird normalerweise von der grafischer Benutzeroberfläche verhindert. Da sich jedoch im Rahmen der Weiterentwicklung des Systems möglicherweise Fehler ergeben könnten, wird auch dieser Fall erkannt.

Jedem Fehler wird ein bestimmter Fehlercode zugewiesen. Dieser Code wird mit dem Aufruf der Methode `errorHandling` an die Klasse `ErrorHandler` übergeben, welche die Auswertung und Behandlung der unterschiedlichen Fehlerfälle übernimmt und den Fehlercode ggf. noch genauer spezifiziert. Dabei werden der Systemzustand und die verantwortliche Komponente berücksichtigt und es wird zwischen kritischen und unkritischen Fehlern unterschieden. Ein kritischer Fehler hat immer zur Folge, dass das System in den Fehlerzustand versetzt und ein Fehlerdatenpaket mit einen Not-Aus-Signal von der Systemsteuerungssoftware in das Netzwerk gesendet wird.
Da die Systemsteuerungssoftware somit eine sicherheitsrelevante Rolle im System einnimmt, muss sichergestellt werden, dass sie während des gesamten Systembetriebs aktiv ist. Zu diesem Zweck sendet die Systemsteuerungssoftware in regelmäßigen Abständen ein ”Keep-Alife-Signal” aus. Falls dieses Signal ausbleibt, muss die Regelungssoftware dies erkennen und das Fahrzeug unverzüglich anhalten. Diese Aufgabe wird durch die Methode `recentMessageKeepAlife` realisiert, welche die von der Systemsteuerungssoftware zuletzt ausgegebene Nachricht mit einer vorgegebenen Rate wiederholt in das Netzwerk sendet, falls keine neue Nachricht zu versenden ist.

10.4.1.4 Protokollierung

Zu Optimierungs- und Fehlerbehandlungszwecken können Systeminformationen und Netzwerkdaten protokolliert werden. Fehlermeldungen werden generell protokolliert und gespeichert, während die Aufzeichnung von Netzwerkdatenpaketen und Informationen über den Systemzustand optional ist. Die Protokollierung wird durch die Klasse `SystemInfoLogger` realisiert (siehe Abbildung 10.19). Durch den Aufruf der Methoden `printMessage` bzw. `printError` werden die übergebenen Texte sowohl in den jeweiligen Bereichen der GUI ausgegeben (siehe Abschnitt 10.4.1.1) als auch in entsprechenden Logdateien gespeichert.
Beim Start der Systemsteuerungssoftware wird in deren Ausführungsverzeichnis im Ordner `log` ein neues Verzeichnis angelegt, in welchem die Logdateien erstellt werden. Analog zur Ausgabe auf der GUI werden Fehlermeldungen zusätzlich gesondert in einer eigenen Logdatei gespeichert. Tritt ein Fehler auf, wird die Meldung außerdem in der Datei `ErrorStore.txt` gespeichert, welche sich im Ausführungsverzeichnis der Systemsteuerungssoftware befindet. Um zu bestätigen, dass zuvor aufgetretene Fehler behoben wurden, muss diese Datei geleert oder gelöscht werden, bevor das System erneut initialisiert werden kann.

10.4.2 Netzwerkkommunikation

Die Netzwerkkommunikation der Systemsteuerungssoftware wird durch Klassen des Paketes `networkCommunication` realisiert (siehe Abbildung 10.19). Die Klasse `NetworkCommunicator` besitzt die Methoden `receiveRecentNetworkMessages` und `sendNetworkMessage`, welche von der Klasse `SystemControl` verwendet werden, um aktuelle Netzwerkdatenpakete zu empfangen und Kontrolldatenpakete (siehe Kapitelabschnitt 4.5) in das Netzwerk zu senden.

10.4.2.1 Netzwerkschnittstelle

In die Klasse `NetworkCommunicator` ist außerdem die Implementierung des JNA-Interfaces `CLibrary` eingebettet, welches die Schnittstelle der Systemsteuerungssoftware zum in C geschrieben Programm `NetworkMessenger` (siehe Kapitelabschnitt 10.2.2) bildet, das für die Kommunikation zwischen den Subsystemen genutzt wird. Dies ist erforderlich, da die Netzwerkdatenpakete, welche vom `NetworkMessenger` ausgegeben und empfangen werden, intern durch den C-Datentyp `Struct` codiert sind und somit nicht direkt in Java verarbeitet werden können. Mit den in `CLibrary` deklarierten Methoden `initReceiver`, `initSender`, `getRecentNetworkMessages`, `sendNetworkMessage` etc. wird auf eine modifizierte Version des Programms `NetworkMessenger` zugegriffen, welches als Bibliothek `NetworkMessengerSharedLib.so` vorliegt. In der `Main.c`-Datei von `NetworkMessengerSharedLib` befinden sich die Methoden mit den entsprechenden Namen und Parametern, welche mit Hilfe von `CLibrary` ausgeführt werden.

10.4.2.2 JNA-Mappings

Damit der Zugriff auf die als `Struct` codierten Netzwerkdaten innerhalb von Java korrekt funktioniert, müssen sog. Mappings für die einzelnen Structs erstellt werden, mit deren Hilfe die Inhalte der C-Structs auf Java-Datentypen abgebildet werden. Diese Mappings werden erstellt, indem das JNA-Interface `Structure` implementiert wird.
Es wurden Mappings für die vier unterschiedlichen Netzwerkdatenpakete `Controldata`, `Faultdata`, `Timedata` und `Vehicledata` erstellt (siehe Kapitelabschnitt 10.2.1), sowie für das C-Struct `Timeval`, welches in diesen vier Netzwerkdatenpaketen verwendet wird. In Abbildung 10.19 sind die Parameter und Methoden dieser Klassen nicht dargestellt, um die Lesbarkeit des Diagramms zu verbessern. Die Klassen besitzen jeweils eine Methode `getFieldOrder`, welche die Reihenfolge der abzubildenden Klassenvariablen festlegt, und eine `print`-Methode. Des Weiteren wurde das Mapping `RecentNetworkMessages` erstellt, welches von der Systemsteuerungssoftware verwendet wird, um die jeweils aktuellsten Netzwerkdaten aus dem `NetworkMessageStore` (siehe 10.2.3) des `Network-`

Messengers zu entnehmen. Außerdem wurde das Mapping Networkpackage implementiert, welches ein Netzwerkdatenpaket repräsentiert, das von der Systemsteuerungssoftware ausgegeben werden soll. Da es effizienter ist, das zu versendende Datenpaket innerhalb des NetworkMessengers zu erzeugen, kommt diese Klasse nicht zum Einsatz.

10.4.3 Erfüllte Anforderungen

- Die Anforderung SW3.1 zur Bedienung des Systems über ein Benutzerinterface, wird – inkl. aller Unteranforderungen – durch die Klasse UserInterface im Zusammenspiel mit SystemControl und SystemInfoLogger erfüllt (siehe Abschnitt 10.4.1). Die Ausgabe der entsprechenden Kontrollbefehle wird dabei durch die Klasse NetworkCommunicator und die damit assoziierten JNA-Hilfsklassen und die Bibliothek NetworkMessengerSharedLib realisiert (siehe Abschnitt 10.4.2).

- Die Anforderung SW3.2 zum Ausführen eines automatischen Not-Aus beim Empfang von Fehlerdatenpaketen der Positionsbestimmungssoftware oder einer Regelungssoftware, wird – inkl. ihrer beiden Unteranforderungen – durch die Klassen SystemControl und ErrorHandler erfüllt (siehe Abschnitt 10.4.1). Die Ausgabe des Not-Aus-Befehls wird wieder durch die Klasse NetworkCommunicator und die damit assoziierten JNA-Hilfsklassen und die Bibliothek NetworkMessenger-SharedLib realisiert.

- Die Basisanforderung SW3 ist mit der Erfüllung von SW3.1 und SW3.2 ebenfalls erfüllt.

- Die Systemsteuerungssoftware besitzt außerdem weitere Funktionalität zur Fehlererkennung und -Behandlung (siehe Abschnitt 10.4.1.3), welche für den sicheren Betrieb des Systems relevant ist:
  - Kontrolle der Einhaltung der Gesamtrealzeitanforderung Sys4.6.
  - Erkennung eines möglichen Ausfalls einer Control-Unit, indem im Rennbetrieb fortlaufend überprüft wird, ob alle aktiven Control-Units regelmäßig Rückmeldungen senden.
  - Kontrolle der Einhaltung der eigenen Realzeitanforderung.
  - Ausgabe eines Keep-Alife-Signals indem die zuletzt gesendete Nachricht wiederholt ausgegeben wird, so dass die Control-Unit das Fahrzeug stoppen kann, falls dieses Signal ausbleibt.
  - Speicherung empfangener Fehlermeldungen in einem Fehlerspeicher. Danach wird eine erneute Initialisierung des Systems solange verhindert, bis dieser

Speicher geleert wurde.

Näheres kann den Ergebnissen der entsprechenden Softwaretests entnommen werden (siehe *RCCARS*-Testbericht).

11 Laufzeitevaluation

Im folgenden Kapitel wird auf die Untersuchung der im System auftretenden Latenzen eingegangen. Die Untersuchungen erstreckten sich vom Auswerten der Softwarelaufzeiten mittels Log-Dateien, über dem Aufbau eines Teststandes und zahlreichen Messungen mittels eines Oszilloskops. Im Folgenden werden die Ergebnisse der Evaluation dargestellt.

11.1 Positionsbestimmung

Die Laufzeit der Positionsbestimmung wurde mittels sechs unterschiedlicher Zeitstempel protokolliert. Jeweils ein Stempel wurde zu Beginn und ein weiterer zum Ende eines Systemdurchlaufs gesetzt. Dies beinhaltet das Anfordern und Bearbeiten eines Kamerabildes, die Objekterkennung, die Koordinatentransformation und das Versenden der Pose. Zwei weitere Zeitstempel wurden jeweils vor und nach dem Anforderungen eines Bildes platziert. Die letzten beiden Zeitstempel protokollieren das Ende der Objekterkennung und den Zeitpunkt, an dem die Fahrzeugpose versendet wurde. Dies ermöglicht es lückenlos die Laufzeit der Positionsbestimmung zu dokumentieren.

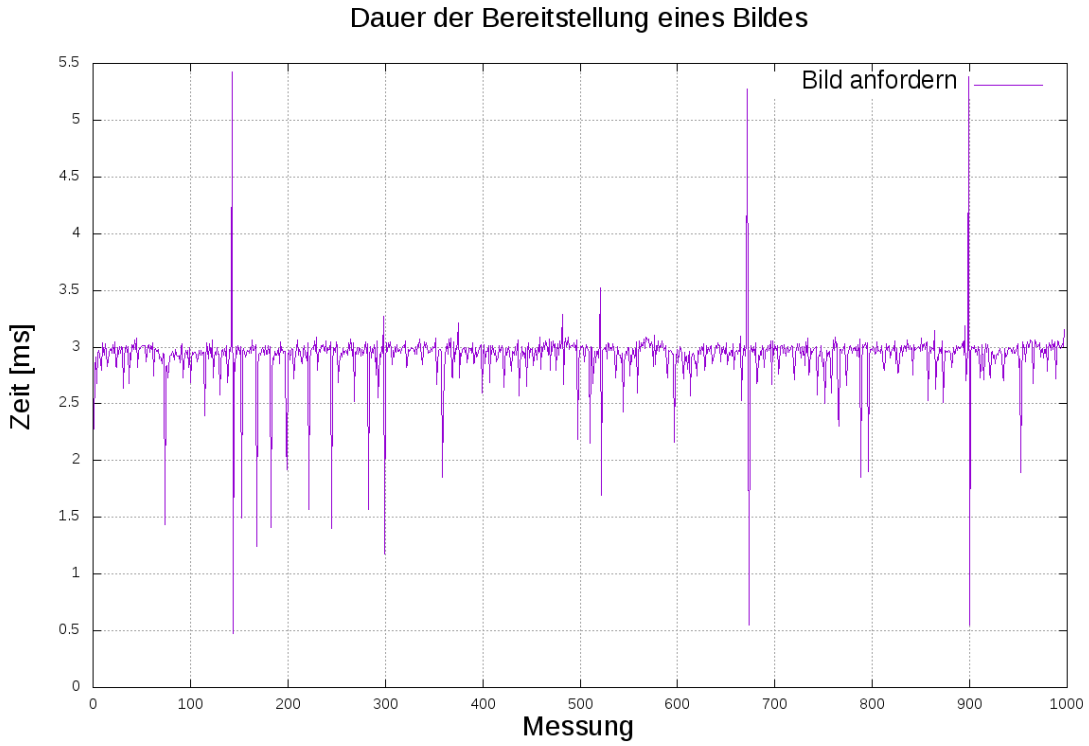


Abbildung 11.1: Dauer der Bereitstellung eines Bildes.

In Abbildung 11.1 dargestellt, wie viel Zeit zwischen dem Anfordern und dem Erhalten eines Kamerabildes vergehen. Im Durchschnitt dauert das Anfordern eines Bilder 2,9 ms, wobei es vereinzelt Abweichungen gibt. Im Worst-Case vergingen im Rahmen der Messungen etwa 5,5 ms bis die Bilddaten am Computer verfügbar waren. Dies liegt an der Konfiguration der Kamera, die mit einer Bildwiederholungsfrequenz von 150 Hz Aufnahmen belichtet und nur die aktuellsten Bilder auf Anfrage der Software übermittelt. Betrachtet man zusätzlich Abbildung 11.2, welche die Zeit zwischen Filtern des Bildes und Bestimmung der Pose veranschaulicht, wird deutlich, weshalb der oben genannte Worst-Case auftreten kann. Die Objekterkennung ist im Schnitt nach 2,3 ms abgeschlossen. Dies bedeutet, dass die Positionsbestimmungssoftware, abzüglich der Zeit, die für das Versenden der Pose und weiteren Aufgaben vergeht, auf ein neues Bild warten muss. An dieser Stelle wird jedoch angemerkt, dass die Positionsbestimmungssoftware ohne Systemsteuerungssoftware ausgeführt wurde. Dies hat zur Folge, dass nach dem Versenden des Fahrzeugdatenpaketes eine weitere Millisekunde auf Eingaben des Administrators gewartet wird.

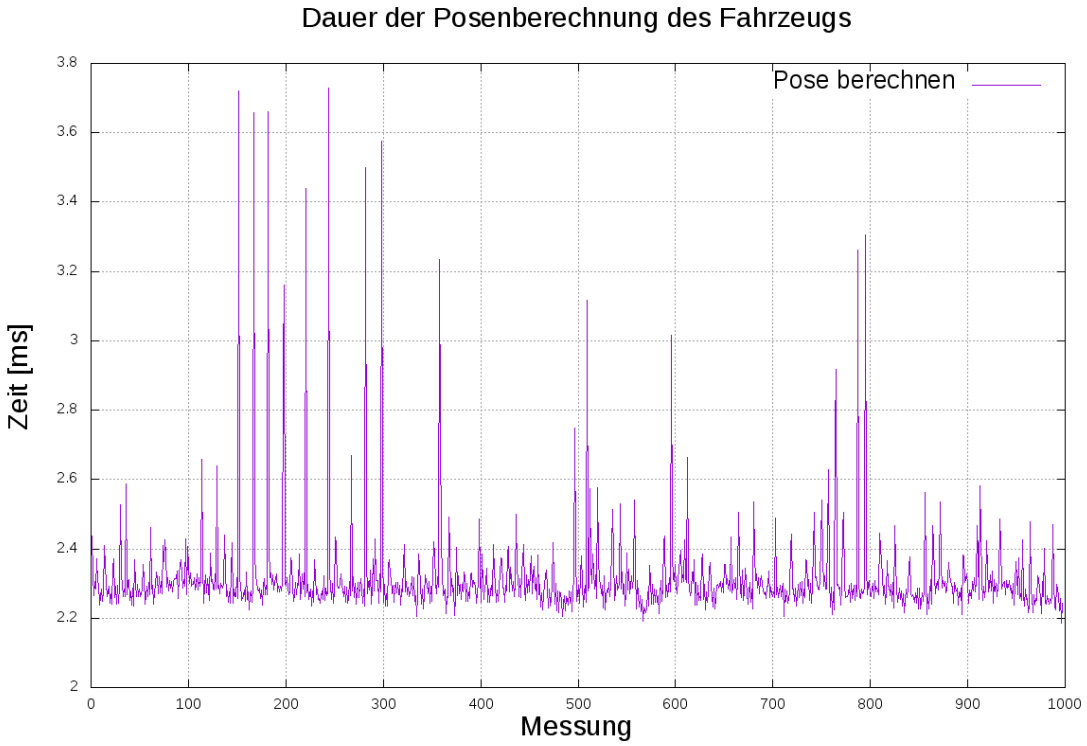


Abbildung 11.2: Dauer der Bearbeitung und Auswertung eines Bildes.

Die folgende Abbildung 11.3 verdeutlicht, in welchem Zeitintervall die Positionsbestimmung Fahrzeugposen in das Netzwerk versendet. Obwohl es in unregelmäßigen Abständen Abweichungen um den Mittelwert gibt, versendet die Positionsbestimmung im Mittel alle 6,67 ms eine neue Pose. Dies entspricht exakt der Belichtungszeit der Kamerabilder. Abweichungen um den Mittelwert sind auf die unterschiedlichen, oben genannten, Wartezeiten zurückzuführen.

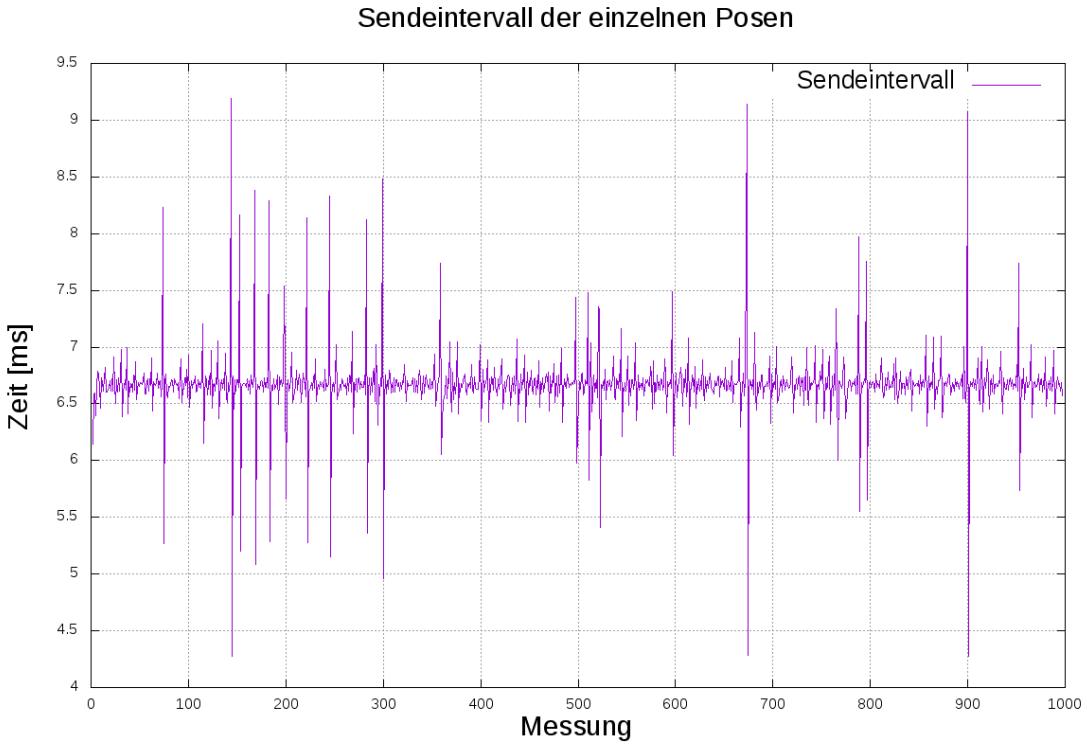


Abbildung 11.3: Dauer der Bearbeitung und Auswertung eines Bildes.

Abbildung 11.4 fasst die ermittelten Worst-Cases Laufzeiten der Positionsbestimmungssoftware zusammen. Mit Hilfe von über dokumentierten 1000 Messwerten wurde festgestellt, dass, sogar im ungünstigsten Fall, die Aufgaben des Positionsbestimmung in weniger als 10 ms durchführbar sind. Damit kann die Anforderung SW1.4 selbst unter Betrachtung mehrerer gleichzeitig auftretender Worst-Cases erfüllt werden.

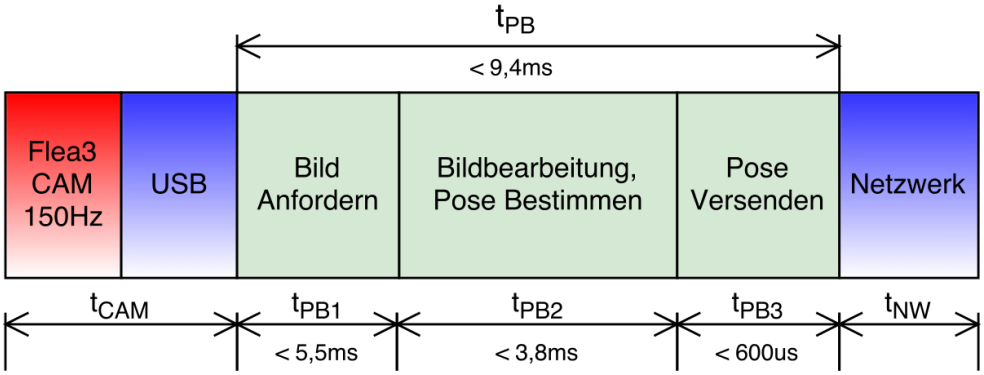


Abbildung 11.4: Worst-Case Betrachtung der Positionsbestimmungssoftware.

11.2 Control-Unit

Die Laufzeit der Software der Control-Unit wurde ebenfalls evaluiert, da sich die Rechenzeit dieses Subsystems unmittelbar auf das Verhalten des Fahrzeugs auswirkt. Im Rahmen der Evaluation wurde daher die mittels *SCADE* generierte Software und die Programme, die Daten für das *SCADE*-Modell vor- und nach bearbeiten, untersucht. Wie bei der Positionsbestimmungssoftware wurden die Laufzeiten mittels Zeitstempeln protokolliert. Die Ergebnisse aus 1200 Messungen werden in den folgenden Unterkapiteln zusammengefasst.

11.2.1 Wrapper-Code und SCADE

Wie man Abbildung 11.5 entnehmen kann, ist die Laufzeit der Vorverarbeitung der Daten, bei welcher u.a. die Fahrzeugpose ausgelesen wird, selbst im vereinzelten Worse-Case Fall mit 15 μs vernachlässigbar gering.

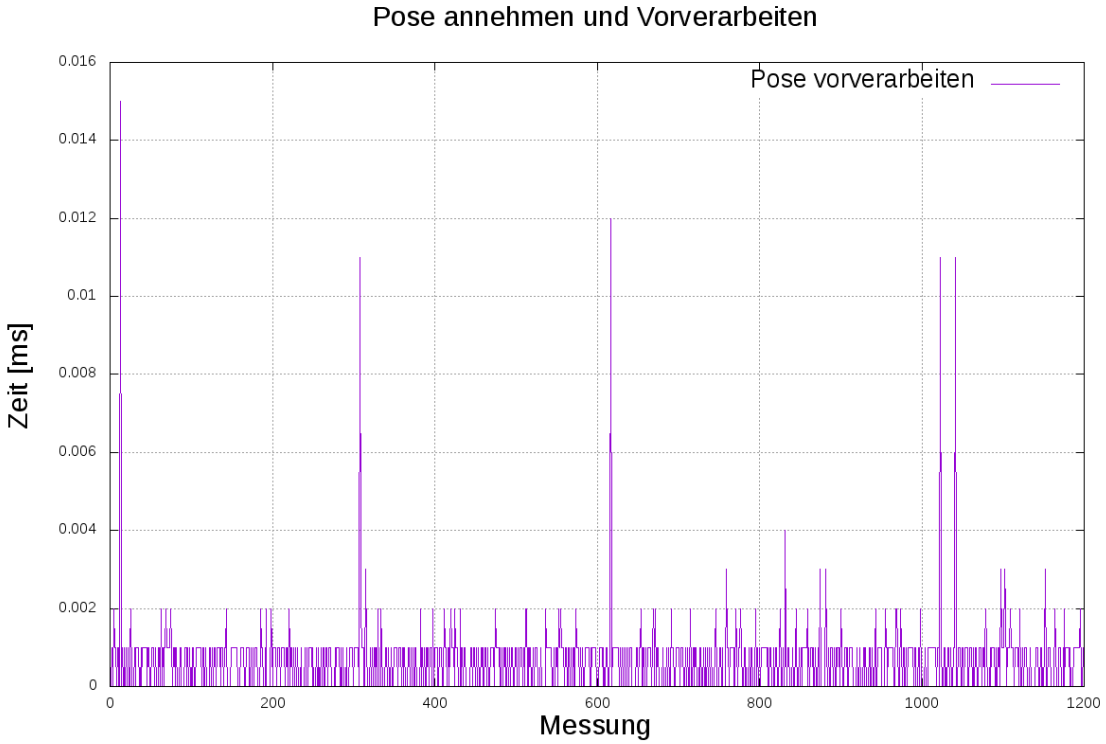


Abbildung 11.5: Lautzeit der Vorverarbeitung.

Der mittels *SCADE* generierte Code ist erwartungsgemäß äußerst performant. Dies bestätigt die Abbildung 11.6, auf welcher zu sehen ist, dass im Durchschnitt 530 μs benötigt werden, um die Regelungsparameter zu berechnen. Auffällig ist jedoch der 368.

Messwert, da dieser mit über 2 ms signifikant von den anderen Messdaten abweicht. Dies könnte evtl. auf einen Messfehler zurückzuführen sein.

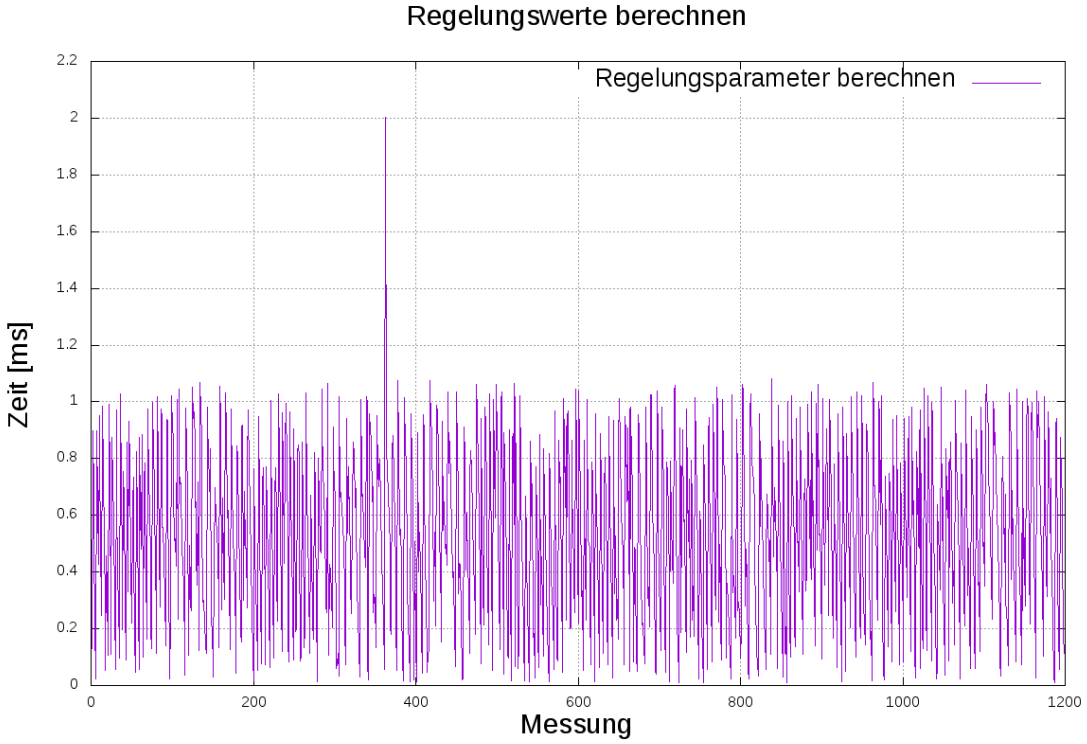


Abbildung 11.6: Laufzeit des Scade-Codes.

Die Laufzeit des Wrapper-Codes in der Nachbearbeitung ist ebenfalls mit durchschnittlich 541 μs gering. Abbildung 11.7 fasst die Gesamtlaufzeit der Control-Unit zusammen. Zwischen dem Zeitpunkt an dem die Vorverarbeitung beginnt und dem Zeitpunkt an dem die Nachbearbeitung abgeschlossen wird, vergeht im Mittel nur eine Millisekunde.

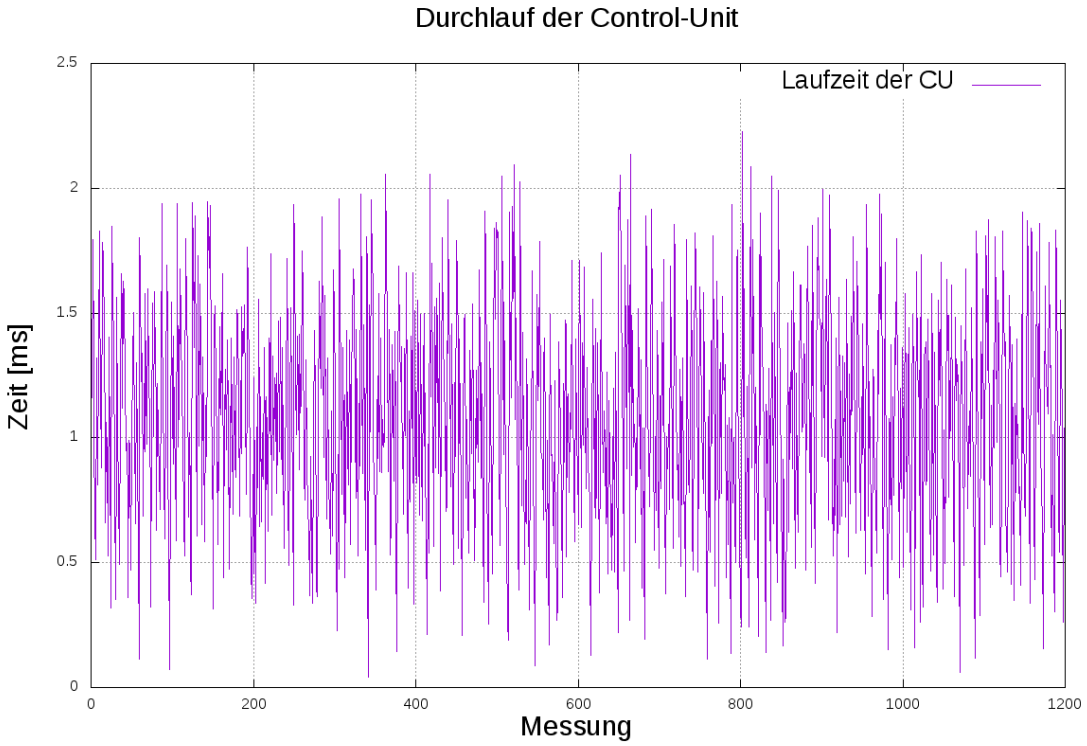


Abbildung 11.7: Gesamtlaufzeit der Control-Unit.

Nichts desto trotz werden zur Schätzung einer oberen Schranke der Laufzeit die Summe sämtlicher Worse-Cases betrachtet (siehe Abbildung 11.8). Die Vor- und Nachbearbeitung beansprucht im schlimmsten Fall über 1,2 ms. Damit benötigt die Control-Unit im Worst-Case 3,4 ms um aus den Daten der Positionsbestimmungssoftware Regelungsdaten zu bestimmen und weiterzuleiten.

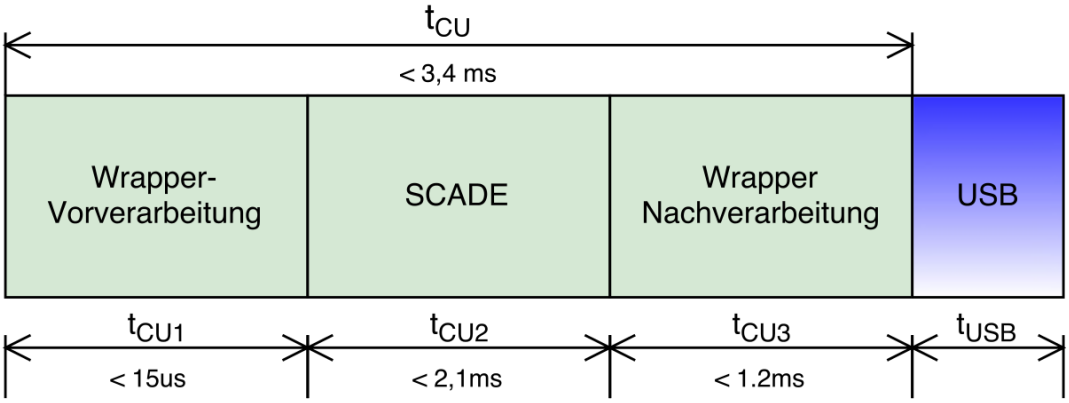


Abbildung 11.8: Worst-Case Betrachtung der Control-Unit.

11.2.2 CarControl und Fahrzeug

Obwohl das Fahrzeug ein eigenes Subsystem darstellt, werden die Ergebnisse der Evaluation gemeinsam mit der CarControl vorgestellt, da beide Komponenten geschlossen getestet wurden (siehe Abbildung 11.9). Dabei wurde mittels des in der Abbildung sichtbaren Oszilloskops die Dauer des Empfangs eines Regelungsdatenpaketes, die Verarbeitungszeit der Daten und die Laufzeit der Digital-Analog-Wandlung ermittelt. Außerdem wurde dokumentiert, wie lange es dauert, bis am Motoreingang Spannungsänderungen feststellbar sind.

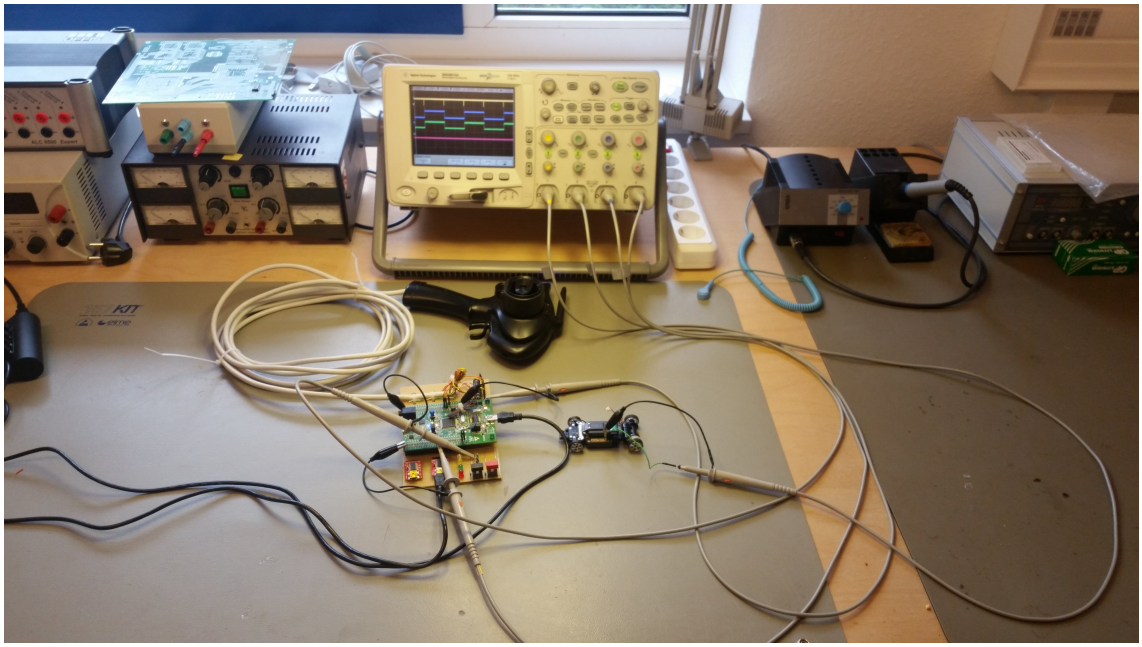


Abbildung 11.9: Teststand CarControl und Fahrzeug.

In Abbildung 11.10 ist das analoge Datenpaket, welches die nächsten Befehle zur Längs- und Querführung des Fahrzeugs enthält, dargestellt. Bei den Untersuchungen wurde festgestellt, dass etwa 1,5 ms vergehen, bis die Daten vollständig der CarControl vorliegen.

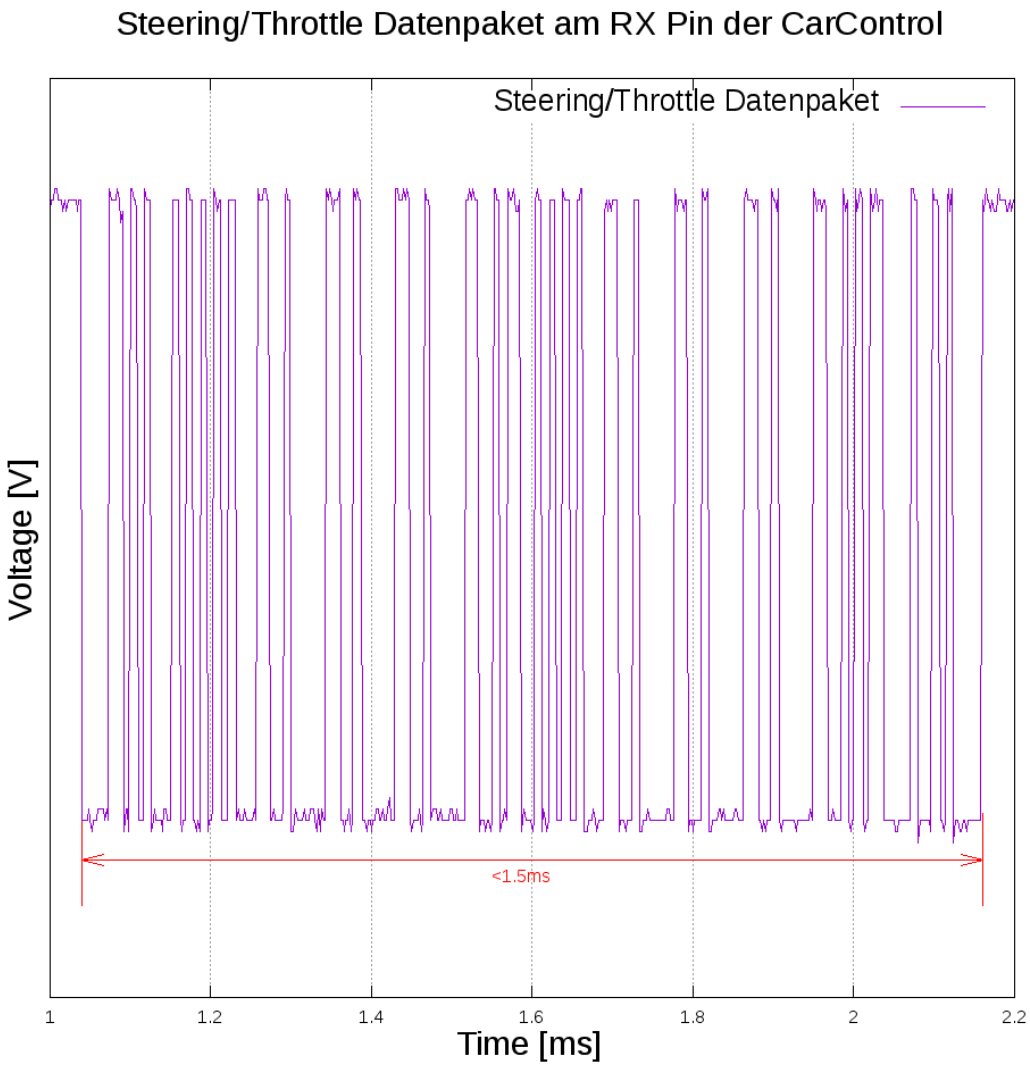


Abbildung 11.10: Datenpaket mit Regelungsparametern.

Um die Verarbeitungszeit der CarControl zu untersuchen, wurde die Spannungsänderung auf einen Ausgabe Pin des Entwicklungsboards beobachtet. Sobald das Datenpaket eingetroffen ist, wertet die CarControl dieses aus und wandelt die Steuerungssignale in Spannungen um. Dies ist in Abbildung 11.11 zu erkennen. Bis zum Zeitpunkt t=3 ms befindet sich der D/A Wandler, welcher mit der Fernsteuerung des Fahrzeugs verbunden ist, in Neutralposition. Nachdem das Datenpaket jedoch ausgewertet wurde, ändert sich der Stellwert binnen 0,8 ms auf 1,5 V, was zur Folge haben wird, dass das Fahrzeug den Befehl erhält, zu beschleunigen.

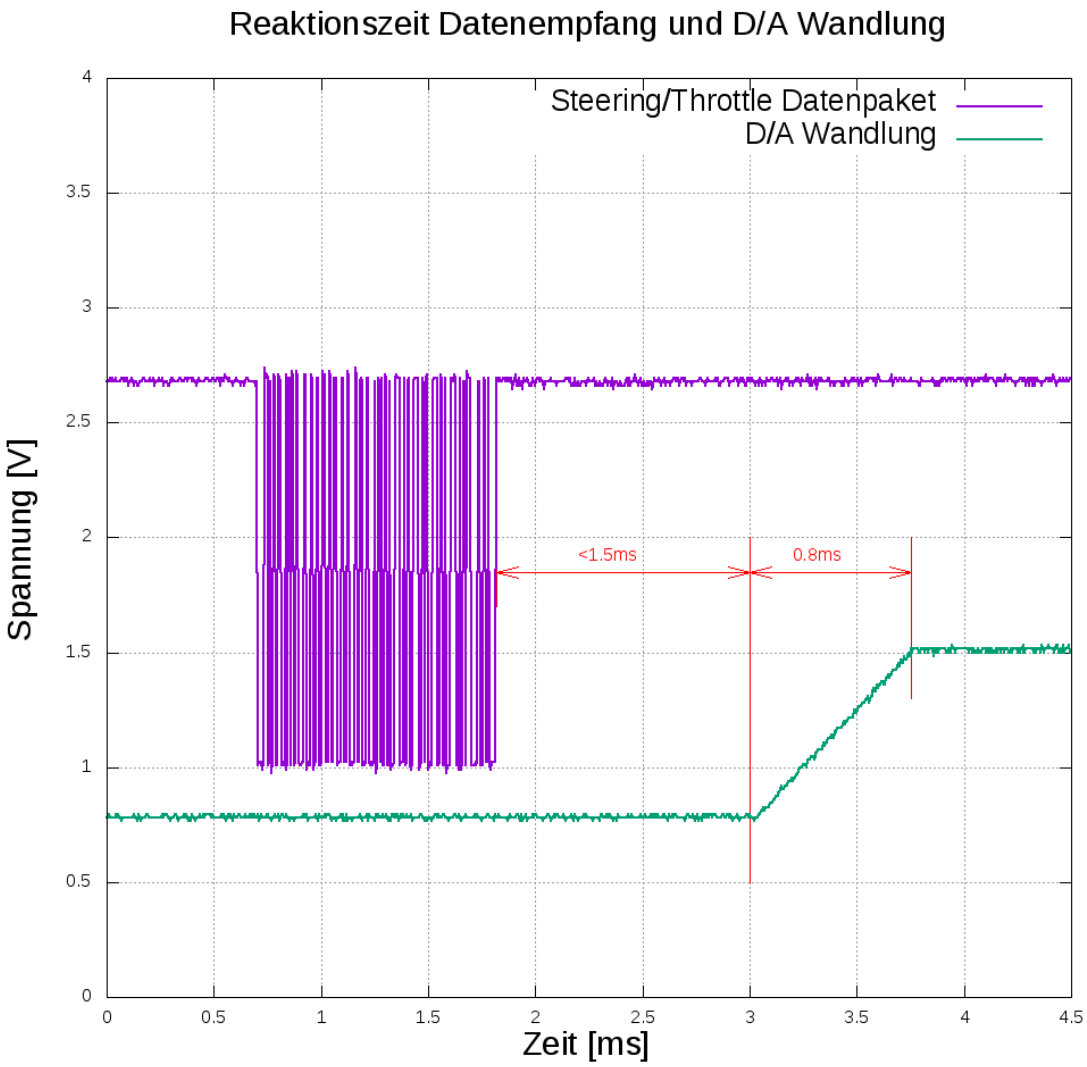


Abbildung 11.11: Veränderung der Spannung an der Fernsteuerung.

Da nun die Latenzen der CarControl bekannt sind, kann festgestellt werden, wie lange es dauert, bis der Motor auf neue Steuerungsbefehle reagiert. Das Verhalten der Spannung am Motoreingang kann in Abbildung 11.12 entnommen werden. Nachdem das Datenpaket zum Zeitpunkt t = 2,6 ms vollständig eingetroffen ist und binnen 0,7 ms von der CarControl ausgewertet wurde, findet die Digital-Analog-Wandlung statt. Sobald die Wandlung ab t = 3,9 ms abgeschlossen ist, sendet die Fernsteuerung einen Beschleunigungsbefehl an das Fahrzeug. Eine Änderung des Signals ist jedoch erst nach etwa 4 ms am Motor messbar. Somit ist die Zeit, die die Fernsteuerung benötigt um neue Steuerungssignale an das Fahrzeug zu leiten bekannt.

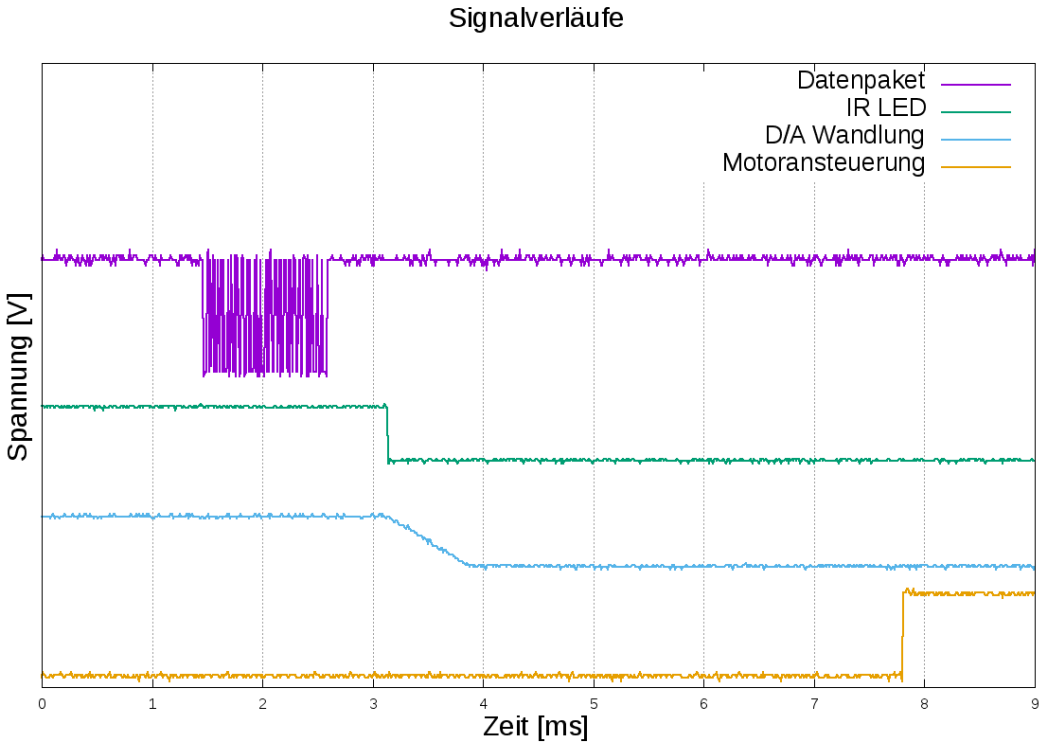


Abbildung 11.12: Veränderung der Spannung an der Fernsteuerung.

Wie auch in den vorigen Unterkapiteln wurde eine Betrachtung sämtlicher Worst-Cases durchgeführt (siehe Abbildung 11.13). Wiederholte Messungen haben ergeben, dass die CarControl höchstens 3,8 ms benötigt um ein Signal zu empfangen, zu verarbeiten und in Spannungen zu wandeln. Seitens der Fernsteuerung wurden nicht einheitliche Zeiten festgestellt. Dies hängt womöglich mit der internen Taktung dieser zusammen. Daher schwankt die Übertragungszeit zwischen 4 und 12 ms. Somit vergehen im schlimmsten Fall etwa 16 ms, bis die berechneten Regelungsparameter der Control-Unit das Fahrzeug erreichen. An dieser Stelle bleibt offen, innerhalb welches Intervalls der Motor die Signale in kinetische Energie umwandeln kann.

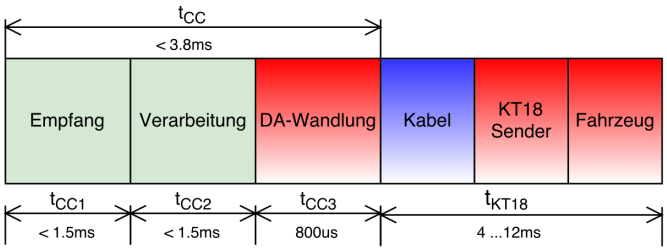


Abbildung 11.13: Worst-Case Betrachtung der CarControl und Motorsteuerung.

11.3 Ergebnisse

In diesem Kapitel werden die Resultate der obigen Untersuchungen in der Abbildung 11.14 zusammengefasst.

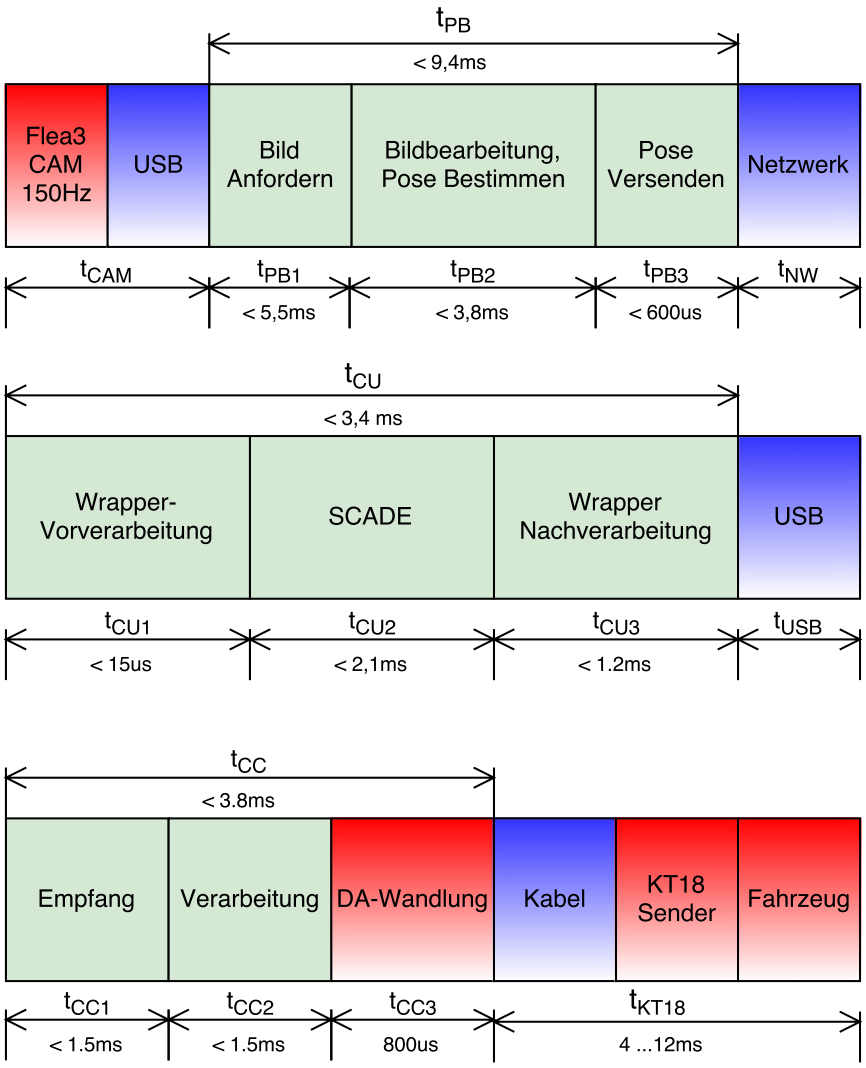


Abbildung 11.14: Worst-Case Betrachtung der CarControl und Motorsteuerung.

Sämtliche Worst-Cases, die innerhalb der Subsysteme im Rahmen der Stichprobenmessung festgestellt werden konnten, sind aufsummiert worden. Damit ergibt sich insgesamt eine Laufzeit von 28.6 ms. Somit unterschreitet diese obere Schranke die in Kapitelabschnitt 9.4 beschriebene Realzeitanforderung an das System.

12 Testen

Obgleich von den im Maßstab 1:43 vorliegenden Fahrzeugen im Falle einer Fehlfunktion keine Lebensgefahr ausgeht, ist das zu entwickelnde System dennoch sicherheitskritisch. Anlass dafür ist, dass sich einzelne Aspekte des Systems zur autonomen Steuerung von RC-Cars auf Kraftfahrzeuge im Maßstab 1:1 übertragen lassen sollen. Aus diesem Grund, sowie aus der Gründen der Qualitätssicherung, wurden umfangreiche Tests und Validierungen während der Entwicklung durchgeführt.
Im folgenden wird zunächst die Testplanung besprochen, welche in diesem Projekt zum Einsatz kam. Anschließend werden grundlegende Begrifflichkeiten zu den verschiedenen Testarten erläutert und es wird deren Anwendung in den hierarchisch organisierten Teststufen festgelegt, welche in den unterschiedlichen Entwicklungsphasen des Systems zum Einsatz kamen. Außerdem wird die Anwendung von bestimmten Technologien und Vorgehensweisen besprochen und es wird gezeigt, wie diese Konzepte realisiert wurden. Zum Schluss des Kapitels werden die grundlegenden Testfälle und Testszenarien, welche im Rahmen der ersten Ausbaustufe des Projektes zu erfüllen waren, aufgeführt und es wird zusammengefasst, in welchem Umfang diese mit den geforderten Technologien und Vorgehensweisen realisiert wurden.

12.1 Testplanung

Die Testplanung orientiert sich an dem im Kapitelabschnitt 3.1 beschriebenen Vorgehensmodell. Innerhalb eines Sprints werden abhängig vom jeweiligen Subsystem die entsprechenden Komponenten entwickelt, welche sich an den in Kapitel 6 definierten Anwendungsfällen orientieren. Die in Abschnitt 12.7 aufgeführten Testfälle definieren, wie diese Komponenten getestet werden müssen. Dies beinhaltet auch die Integration der Komponenten in das jeweilige Subsystem. In Abschnitt 12.8 wird bei einem abschließenden Testszenario die Funktionalität des gesamten Systems im Bezug auf das Szenario ”Unfallfreie Fahrt” (siehe Kapitelabschnitt 2.4) überprüft.
Des Weiteren war der Einsatz von statischen Testverfahren vorgesehen. So sollten während des gesamten Entwicklungsprozesses Kreuztests vorgenommen werden. Bei diesen werden die zuvor beschriebenen Tests nicht vom Entwickler der zu testenden Komponente

selbst, sondern von einem anderen Mitglied des Teams durchgeführt, um die unabhängige Durchführung der Tests zu unterstützen. Auch Code-Reviews sollten regelmäßig nach der Fertigstellung einer Komponente absolviert werden.
Das verwendete Entwicklungswerkzeug *SCADE*, welches für die Modellierung der Regelung der autonomen Fahrzeugsteuerung verwendet wurde, verfügt außerdem über interne Simulationsfunktion, mit deren Hilfe sicherheitskritische Aspekte der Regelung getestet werden sollten.

12.2 Testarten

Bevor die Testorganisation näher beschrieben wird, werden einige grundlegende Begriffe des Testens erläutert.

12.2.1 Blackboxtests

Blackboxtests sind funktionelle Tests, welche ohne Kenntnisse über die innere Funktionsweise des Systems bzw. der Systemkomponenten durchgeführt werden. Nur das nach außen erkennbare Verhalten wird betrachtet. Bei Softwaretests werden die Testfälle daher nicht gemäß der Implementierung der zu testenden Komponente definiert, sondern anhand ihrer Spezifikation. Häufig kommen Äquivalenzklassentests (siehe Abschnitt 12.2.3) zum Einsatz, um die Anzahl der zu testenden Eingabewerte zu verringern.

12.2.2 Whiteboxtests

Whiteboxtests sind strukturbasierte Tests, insbesondere Softwarequellcodetests. Im Gegensatz zu Blackboxtests, ist die innere Funktionsweise bzw. Implementierung der Systemkomponenten bekannt und wird gezielt überprüft. Für Softwaretests gelten die sog. Überdeckungskriterien:

- Anweisungsüberdeckung:
  Alle Programmanweisungen der Software müssen durch Tests abgedeckt sein.
- Kantenüberdeckung:
  Alle Kanten bzw. Zweige im Quellcode müssen durch Tests abgedeckt sein.
- Bedingungsüberdeckung:
  Es müssen alle Varianten der logischen Bedingungen des Programmcodes durch die Tests abgedeckt sein.

- Pfadüberdeckung:
  Die Pfadüberdeckung verlangt, dass alle funktionalen Programmpfade getestet werden. Das bedeutet, dass auch alle möglichen Variablenbelegungen getestet werden müssen. Auf Grund der hohen Anzahl der möglichen Pfade, ist die Erfüllung der Pfadüberdeckung für dieses Projekt nicht realistisch. Daher müssen Äquivalenzklassen definiert werden, um die Anzahl der zu testenden Werte zu verringern.

Aufgrund der personellen Unterbesetzung der Projektgruppe wurde entschieden, dass lediglich die Anweisungsüberdeckung erfüllt werden soll. Durch die Kombination mit Äquivalenzklassentests soll sicher gestellt werden, dass Testeingaben so gewählt werden, dass alle sicherheitskritischen Fälle berücksichtigt werden. Beispielsweise muss eine Methode, welche unterschiedliche Fehlercodes auswertet, durch einen Unit-Test (siehe Abschnitt 12.3.1) überprüft werden, welcher jeden möglichen Fehlercode berücksichtigt, auch wenn eine Anweisungsüberdeckung bereits durch eine Teilmenge aller Fehlercodes erreicht wird.

12.2.3 Äquivalenzklassentests

Da es nicht realistisch ist alle möglichen Belegungen einer Variable zu testen, muss der Wertebereich in Äquivalenzklassen eingeteilt werden. Dies sind Intervalle innerhalb des Wertebereichs der Variable, welche das selbe Programmverhalten repräsentieren. Diese Intervalle müssen von einem Entwickler der entsprechenden Komponente manuell spezifiziert werden. In der Regel werden zum Testen die Randbereiche dieser Intervalle verwendet, da die Wahrscheinlichkeit für das Auftreten von Fehlern dort am höchsten ist.

12.2.4 Performanztests

Um zu zeigen, dass die Realzeitanforderungen erfüllt werden, müssen Performanztests durchgeführt werden. Diese müssen zeigen, dass die Softwarekomponenten auf der verwendeten Hardware ausreichend schnell ausgeführt werden.

12.3 Testautomatisierung

Durch eine Testautomatisierung soll die Effizienz des Testvorgangs gesteigert werden. Insbesondere soll die wiederholte Ausführung von Tests nach Änderungen des Systems (Regressionstests) automatisiert werden.

12.3.1 Unit-Tests

Um die Ausführung von Komponententests zu automatisieren wird ein Unit-Testframework verwendet. Solch ein Framework bietet die Möglichkeit, Testklassen und -Methoden zu implementieren, mit denen Whiteboxtest automatisch ausgeführt werden können. Mehrere Testmethoden können zu so genannten Testsuits zusammengefasst werden. Dies ist insbesondere für das Regressionstesten wichtig, da auf diese Weise alle Unit-Tests für ein Subsystem oder das gesamte System automatisch wiederholt werden können. Es ist jedoch zu beachten, dass die von den Testmethoden zurückgelieferten Testergebnisse nur dann korrekt sein können, wenn der Test korrekt implementiert wurde.
Im Projekt *RCCARS* kommt das Testframework *CUTE* [LF15] zum Einsatz. *CUTE* ist eine plattformübergreifende Software für die Erstellung von Unit-Tests in C++ und kann als Plugin in die verwendete Entwicklungsumgebung *Eclipse* integriert werden. Dies vereinfacht das Anlegen von Unit-Tests, da automatisch ein Testprojekt inkl. Testklasse und Testsuit generiert wird. Der zu testende C++/C-Code kann in Form von Bibliotheken in die Testklasse eingebunden werden. Das Framework unterstützt die für das Unit-Testing typischen Assertmethoden (assertTrue(), assertFalse(), assertEquals()).
Für die zum Großteil in Java entwickelte Systemsteuerungssoftware steht das Unit-Testframework *JUnit* [JU14] zur Verfügung, welches ebenfalls über diese Funktionalität verfügt.

12.4 Teststufen

Analog zur den Entwicklungsphasen des Systems ist das Testen in unterschiedliche Teststufen unterteilt. Auf der untersten Stufe stehen die Tests von Systemkomponenten, welche innerhalb eines Sprints hinzugefügt oder modifiziert wurden, während an oberster Stelle der Systemabnahmetest steht (siehe Abbildung 12.1).

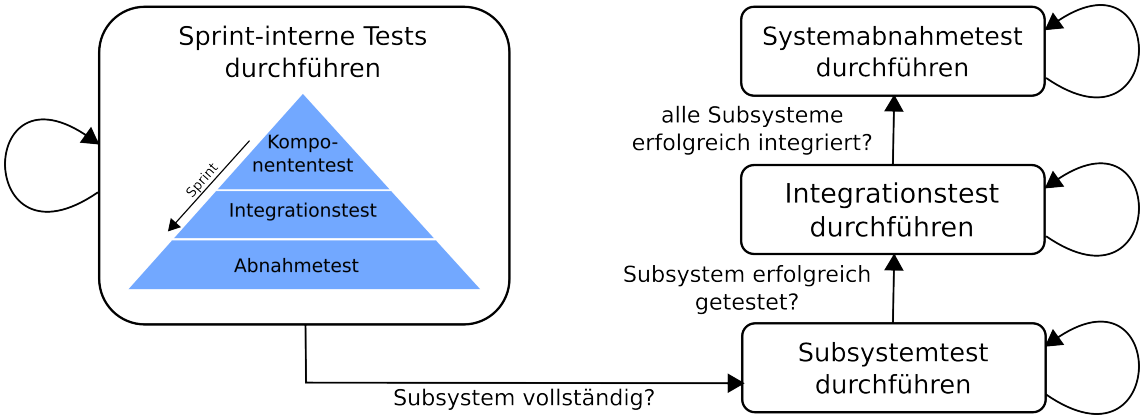


Abbildung 12.1: Grafische Darstellung des Testablaufs im Projekt *RCCARS*.

12.4.1 Sprint-interne Tests

Gemäß des Vorgehensmodells *ScrVm* ist die Systementwicklung in mehrere Sprints untergliedert. In der Regel werden mehrere Sprints gleichzeitig durchgeführt, so dass die Subsysteme parallel entwickelt werden können. Auf jede Entwicklungsphase eines Sprints folgt eine entsprechende Testphase (Abbildung 12.1 links), in der alle von Änderungen betroffenen Komponenten getestet werden. Um sicher zu stellen, dass jede dieser Komponenten korrekt funktioniert und keine Fehler in das System einbringt, müssen – ähnlich wie beim V-Modell – folgende Teststufen durchgeführt werden. Bei den sprint-internen Tests handelt es sich um Whiteboxtests.

- Komponententests:
  Unter Komponenten sind funktionelle Einheiten eines Subsystems zu verstehen. Jede neue oder modifizierte Komponente muss getestet werden.

- Integrationstests:
  Wenn eine neue oder modifizierte Komponente in ein Subsystem integriert wird, muss getestet werden, ob diese korrekt innerhalb des Subsystem funktioniert. Außerdem muss sichergestellt werden, dass keine Fehler in das Subsystem eingebracht wurden. Zu diesem Zweck müssen Regressionstests durchgeführt werden. Das bedeutet, dass alle vorangegangen Testfälle des jeweiligen Subsystems erneut ausgeführt werden.

- Abnahmetest:
  Durch den Abnahmetest eines Sprints soll festgestellt werden, ob alle für den Sprint relevanten Anforderungen erfüllt sind. Der Abnahmetest eines Sprints ist erfüllt, wenn alle im Sprint neu entwickelten oder modifizierten Komponenten erfolgreich getestet und in das Subsystem integriert wurden.

12.4.2 Subsystemtests

Um zu zeigen, dass die Entwicklung eines Subsystems erfolgreich abgeschlossen wurde, muss ein Subsystemtest durchgeführt werden. Zum Subsystemtest gehört ein Blackboxtest, der alle Anforderungen abdecken muss, welche an das Subsystem gestellt werden. Ein Subsystemtest kann nur dann erfolgreich sein, wenn alle Komponenten- und Integrationstests (Whiteboxtests) aus den entsprechenden Sprints erfolgreich waren. Bei der Durchführung der Subsystemtests der Postionsbestimmung bzw. Control-Unit muss auch gezeigt werden, dass die in Tabelle 12.26 bzw. 12.27 der Anforderungsdefinition spezifizierten Testszenarien erfolgreich absolviert werden.

12.4.3 Integrationstests

Integrationstests auf Systemebene sind durchzuführen, wenn ein Subsystem in das System integriert wird. Falls Änderungen an einem Subsysteme vorgenommen werden, muss der Integrationstest wiederholt werden. Bei der Definition und Durchführung von Integrationstests sind insbesondere die Schnittstellen zwischen dem zu integrierenden Subsystem und dem Rest des Systems zu beachten. Bei Integrationstests auf Systemebene handelt es sich um Blackboxtests.

12.4.4 Systemabnahmetest

Durch den Systemabnahmetest wird überprüft, ob der Entwicklungsstand des Projektes zum Abschluss der aktuellen Projektgruppe das Szenario ”Unfallfreie Fahrt” (siehe Kapitelabschnitt 2.4) der ersten Ausbaustufe realisiert und somit auch das in Tabelle 12.28 spezifizierte Testszenario besteht. Dies wird durch einen entsprechend umfangreichen Blackboxtest überprüft. Eine erfolgreiche Durchführung des Systemabnahmetests setzt auch voraus, dass alle Subsystem- und Integrationstests erfolgreich waren.

12.5 Statisches Testen

Techniken des Testens, welche keine Ausführung des Programmcodes benötigen, bezeichnet man als statisches Testen. Dazu zählen beispielsweise Codereviews und die Durchführung von modellbasierten Simulationen, welche in diesem Projekt zum Einsatz kommen.

12.5.1 Simulationsumgebung

Das Softwarewerkzeug *SCADE* [SCA] bietet die Möglichkeit zur Simulation. Die Eingangsparameter der Operatoren des *SCADE*-Modells können beliebig geändert werden. Die Auswirkungen dieser Eingaben können dann in den einzelnen Simulationsschritten nachvollzogen werden. Dies ist insbesondere zur Entwicklung und zum Testen der Statemachines des *SCADE*-Modells wichtig, denn deren Auswirkungen lassen sich gerade bei einer tieferen Verschachtelung und einem breiten Funktionsspektrum schwer nachvollziehen. Abgesehen von der Nachvollziehbarkeit ist es teilweise nur unter großem Aufwand möglich, jeden Zustand einer Statemachine während einer Ausführung gezielt herbeizuführen und deren Auswirkungen nachzuvollziehen.
Um neu entwickelte Regelalgorithmen durch Simulationsdurchläufe testen zu können, wird ein funktionstüchtiges Fahrzeugmodell benötigt. Dies ist mit zusätzlichen Entwicklungsaufwand verbunden, bietet dafür aber einen hohen Nutzen. Ohne eine Simulationsfunktion muss beispielsweise jede Implementierung sowie jede Veränderung der Regelungsalgorithmen direkt am Fahrzeug getestet werden und es wird eine funktionsfähige Version der Positionsbestimmungssoftware (siehe Kapitelabschnitt 10.1) und des Wrapper-Codes (siehe Kapitelabschnitt 8.4.4.1 und 10.3.1) benötigt.

12.5.2 Codereviews

Wenn ein Entwickler die Implementierung einer Softwarekomponente beendet hat, soll ein weiteres Gruppenmitglied den Programmcode begutachten und dem Entwickler ggf. Rückmeldungen über mögliche Fehlerquellen und Optimierungspotentiale geben. Auch die Implementierung von Unit-Tests sollte durch Codereviews verifiziert werden, da Fehler in der Testimplementierung das Testergebnis verfälschen können.

12.6 Testdokumentation

Die geforderten Tests müssen sorgfältig und fristgerecht von den Entwicklern durchgeführt werden. Alle Testfälle müssen vollständig und verständlich definiert und dokumentiert werden damit sie reproduzierbar sind. Die erzielten Ergebnisse der Tests müssen archiviert werden. Auch die Ergebnisse fehlgeschlagener Tests sind festzuhalten.

12.6.1 Testdatenbank

Um die Definition und Verwaltung von Testfällen zu vereinfachen wird eine Testdatenbank verwendet. Das Projekt *RCCARS* greift auf die gleiche Testdatenbank zurück, wel-

che bereits von der Projektgruppe *MIPSwarm* [Mip14] verwendet und in deren Zwischenbericht [HB13] ausführlich beschrieben wurde.

uID	ID	Bezeichnung	Testtyp	Version	Verantwortlich	Deadline	Ergebnisse
26	GT-1	Positionsbestimmung getestet	Subsystemtest	0.2	Nikolai Braeuer	2016-09-19	0 ✗
8	B01_T	➥ Kamerabild einlesen	Komponententest	0.2	Anatolij Fandrich	2016-06-23	1 ✓
9	B02_T	➥ Fahrzeugpose bestimmen	Komponententest	0.2	Anatolij Fandrich	2016-06-23	1 ✓
34	SW1.2.1	➥ Kamerabilder nachbearbeiten	Komponententest (Sprint)	0.6	Anatolij Fandrich	2016-04-30	1 ✓
35	SW1.2.2	➥ Fahrzeuge erfassen	Komponententest (Sprint)	0.6	Anatolij Fandrich	2016-04-30	1 ✓
37	SW1.2.3	➥ Fahrzeugkoordinaten ermitteln	Komponententest (Sprint)	0.4	Anatolij Fandrich	2016-04-30	1 ✓
38	SW1.2.4	➥ Ausrichtung feststellen	Komponententest (Sprint)	0.4	Anatolij Fandrich	2016-04-30	2 ✓
39	SW1.2.5	➥ Identität festellen	Komponententest (Sprint)	0.4	Anatolij Fandrich	2016-10-19	0 ✗
84	SW1.4	➥ Realzeitanforderungen einhalten	Komponententest (Sprint)	0.3	Anatolij Fandrich	2016-10-20	0 ✗
85	SW1.5	➥ Rennstreckenpose erfassen	Komponententest (Sprint)	0.1	Anatolij Fandrich	2016-06-23	1 ✓
86	SW1.6	➥ Störungen erkennen	Komponententest (Sprint)	0.2	Nikolai Braeuer	2016-10-20	0 ✗
87	SW1.6.1	➥ Verschiebung der Rennstrecke erkennen	Komponententest (Sprint)	0.4	Nikolai Braeuer	2016-06-23	2 ✓
88	SW1.6.2	➥ Störung melden	Komponententest (Sprint)	0.2	Nikolai Braeuer	2016-10-20	0 ✗
10	B03_T1	➥ Fahrzeugpose ausgeben	Komponententest	0.2	Anatolij Fandrich	2016-06-23	1 ✓

Abbildung 12.2: Ausschnitt der Testfalltabelle der Testdatenbank, Stand 01.07.2016

12.6.1.1 Definition von Testfällen

Zu Beginn der Sprintphase wurden die grundlegenden Testfälle und -Szenarien, welche in diesem Dokument in den Kapitelabschnitten 12.7 und 12.8 der beschriebenen wurden, in die Testdatenbank eingetragen. Diese basieren auf den in Kapitel 6 spezifizierten Anwendungsfällen, welche alle funktionalen Anforderungen und Rahmenbedingungen abdecken. Sobald die Implementierungsphase des Sprints abgeschlossen ist, werden Testfälle für die im Sprint zu realisierenden Anforderungen als Untertestfälle in der Testdatenbank angelegt und entsprechend zugeordnet (siehe Abbildung 12.2). Diese Testfälle sollen bis zum Ende des Sprints erfüllt werden und sind mit einer entsprechenden Deadline versehen. Die in der Testdatenbank definierten Testfälle müssen alle relevanten Informationen, welche für das Verständnis und die Reproduzierbarkeit des Testes erforderlich sind, enthalten. Dies sind insbesondere die Vorbedingungen des Testes, das auslösende

Ereignis und das erwartete Testergebnis.

12.6.1.2 Erweiterung der Testdatenbank

Es wurden Erweiterungen der Testdatenbank vorgenommen, um diese an die Bedürfnisse der Projektgruppe *RCCARS* anzupassen. Diese betreffen folgende Funktionalität.

- Anpassung der Testtypen:
  Die Testtypen wurden den Testphasen des Projektes *RCCARS* entsprechend angepasst (siehe Abschnitt 12.4). Es stehen nun Komponententest (Sprint), Integrationstest (Sprint), Abnahmetest (Sprint), Subsystemtest, Integrationstest und Systemabnahmetest zur Auswahl.

- Format der Testfall-ID:
  Das Format der Testfall-ID wurde angepasst, damit die Testfälle IDs erhalten können, welche den Bezeichnern der Anforderungen entsprechen, auf die sie sich beziehen. Auf diese Weise wird eine direkte Rückverfolgbarkeit zwischen Testfall und Anforderung hergestellt.

- Übergeordnete Testfälle sind nun abhängig von Untertestfällen:
  Im Gegensatz zur Vorgängerversion wird ein übergeordneter Testfall, auch wenn er bereits als erfolgreich getestet markiert war, nun automatisch als nicht erfolgreich (rotes X) markiert, sobald einer der Untertestfälle nicht mehr erfüllt ist. So müssen beispielsweise alle Testfälle der Positionsbestimmung erfolgreich sein, damit der ihnen übergeordnete Subsystemtest ”Positionsbestimmung vollständig getestet” erfüllt werden kann.

- Sortierbare Testfalltabelle:
  Die tabellarische Auflistung der Testfälle auf der Website der Testdatenbank ist nun sortierbar. Durch anklicken einer Spalte kann die Tabelle nach ID, Testtyp etc. sortiert werden.

12.7 Testfälle

In diesem Kapitelabschnitt sind die Testfälle beschrieben, welche benötigt werden, um die Funktionalität des System zu testen, welche durch die Anwendungsfälle aus Kapitel 6 spezifiziert wird.
Die Tabellen der Testfälle sind alle nach dem folgenden Schema aufgebaut: Die eindeutige **ID** entspricht der ID des Anwendungsfalls, von welchem der Test abgeleitet wurde. Auf diesen Anwendungsnamen folgt ein angehängtes ”_T” und eine Nummer, falls aus

dem entsprechenden Anwendungsfall mehrere Testfälle abgeleitet wurden. Die Zeile **Beschreibung** erläutert die zu testende Funktionalität und **Erwartetes Resultat** definiert die im Erfolgsfall vorliegende Situation. Der **Auslöser** beschreibt die Aktionen, welche auftreten müssen, damit der zu testende Anwendungsfall eintritt.

ID	A01_T
Anwendungsfall	System starten
Beschreibung	Der Administrator schaltet die Hardwarekomponenten des Systems ein und startet die Systemsteuerungssoftware.
Auslöser	Das System soll gestartet werden.
Erwartetes Resultat	Die Hardwarekomponenten des Systems und die Systemsteuerungssoftware wurden erfolgreich gestartet und das System ist bereit für die Inspektion.

Tabelle 12.1: Testfall System starten

ID	A02_T
Anwendungsfall	Inspektion durchführen
Beschreibung	Nachdem der Administrator das System gestartet hat, muss er eine Inspektion durchführen. Dabei werden die Positionsbestimmungssoftware und die Software der Control-Unit gestartet und es wird sichergestellt, dass alle Rahmenbedingungen eingehalten wurden. Falls die Inspektion erfolgreich war, bestätigt der Administrator dies mit Hilfe der Systemsteuerungssoftware.
Auslöser	Eine Inspektion des Systems muss durchgeführt werden.
Erwartetes Resultat	Die Positionsbestimmungssoftware sowie die Software der Control-Unit wurden erfolgreich gestartet und alle Rahmenbedingungen werden eingehalten. Nach der Bestätigung der erfolgreichen Inspektion gibt die Systemsteuerungssoftware das System zur Initialisierung frei.

Tabelle 12.2: Testfall Inspektion durchführen

ID	A03_T
Anwendungsfall	Systeminitialisierung starten
Beschreibung	Nachdem der Administrator die erfolgreiche Inspektion bestätigt hat, leitet er die Systeminitialisierung ein.

Auslöser	Der Administrator startet die Systeminitialisierung mit Hilfe der Systemsteuerungssoftware.
Erwartetes Resultat	Das System wird initialisiert und meldet dem Administrator zurück, dass die Initialisierung erfolgreich war. Danach wechselt das System automatisch in den Standby-Modus.

Tabelle 12.3: Testfall Systeminitialisierung starten

ID	A04_T
Anwendungsfall	Rennbetrieb starten
Beschreibung	Nachdem die Initialisierung erfolgreich war, startet der Administrator den Rennbetrieb.
Auslöser	Der Administrator startet den Rennbetrieb mit Hilfe der Systemsteuerungssoftware.
Erwartetes Resultat	Das Fahrzeug setzt sich in Bewegung und die Systemsteuerungssoftware informiert den Administrator darüber, dass sich das System im Rennbetrieb befindet.

Tabelle 12.4: Testfall Rennbetrieb starten

ID	A05_T
Anwendungsfall	Rennbetrieb stoppen
Beschreibung	Der Administrator beschließt den Rennbetrieb zu beenden. Daher teilt er dem System über die Systemsteuerungssoftware mit, dass der Rennbetrieb eingestellt werden soll.
Auslöser	Der Administrator beendet den Rennbetrieb mit Hilfe der Systemsteuerungssoftware.
Erwartetes Resultat	Das Fahrzeug bleibt stehen und die Systemsteuerungssoftware meldet dem Administrator, dass sich das System im Standby-Modus befindet und auf den erneuten Start des Rennbetriebs wartet.

Tabelle 12.5: Testfall Rennbetrieb stoppen

ID	A06_T
Anwendungsfall	System ausschalten

Beschreibung	Alle Soft- und Hardwarekomponenten des Systems werden ausgeschaltet.
Auslöser	Der Administrator schließt die Systemsteuerungssoftware.
Erwartetes Resultat	Falls sich das System im Rennbetrieb befand, wurde das Fahrzeug sicher angehalten. Das Fahrzeug steht still und die Software der Control-Unit sowie die Positionsbestimmungssoftware wurden von der Systemsteuerungssoftware beendet. Die Systemsteuerungssoftware und alle Hardwarekomponenten des Systems sind ausgeschaltet.

Tabelle 12.6: Testfall Manueller Notstopp

ID	A07_T
Anwendungsfall	Manueller Notstopp
Beschreibung	Der Administrator muss den Systembetrieb aufgrund einer sicherheitskritischen Störung anhalten.
Auslöser	Der Administrator versetzt das System mit Hilfe der Systemsteuerungssoftware in den Fehlerzustand.
Erwartetes Resultat	Falls sich das System im Rennbetrieb befand, wurde das Fahrzeug sicher angehalten. Das Fahrzeug steht still und das System wurde in den Fehlerzustand überführt.

Tabelle 12.7: Testfall Manueller Notstopp

ID	B01_T
Anwendungsfall	Kamerabild einlesen
Beschreibung	Die Positionsbestimmung liest mit Hilfe der Kamera ein Bild der Rennstrecke mit dem darauf befindlichem Fahrzeug ein.
Auslöser	Die Positionsbestimmungssoftware empfängt ein Bild von der Kamera.
Erwartetes Resultat	Ein Kamerabild der Rennstrecke mit dem darauf befindlichem Fahrzeug wurde eingelesen und wird von der Positionsbestimmungssoftware verarbeitet.

Tabelle 12.8: Testfall Kamerabild einlesen

ID	B02_T

Anwendungsfall	Fahrzeugpose bestimmen
Beschreibung	Die Positionsbestimmungssoftware ermittelt aus dem eingelesen Kamerabild die Pose des Fahrzeugs.
Auslöser	Die Positionsbestimmungssoftware hat ein neues Kamerabild eingelesen.
Erwartetes Resultat	Die Fahrzeugpose wurde von der Positionsbestimmungssoftware bestimmt.

Tabelle 12.9: Testfall Fahrzeugpose bestimmen

ID	B03_T
Anwendungsfall	Fahrzeugpose übermitteln
Beschreibung	Die Positionsbestimmung leitet die ermittelte Fahrzeugpose an die Control-Unit weiter.
Auslöser	Die Positionsbestimmungssoftware hat eine neue Fahrzeugpose bestimmt.
Erwartetes Resultat	Die Positionsbestimmung hat die Pose des Fahrzeugs an die Control-Unit übermittelt.

Tabelle 12.10: Testfall Fahrzeugpose ausgeben

ID	B04_T1
Anwendungsfall	Automatischer Notstopp
Beschreibung	Die Positionsbestimmung kann das Fahrzeug in mehr als zwei aufeinander folgenden Kamerabildern nicht erfassen, weil dieses das Sichtfeld der Kamera verlassen hat.
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Das Fahrzeug hat das Sichtfeld der Kamera verlassen.
Erwartetes Resultat	Das System wird in den Fehlerzustand überführt und das Fahrzeug wird gestoppt. Das System meldet dem Administrator über die Systemsteuerungssoftware, dass ein Fehler vorliegt. Nähere Informationen können dem Fehlerprotokoll entnommen werden.

Tabelle 12.11: Testfall Sichtfeld der Kamera verlassen

ID	B04_T2

Anwendungsfall	Automatischer Notstopp
Beschreibung	Die Positionsbestimmung kann das Fahrzeug nicht erfassen, weil keine aktuellen Kamerabilder eingelesen werden können.
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Die Positionsbestimmung hat die Verbindung zur Kamera verloren.
Erwartetes Resultat	Das System wird in den Fehlerzustand überführt und das Fahrzeug wird gestoppt. Das System meldet dem Administrator über die Systemsteuerungssoftware, dass ein Fehler vorliegt. Nähere Informationen können dem Fehlerprotokoll entnommen werden.

Tabelle 12.12: Testfall Verbindung zur Kamera verloren

ID	B04_T3
Anwendungsfall	Automatischer Notstopp
Beschreibung	Die Positionsbestimmung kann das Fahrzeug in mehr als zwei aufeinander folgenden Kamerabildern nicht erfassen, weil die Reflexionsmarker nicht erkennbar sind.
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Die Beleuchtungseinheit funktioniert nicht korrekt oder die Sicht auf die Reflexionsmarker ist beeinträchtigt.
Erwartetes Resultat	Das System wird in den Fehlerzustand überführt und das Fahrzeug wird gestoppt. Das System meldet dem Administrator über die Systemsteuerungssoftware, dass ein Fehler vorliegt. Nähere Informationen können dem Fehlerprotokoll entnommen werden.

Tabelle 12.13: Testfall Infrarotlampe ausgeschaltet oder defekt

ID	B04_T4
Anwendungsfall	Automatischer Notstopp
Beschreibung	Während des Rennbetriebs wird die Rennstrecke verschoben.
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Die Rennstrecke wurde nach der Initialisierung bewegt.

Erwartetes Resultat	Die Positionsbestimmung stellt fest, dass sich die Position der Rennstrecke nach der Initialisierung verändert hat. Das Fahrzeug wird gestoppt und das System wird in den Fehlerzustand überführt. Das System meldet dem Administrator über die Systemsteuerungssoftware, dass ein Fehler vorliegt. Nähere Informationen können dem Fehlerprotokoll entnommen werden.

Tabelle 12.14: Testfall Rennstrecke verschoben

ID	B04_T5
Anwendungsfall	Automatischer Notstopp
Beschreibung	Während des Rennbetriebs wird das Gerüst verschoben.
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Das Gerüst wurde nach der Initialisierung bewegt.
Erwartetes Resultat	Die Positionsbestimmung stellt fest, dass sich die Position der Rennstrecke nach der Initialisierung verändert hat. Das Fahrzeug wird gestoppt und das System wird in den Fehlerzustand überführt. Das System meldet dem Administrator über die Systemsteuerungssoftware, dass ein Fehler vorliegt. Nähere Informationen können dem Fehlerprotokoll entnommen werden.

Tabelle 12.15: Testfall Gerüst verschoben

ID	B04_T6
Anwendungsfall	Automatischer Notstopp
Beschreibung	Während des Rennbetriebs wird die Verbindung zwischen dem Rechner und der CarControl getrennt.
Auslöser	Ein sicherheitskritischer Fehlerfall ist aufgetreten. Die Verbindung zur CarControl wurde getrennt.
Erwartetes Resultat	Das Fahrzeug wird von der Software auf der CarControl gestoppt und das System wird in den Fehlerzustand überführt. Das System meldet dem Administrator über die Systemsteuerungssoftware, dass ein Fehler vorliegt. Nähere Informationen können dem Fehlerprotokoll entnommen werden.

Tabelle 12.16: Testfall Verbindung zur CarControl getrennt

ID	B05_T
Anwendungsfall	Fahrzeugpose empfangen
Beschreibung	Die Control-Unit empfängt eine Fahrzeugpose von der Positionsbestimmung.
Auslöser	Die Positionsbestimmung übermittelt eine Fahrzeugpose an die Control-Unit.
Erwartetes Resultat	Die Control-Unit hat die Fahrzeugpose empfangen und die Regelungssoftware berechnet die entsprechenden Regelungsparameter für die autonome Fahrzeugsteuerung.

Tabelle 12.17: Testfall Fahrzeugpose empfangen

ID	B06_T
Anwendungsfall	Fahrzeugregelung berechnen
Beschreibung	Die Regelungssoftware der Control-Unit hat die Pose des Fahrzeugs empfangen und berechnet daraus die Regelungsparameter für die autonome Fahrzeugsteuerung.
Auslöser	Die Control-Unit hat eine Fahrzeugpose von der Positionsbestimmung empfangen.
Erwartetes Resultat	Die Regelungssoftware hat die Parameter für die autonome Fahrzeugsteuerung gemäß der empfangenen Fahrzeugpose berechnet.

Tabelle 12.18: Testfall Fahrzeugregelung berechnen

ID	B07_T1
Anwendungsfall	Fahrzeug autonom steuern
Beschreibung	Die Control-Unit sendet die digitalen Steuerungssignale an die CarControl und diese wandelt die Signale in analoge Spannungsstärken um, welche an die Fernbedienung übergeben werden.
Auslöser	Die Regelungssoftware hat die Regelungsparameter für die autonome Fahrzeugsteuerung berechnet.
Erwartetes Resultat	Die Fernbedienung erhält die in analoge Spannungsstärken korrekt umgewandelten Steuerungssignale.

Tabelle 12.19: Testfall Steuerungssignale ausgeben

ID	B07_T2
Anwendungsfall	Fahrzeug autonom steuern
Beschreibung	Die Fernbedienung sendet die Steuerungssignale an das Fahrzeug.
Auslöser	Die in analoge Spannungsstärken umgewandelten Regelungsparameter wurden an die Fernbedienung übergeben.
Erwartetes Resultat	Das Fahrzeug wird autonom gesteuert und befährt unfallfrei den befahrbaren Bereich der Rennstrecke und erreicht dabei die geforderte Durchschnittsgeschwindigkeit.

Tabelle 12.20: Testfall Fahrzeug autonom steuern

ID	B08_T
Anwendungsfall	Auf Rennstrecke fahren
Beschreibung	Das Fahrzeug fährt auf der Rennstrecke.
Auslöser	Das Fahrzeug empfängt Steuerungssignale über die CarControl und die Fernbedienung von der Control-Unit.
Erwartetes Resultat	Das Fahrzeug setzt die Steuerungssignale korrekt um und befährt den befahrbaren Bereich der Rennstrecke.

Tabelle 12.21: Testfall Auf Rennstrecke fahren

ID	B09_T
Anwendungsfall	Infrarotlicht reflektieren (Fahrzeug)
Beschreibung	Die Beleuchtung der Positionsbestimmung bestrahlt die gesamte Rennstrecke mit Infrarotlicht. Die Reflexionsmarker auf dem Fahrzeug reflektieren das Infrarotlicht in Richtung der Kamera.
Auslöser	Die Reflexionsmarker des Fahrzeugs werden von der Beleuchtungseinheit angestrahlt.
Erwartetes Resultat	Die Pose des Fahrzeugs ist durch die Reflexionsmarker auf dem Kamerabild deutlich zu erkennen.

Tabelle 12.22: Testfall Infrarotlicht reflektieren (Fahrzeug)

ID	B10_T
Anwendungsfall	Befahrbaren Bereich abgrenzen
Beschreibung	Das Fahrzeug fährt zu Testzwecken mit voller Geschwindigkeit gegen die Fahrbahnbegrenzung.
Auslöser	Das Fahrzeug kollidiert bei voller Geschwindigkeit mit der Streckenbegrenzung.
Erwartetes Resultat	Das Fahrzeug verlässt den befahrbaren Bereich der Rennstrecke nicht, da es die Fahrbahnbegrenzung nicht überwinden kann.

Tabelle 12.23: Testfall Befahrbaren Bereich abgrenzen

ID	B11_T
Anwendungsfall	Infrarotlicht reflektieren (Rennstrecke)

Beschreibung	Die Beleuchtung der Positionsbestimmung bestrahlt die gesamte Rennstrecke mit Infrarotlicht. Die an den äußeren Ecken der Rennstrecke platzierten Reflexionsmarker reflektieren das Licht der Infrarotlampe.
Auslöser	Die Reflexionsmarker der Rennstrecke werden von der Beleuchtungseinheit angestrahlt.
Erwartetes Resultat	Die Lage der Rennstrecke ist durch die Reflexionsmarker auf dem Kamerabild deutlich zu erkennen.

Tabelle 12.24: Testfall Infrarotlicht reflektieren (Rennstrecke)

Tabelle 12.25 zeigt, dass alle Anwendungsfälle durch Testfälle abgedeckt werden. Die Zeilen der Tabelle entsprechen den einzelnen Testfällen und die Spalten den Anwendungsfällen. Der Anwendungsfall A08 zur Einhaltung des Mindestabstands der Zuschauer zur Rennstrecke wird gemeinsam mit A02 durch den Testfall A02_T abgedeckt, der die Inspektion betrifft und somit auch die Einhaltung aller Rahmenbedingungen vorsieht.

Anw. / Testf.	A01	A02	A03	A04	A05	A06	A07	A08	B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11
A01_T	X																		
A02_T		X						X											
A03_T			X																
A04_T				X															
A05_T					X														
A06_T						X													
A07_T							X												
B01_T									X										
B02_T										X									
B03_T											X								
B04_T1												X							
B04_T2												X							
B04_T3												X							
B04_T4												X							
B04_T5												X							
B04_T6												X							
B05_T													X						
B06_T														X					
B07_T1															X				
B07_T2															X				
B08_T																X			
B09_T																	X		
B10_T																		X	
B11_T																			X

Tabelle 12.25: Diese Matrix zeigt die Abdeckung aller in Kapitel 6 definierten Anwendungsfälle durch entsprechende Testfälle.

12.8 Testszenarien

In diesem Kapitelabschnitt werden Testszenarien spezifiziert, welche die Prüfung zusammenhängender Testfälle umfassen. Testfälle, die das Verhalten des Systems in einem bestimmten Fehlerfall dokumentieren, werden nicht in Testszenarien zusammengefasst.
Die Tabellen der Testszenarien sind nach dem folgenden Schema aufgebaut: Jedes Testszenario besitzt eine eindeutige **ID**. Unter **Enthaltene Testfälle** werden alle Testfälle aufgelistet, welche ein Teil des jeweiligen Testszenarios sind. Die Zeile **Beschreibung** erläutert die zu testende Funktionalität und **Erwartetes Resultat** definiert die im Erfolgsfall vorliegende Situation.

ID	TS_01
Enthaltene Testfälle	B01_T, B02_T, B03_T, B09_T, B11_T
Beschreibung	Der gesamte Ablauf der Positionsbestimmung vom Einlesen eines Kamerabildes bis zur Ausgabe der Fahrzeugpose wird durchlaufen.
Auslöser	Die Positionsbestimmungssoftware empfängt ein Bild von der Kamera.
Erwartetes Resultat	Die Positionsbestimmungssoftware gibt die Fahrzeugpose aus.

Tabelle 12.26: Testszenario Positionsbestimmung Durchlauf

ID	TS_02
Enthaltene Testfälle	B05_T, B06_T, B07_T1, B07_T2
Beschreibung	Der gesamte Ablauf der Control-Unit vom Empfang der Fahrzeugpose bis zur Übergabe der analogen Steuerungssignale an die Fernbedienung wird durchlaufen.
Auslöser	Es wird eine Fahrzeugpose an die Control-Unit übermittelt.
Erwartetes Resultat	Die Regelungsparameter für die autonome Fahrzeugsteuerung wurden gemäß der empfangenen Fahrzeugpose berechnet und an die Fernbedienung übergeben.

Tabelle 12.27: Testszenario Control-Unit Durchlauf

ID	TS_03
Enthaltene Testfälle	A01_T, A02_T, A03_T, A04_T, A05_T, A06_T, A08_T, B01_T, B02_T, B03_T, B05_T, B06_T, B07_T1, B07_T2, B08_T, B09_T, B11_T
Beschreibung	Das gesamte in Kapitelabschnitt 2.4 beschriebene Szenario ”Unfallfreie Fahrt” wird durchlaufen.
Auslöser	Das System soll gestartet werden. Daher schaltet der Administrator die Hardwarekomponenten des Systems ein und startet die Systemsteuerungssoftware.
Erwartetes Resultat	Nachdem die Inspektion durch den Administrator durchgeführt wurde, liegen keine Fehler oder Störungen vor. Die erfolgreiche Inspektion wird bestätigt und das System wird korrekt initialisiert. Der Administrator startet den Rennbetrieb mit Hilfe der Systemsteuerungssoftware und das Fahrzeug setzt sich in Bewegung. Es fährt fünf Runden lang, unfallfrei mit einer Durchschnittsgeschwindigkeit von mindestens 1,5 m/s auf dem befahrbaren Bereich der Rennstrecke. Danach beendet der Administrator den Rennbetrieb und schaltet das System aus.

Tabelle 12.28: Testszenario unfallfreie Fahrt

Tabelle 12.29 zeigt, welche Testfälle von welchen Testszenarien abgedeckt werden. Die Zeilen stellen die jeweiligen Testfälle dar. Die Spalten zeigen die zugehörigen Testszenarien. Die Testfälle A07_T und B04_T1 bis B04_T6 sowie B10_T betreffen das Verhalten des Systems im Fehlerfall und können daher nur einzeln getestet werden.

12.9 Umsetzung der Testplanung

In den vorangegangen Abschnitten dieses Kapitel wurde beschrieben, welche Arten und Technologien des Testens bei der Entwicklung des sicherheitskritischen Projektes *RCCARS* zum Einsatz kommen sollten. Nachdem sich die Anzahl der Projektgruppenmitglieder schon in der Phase der Anforderungserhebung auf vier Personen reduziert hatte, musste das Testen auf ein Mindestmaß reduziert werden, welches den verbleibenden Gruppenmitgliedern erlaubt, die durch das Szenario ”Unfallfreie Fahrt” geforderte Funktionalität zu realisieren und dabei dennoch mit Hilfe der erstellten Testfälle zu zeigen, dass die entsprechenden Anforderungen erfüllt wurden.

Testfall \ Testszenario	TS_01	TS_02	TS_03
A01_T			X
A02_T			X
A03_T			X
A04_T			X
A05_T			X
A06_T			X
A07_T			
A08_T			X
B01_T	X		X
B02_T	X		X
B03_T	X		X
B04_T1			
B04_T2			
B04_T3			
B04_T4			
B04_T5			
B04_T6			
B05_T		X	X
B06_T		X	X
B07_T1		X	X
B07_T2		X	X
B08_T			X
B09_T	X		X
B10_T			
B11_T	X		X

Tabelle 12.29: Diese Matrix zeigt die Abdeckung der Testfälle durch die Testszenarien.

12.9.1 Testarten

In den ersten beiden Sprintphasen wurden nur vereinzelte Whiteboxtests durchgeführt, da ein wesentlicher Teil des Programmcodes sich noch in einer prototypischen Entwicklungsphase befand. Stattdessen wurden die im Rahmen der Sprints definierten anforderungsspezifischen Testfälle vorerst nur empirisch durchgeführt. In der dritten und vierten Sprintphase wurde die zum Testen veranschlagte Zeit verdoppelt (siehe *ScrVm*-Planungsdokument). Jedoch musste ein Großteil dieser Zeit als Puffer verwendet werden, um die für den Sprint geforderten Anforderungen zu realisieren, so dass wiederum nur vereinzelt Whiteboxtests durchgeführt werden konnten. Stattdessen wurden vermehrt Blackboxtests vorgenommen, um zu zeigen, dass die jeweiligen Anforderungen umgesetzt wurden. Zur Dokumentation dieser Testergebnisse wurden Textausgaben und Screenshots der entsprechenden Systemkomponenten in die Testdatenbank eingetragen. Aufgrund der geringen

Anzahl der durchgeführten Whiteboxtests konnte die in Abschnitt 12.2 geforderte Anweisungsüberdeckung nicht vollständig erreicht werden.

12.9.2 Testautomatisierung

Die vereinzelt durchgeführten Whiteboxtests der Softwarekomponenten der Positionsbestimmung und des Control-Unit wurden durch Unit-Tests realisiert. Mit Hilfe der Testframeworks *CUTE* wurde eine Testsuit erstellt, mit welcher die implementierten Tests automatisch ausgeführt werden konnten. Insbesondere wurden Unit-Test für die Softwareanforderungen SW1.6, SW1.2.5 und SW2.1.1 erstellt. Die erstellten Tests und Testsuits können bei der Entwicklung weiterer Ausbaustufen der Projektes wiederverwendet und vervollständigt werden.
Die zu realisierenden Anforderungen der Systemsteuerungssoftware beziehen sich vor allem auf den Wechsel des Systemzustands. Diese sind abhängig von der Auswertung empfangener Netzwerkdatenpakte und der Interaktionen des Administrators mit dem grafischen Benutzerinterface. Eine vollständige Automatisierung dieser Eingaben ist vergleichsweise aufwändig. Daher wurden für die Systemsteuerungssoftware keine Unit-Tests implementiert. Stattdessen wurde eine als `DummyPackageGenerator` bezeichnete Modifikation des `NetworkMessengers` (siehe Kapitelabschnitt 10.2.2) entwickelt, um den Empfang von bestimmten Netzwerkdatenpaketen zu simulieren. Mit Hilfe der Methoden des `DummyPackageGenerators` werden Netzwerkdatenpakte erzeugt, mit denen gezielt Whitebox- und Performanztest durchgeführt werden können, um zu zeigen, dass die Systemsteuerungssoftware die geforderten Anforderungen erfüllt. Näheres kann der Quellcodedokumentation und der Bedienungsanleitung des Programms (siehe Anhang 14.7) entnommen werden.

Ein relevanter Aspekt des Unit-Testings ist außerdem die automatische Durchführung von Regressionstests. Bei der Entwicklung der ersten Ausbaustufe des Projektes *RCCARS* konnte die Ausführung von Regressionstests jedoch nicht merklich erleichtert werden, da bereits implementierte Tests, bedingt durch Änderungen des Quellcodes der entsprechenden Softwarekomponente, wiederholt angepasst werden mussten. Außerdem war sowohl die Anzahl der implementierten Unit-Tests, als auch die Anzahl der Sprints zu gering, um diesbezüglich Rückschlüsse ziehen zu können.

12.9.3 Teststufen

Bei der Entwicklung des Systems wurden alle in Abschnitt 12.4 beschriebenen Teststufen durchlaufen. Da das Whiteboxtesting der Systemkomponenten aus den zuvor genannten

Gründen nicht im geplanten Umfang umgesetzt werden konnte, wurden zum Ausgleich vermehrt Blackboxtests der Subsysteme sowie Integrationstests auf Systemebene durchgeführt. Die Durchführung und die Ergebnisse dieser Tests wurden mit Hilfe Logdateien, Screenshots und Testausgaben dokumentiert. Diese Daten wurden in die Testdatenbank eingetragen, um zu bestätigen, dass die entsprechenden Anforderungen erfüllt wurden.

12.9.4 Statisches Testen

Das Softwarewerkzeug *SCADE* (siehe Kapitelabschnitt 3.3) bietet zum Einen die Möglichkeit des Simulierens. Zum Anderen wird eine eigene Testumgebung inklusive der Möglichkeit eines eigenem Testprojekts zur Verfügung gestellt.
Es wurde ein entsprechendes *SCADE* Testprojekt erstellt und verwendet um den *DirectionDetection* Operator zu testen (siehe Kapitelabschnitt 10.3.3.5). Dieser Operator überprüft die Fahrtrichtung des Fahrzeugs und löst einen Fehler aus, sobald das Fahrzeug in die falsche Richtung fährt (siehe SW2.1.10.3). Im Weiteren wurde die State-Machine im *RootNode* Operator überprüft (siehe Kapitelabschnitt 10.3.3). Dieser Zustandsautomat kontrolliert kontinuierlich, ob das entsprechende Fahrzeug gefunden wurde. Wurde das Fahrzeug nicht gefunden, wird ein Fehler ausgelöst (siehe SW2.1.10.4). Die verhältnismäßig zeitintensive Erstellung dieser Modelltests, bietet die Möglichkeit sehr hochwertige Testfälle zu erstellen, indem beispielsweise die Eingangsbelegungen einer State-Machine manuell vorgegeben werden und die erwarteten Werte und aktiven Zustände in bestimmten Ausführungszyklen explizit vorgegeben werden. Zusätzlich können entsprechende Testberichte generiert werden.
Für die Erstellung weiterer Testfälle wäre nicht nur mehr Zeit, sondern auch ein funktionsfähiges Fahrzeugmodell erforderlich gewesen. Zum Ende des vierten Sprints wurde zwar ein entsprechendes Fahrzeugmodell entwickelt (siehe Kapitelabschnitt 10.3.3.14), jedoch erlaubten der Mangel an personellen Ressourcen und die zu diesem Zeitpunkt noch ausstehenden Dokumentationsarbeiten es nicht mehr dieses Modell in das System zu integrieren und ausreichend zu testen. Daher war auch dessen Nutzung zu Simulationszwecken im Rahmen der ersten Ausbaustufe des Projektes *RCCARS* nicht mehr möglich. Das Modell steht jedoch für Folgeprojekte bereit.
Wie zuvor bereits angesprochen bietet *SCADE* auch die Funktion des Simulierens. Bei dem Simulieren können interaktiv am Modell einzelne Eingänge belegt und die Auswirkungen im Modell beobachtet werden. Diese Funktion ähnelt dem Debuggen bei der herkömmlichen Softwareentwicklung. Jeder in *SCADE* entwickelte Operator wurde vor der Generierung des Codes durch ausgiebige Simulationsläufe getestet.

Codereviews wurden insbesondere zur Fehlersuche und bei kritischen Implementierungs-

entscheidungen durchgeführt. Umfassende Codereviews sämtlicher entwickelter Softwarekomponenten wären aufgrund des Umfangs des Quellcodes mit den verfügbaren personellen Ressourcen nicht zu leisten gewesen.

12.9.5 Testdokumentation

Neben den in den Abschnitten 12.7 und 12.8 definierten grundlegenden Testfällen und Testszenarien, welche die in Kapitel 6 beschriebenen Anwendungsfälle des Szenarios ”Unffalfreie Fahrt” abdecken, wurden zur Testdokumentation weitere Testfälle für alle System-, Software-, und Nicht-Funktionalen Anforderungen in der Testdatenbank erstellt. Für die Hardwarebasisanforderungen wurden ebenfalls Testfälle angelegt. Diese verweisen auf die entsprechenden Abschnitte des Kapitels 7, in welchen bereits gezeigt wurde, dass die Hardwarebasisanforderungen inkl. aller Unteranforderungen erfüllt wurden.

Sys1.3	➥ Control-Unit vollständig getestet	Subsystemtest	0.3	Tom Reske	2016-10-20	2 ✓
HW3	➥ Contol-Unit-Hardware geeignet	Abnahmetest (Sprint)	0.3	Tom Reske	2016-10-20	1 ✓
NHW1	➥ CarControl selbst konstruiert	Abnahmetest (Sprint)	0.2	Michael Bukowski	2016-10-20	1 ✓
SW2	➥ Softwarekomponenten der Contol-Unit vollständig und korrekt	Abnahmetest (Sprint)	0.3	Tom Reske	2016-10-20	2 ✓
SW2.1	➥ Regelungssoftware vollständig und korrekt	Integrationstest (Sprint)	0.2	Tom Reske	2016-10-20	2 ✓
SW2.1.01	➥ Regelungssoftware kann Daten der Positionsbestimmung empfangen	Integrationstest	0.4	Michael Bukowski	2016-05-12	3 ✓
SW2.1.01_T1	➥ Initialisierung eines UDP-Receivers	Komponententest (Sprint)	0.2	Michael Bukowski	2016-10-20	2 ✓
SW2.1.01_T2	➥ Empfang von Fahrzeugdaten	Komponententest (Sprint)	0.3	Michael Bukowski	2016-10-20	2 ✓
SW2.1.01_T2.1	➥ Speicherung der Fahrzeugdaten im Ringpuffer	Komponententest (Sprint)	0.2	Michael Bukowski	2016-10-20	1 ✓
SW2.1.01_T3	➥ Deinitialisierung eines UDP-Receivers	Komponententest (Sprint)	0.1	Michael Bukowski	2016-10-20	1 ✓
SW2.1.02	➥ Regelungsparameter berechnen	Komponententest (Sprint)	0.4	Tom Reske	2016-10-20	1 ✓
SW2.1.03	➥ Fahrzeug entlang einer vorgegebenen Route steuern	Integrationstest (Sprint)	0.3	Tom Reske	2016-10-20	1 ✓
SW2.1.04	➥ Laden des Verlaufes der Rennstrecke und Zieltrajektorie	Komponententest (Sprint)	0.2	Michael Bukowski	2016-06-24	1 ✓
SW2.1.04_T1	➥ Öffnen einer CSV Datei	Komponententest (Sprint)	0.2	Michael Bukowski	2016-06-24	1 ✓
SW2.1.04_T2	➥ Ablegen der Informationen aus der Datei in das SCADE-MapArray	Komponententest (Sprint)	0.2	Michael Bukowski	2016-06-24	1 ✓
SW2.1.05	➥ Fahrzeug gegen den Uhrzeigersinn steuern	Komponententest (Sprint)	0.3	Tom Reske	2016-08-26	1 ✓
SW2.1.06	➥ Mindestdurchschnittsgeschwindigkeit	Integrationstest (Sprint)	0.2	Tom Reske	2016-10-20	2 ✓
SW2.1.07	➥ Kurvengeschwindigkeit von 1,2 m/s nicht überschreiten	Integrationstest	0.4	Tom Reske	2016-10-20	2 ✓
SW2.1.08	➥ Fünf Runden lang fahren	Komponententest (Sprint)	0.2	Tom Reske	2016-10-20	1 ✓
SW2.1.09	➥ Regelungssoftware übermittelt Regelungsdaten an CarControl	Integrationstest	0.1	Michael Bukowski	2016-06-24	1 ✓
SW2.1.10	➥ Regelungssoftware erkennt Störungen und Unfälle	Integrationstest (Sprint)	0.4	Michael Bukowski	2016-10-20	1 ✓
SW2.1.10.1	➥ Erkennung des Verlassens des befahrbaren Bereichs	Komponententest (Sprint)	0.5	Tom Reske	2016-06-24	2 ✓

Abbildung 12.3: Ausschnitt der Testfalltabelle der Testdatenbank zum Abschluss des Projektes (auf Anforderungsspezifikation basierte Struktur).

Um die Testdatenbank sowie die daraus generierte Testdokumentation übersichtlich zu gestalten und die Rückverfolgbarkeit der Testfälle auf die entsprechenden Anforderungen zu vereinfachen, wurden alle Testfälle, welche aus den einzelnen Anforderungssätzen hervorgingen in eine separate Hierarchie ausgelagert. Abbildung 12.3 zeigt, dass deren Aufbau der Struktur der Anforderungsspezifikation entspricht. Das bedeutet, dass sich auf

oberster Ebene die Basissystemanforderungen befinden, welche die Systemabnahmetests darstellen. Von diesen leiten sich alle Subsystemtests ab, welche wiederum in Integrationstests und sprint-interne Abnahme- und Komponententests zerfallen. Als ID dieser Testfälle wird die ID des jeweiligen Anforderungssatzes verwendet. Der Bezug zu den grundlegenden Testfällen und -Szenarien wurde dabei über das Feld ”Bezug” der jeweiligen Testfälle festgehalten. Die auf den Anwendungsfällen basierenden Testfälle und die

SST-1	Subsystem 1 (Fahrzeug) getestet	Subsystemtest	0.5	Michael Bukowski	2016-10-20	1 ✓
B08_T	➥ Auf Rennstrecke fahren	Integrationstest	0.4	Michael Bukowski	2016-10-20	1 ✓
B09_T	➥ Infrarotlicht reflektieren (Fahrzeug)	Integrationstest	0.2	Michael Bukowski	2016-06-23	1 ✓
SST-2	Subsystem 2 (Positionsbestimmung) getestet	Subsystemtest	0.3	Anatolij Fandrich	2016-09-20	1 ✓
B04_T1	➥ Sichtfeld der Kamera verlassen	Abnahmetest (Sprint)	0.3	Anatolij Fandrich	2016-10-20	1 ✓
B04_T2	➥ Verbindung zur Kamera verloren	Abnahmetest (Sprint)	0.3	Anatolij Fandrich	2016-10-20	2 ✓
B04_T3	➥ Infrarotlampe ausgeschaltet oder defekt	Abnahmetest (Sprint)	0.3	Anatolij Fandrich	2016-10-20	1 ✓
B04_T4	➥ Rennstrecke verschoben	Abnahmetest (Sprint)	0.3	Anatolij Fandrich	2016-10-20	1 ✓
B04_T5	➥ Gerüst verschoben	Abnahmetest (Sprint)	0.3	Anatolij Fandrich	2016-10-20	1 ✓
TS_01	➥ Testszenario Positionsbestimmung	Integrationstest	0.3	Anatolij Fandrich	2016-10-20	1 ✓
B01_T	➥ Kamerabild einlesen	Abnahmetest (Sprint)	0.3	Anatolij Fandrich	2016-06-23	2 ✓
B02_T	➥ Fahrzeugpose bestimmen	Abnahmetest (Sprint)	0.2	Anatolij Fandrich	2016-06-23	1 ✓
B03_T	➥ Fahrzeugpose ausgeben	Integrationstest	0.1	Anatolij Fandrich	2016-06-23	1 ✓
SST-3	Subsystem 3 (Control-Unit) getestet und korrekt	Subsystemtest	0.4	Tom Reske	2016-10-20	2 ✓
B04_T6	➥ Verbindung zur CarControl getrennt	Abnahmetest (Sprint)	0.5	Michael Bukowski	2016-10-20	1 ✓
TS_02	➥ Testszenario Control-Unit	Integrationstest	0.2	Tom Reske	2016-10-20	1 ✓
B05_T	➥ Fahrzeugpose empfangen	Integrationstest	0.6	Michael Bukowski	2016-06-24	1 ✓
B06_T	➥ Fahrzeugregelung berechnen	Abnahmetest (Sprint)	0.6	Tom Reske	2016-10-20	1 ✓
B07_T1	➥ Steuerungssignale ausgeben	Integrationstest	0.3	Michael Bukowski	2016-10-20	1 ✓
B07_T2	➥ Fahrzeug autonom steuern	Integrationstest	0.5	Tom Reske	2016-10-20	2 ✓
SST-4	Subsystem 4 (Rennstrecke) getestet	Subsystemtest	0.4	Nikolai Braeuer	2016-06-23	2 ✓
B10_T	➥ Befahrbaren Bereich abgrenzen	Integrationstest	0.2	Michael Bukowski	2016-06-23	1 ✓
B11_T	➥ Infrarotlicht reflektieren (Rennstrecke)	Integrationstest	0.1	Anatolij Fandrich	2016-10-20	1 ✓

Abbildung 12.4: Ausschnitt der Testfalltabelle der Testdatenbank zum Abschluss des Projektes (auf grundlegenden Testfällen und -Szenarien basierte Struktur).

Testszenarien der Positionsbestimmung und Control-Unit wurden separaten Subsystemtests zugeordnet (siehe Abbildung 12.4). Der Systemabnahmetest für das Testszenario ”Unfallfreie Fahrt” wird einzeln aufgeführt und die Referenzen auf die o. g. Subsystemtests wurden im Feld ”Bezug” eingetragen.
Durch diese Vorgehensweise wurde eine vollständige Rückverfolgbarkeit für alle Anforderungen erreicht, welche sich vom Szenario ”Unfallfreie Fahrt” ausgehend über die grundlegenden Anwendungsfälle und die daraus abgeleiteten funktionalen Systemanforderungen bis hin zu den spezifischen Soft- und Hardwareanforderungen und den jeweiligen Testfällen erstreckt.

12.10 Testergebnisse

Die in die Testdatenbank eingetragenen Ergebnisse zeigen, dass bis auf eine Ausnahme alle Testfälle erfolgreich durchgeführt wurden. Die Ausnahme betrifft den Test zum Ausführen der Softwarekomponenten mit höchster Priorität. Da das entwickelte System bereits alle Realzeitanforderungen erfüllt war diese Priorisierung für das Erreichen des Gesamtziels nicht erforderlich. Aufgrund des Zeitmangels, wurden daher alle personellen Ressourcen für die Umsetzung anderer Anforderungen benötigt, welche für die Erfüllung des Systemabnahmetest relevant waren.
Des Weiteren musste ein Kompromiss bzgl. der geforderten Mindestdurchschnittsgeschwindigkeit eingegangen werden. Bedingt durch das Fahren auf der Mittellinie erreicht das Fahrzeug nicht die Mindestdurchschnittsgeschwindigkeit von 1,5 m/s. Da das Fahren auf Mittellinie und somit das Einhalten eines sicheren Abstands zur Fahrbahnbegrenzung ein entscheidender Sicherheitsfaktor im System ist, wurde nach Absprache mit den Auftraggebern des Systems beschlossen, dass eine Durchschnittsgeschwindigkeit von 1,2 m/s akzeptabel ist. Die zum Ende der ersten Ausbaustufe des Projektes *RCCARS* erreichte Durchschnittsgeschwindigkeit lag bei der Durchführung der entsprechenden Tests oberhalb von 1,3 m/s (siehe auch Kapitelabschnitt 13.1). Somit wird die Durchführung dieser Tests als erfolgreich angesehen.
Folgende Matrix 12.30 zeigt wie die durchgeführten Tests die in den Abschnitten 12.7 und 12.8 definierten grundlegenden Testfällen und Testszenarien abdecken.

Anw. / Testf.	A01	A02	A03	A04	A05	A06	A07	A08	B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11
A01_T	✓																		
A02_T		✓						✓											
A03_T			✓																
A04_T				✓															
A05_T					✓														
A06_T						✓													
A07_T							✓												
B01_T									✓										
B02_T										✓									
B03_T											✓								
B04_T1												✓							
B04_T2												✓							
B04_T3												✓							
B04_T4												✓							
B04_T5												✓							
B04_T6												✓							
B05_T													✓						
B06_T														✓					
B07_T1															✓				
B07_T2*															✓				
B08_T																✓			
B09_T																	✓		
B10_T																		✓	
B11_T																			✓
TS_01									✓	✓	✓						✓		✓
TS_02*													✓	✓	✓				
TS_03*	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓		✓

Tabelle 12.30: Diese Matrix zeigt, dass die grundlegenden Testfälle und Testszenarien, welche alle in Kapitel 6 definierten Anwendungsfälle abdecken, erfolgreich durchgeführt wurden. *Nach Absprache mit den Auftraggebern des Systems wurde beschlossen, dass eine Durchschnittsgeschwindigkeit von 1,2 m/s akzeptabel ist.

12.11 Fazit

Als abschließendes Fazit ist festzustellen, dass der Aufwand des Testens eines sicherheitskritisches System auch bei Projekten von mittlerer Komplexität bereits mindestens so viel Zeit erfordert wie die Implementierung der Funktionalität. Obwohl es bei der Testplanung bereits vorgesehen war, den Testaufwand zu reduzieren, konnte dennoch nicht im geplanten Umfang getestet werden. Whiteboxtests wurden lediglich exemplarisch durchgeführt, um im Rahmen dieses vorwiegend didaktischen Projektes zu zeigen, das die entsprechenden Techniken und Fertigkeiten von den Mitgliedern der Projektgruppe bewältigt werden können. Ersatzweise wurden daher vermehrt Blackboxtests und Integrationstests auf Systemebene durchgeführt, um nachzuweisen, dass die geforderten Anforderungen dennoch realisiert wurden. In Folgeprojekten sollten fehlende Whiteboxtests jedoch nachgeholt

werden, sobald entsprechende Ressourcen zur Verfügung stehen.
Das Projekt profitierte von einer detaillierten Anforderungsspezifikation, welche es im Zusammenspiel mit der Testdatenbank ermöglichte genau nachzuweisen, dass das Projektziel erreicht wurde.

13 Ergebnisse und Ausblick

Dieses Kapitel befasst sich mit den Ergebnissen des Projekts *RCCARS*, wobei der erreichte Entwicklungsfortschritt aufgezeigt wird. Im Weiteren wird ein Fazit über den gesamten Projektverlauf gezogen und im Anschluss ein Ausblick für mögliche zukünftige Entwicklungsarbeiten am System gegeben.

13.1 Ergebnisse

Der Projektgruppe *RCCARS* wurde die Aufgabe gestellt, das Szenario unfallfreie Fahrt umzusetzen (siehe Kapitelabschnitt 2.3). Um dieses Szenario zu realisieren, musste ein einzelnes Fahrzeug auf einer geschlossenen Rennstrecke (siehe Kapitelabschnitt 4.4) autonom mit einer Mindestdurchschnittsgeschwindigkeit von 1,5 $\frac{m}{s}$ fünf Runden ohne eine Kollision mit der Fahrbahnbegrenzung fahren.

Diese Aufgabe wurde größtenteils umgesetzt. Das System ist fähig, ein Fahrzeug autonom auf einer geschlossenen Rennstrecke ohne eine Kollision mit der Fahrbahnbegrenzung fahren zu lassen. Lediglich im Hinblick auf die Mindestdurchschnittsgeschwindigkeit von 1,5 $\frac{m}{s}$ gab es Einschränkungen bei dem Erreichen des Projektziels. Das System ist in der Lage das Fahrzeug stabil mit einer Durchschnittsgeschwindigkeit von über 1,2 $\frac{m}{s}$ für mindestens fünf Runden am Stück fahren zu lassen. Je nach Akkuladestand und sonstigen Umwelteinflüssen (Temperatur, Regelgüte,...) werden bis zu 1,4 $\frac{m}{s}$ erreicht. Da dieser Wert jedoch nicht stabil erreicht werden kann, schließt die Projektgruppe die Entwicklung mit einer erreichten Durchschnittsgeschwindigkeit von über 1,2 $\frac{m}{s}$ ab.
Die Zielgeschwindigkeit konnte unter anderem bedingt durch die Position der Trajektorie - in der Mitte der Rennstrecke - nicht erreicht werden. Die Kurven sind so zu eng. Die Performance hätte voraussichtlich durch den Ersatz der Mittellinie zur Ideallinie erhöht werden können. Weiteren konnte zwar ein Fahrzeugmodell implementiert, jedoch nicht verwendet werden. Die Implementierung wurde erst kurz vor Ende der Projektgruppe abgeschlossen. Außerdem konnte zeitlich bedingt keine modellprädiktive Regelung (Model Predictive Control, kurz MPC) erarbeitet werden. Diese MPC wäre von großem Vorteil, da der gesamte Regelkreis mit Latenzen belastet ist (siehe Kapitelabschnitt 11). Anhand

der modellprädiktiven Regelung könnten die Latenzen sowie die Trägheit des Fahrzeugs und der Regler im gewissen Maße ausgeglichen und somit die Performance erhöht werden. Weitere Ansätze zur Optimierung am gesamten System sind im Kapitelabschnitt 13.3 zu finden.

Die Funktionalität des Systems wird weiterhin durch die Testdatenbank sowie den entsprechenden Testbericht nachgewiesen, worin alle erfüllten und getesteten Anforderungen inklusive dem Testergebnis aufgeführt sind. Durch die Testdatenbank wurde eine vollständige Rückverfolgbarkeit aller Anforderungen erreicht.

Die letztendliche Validierung und Endabnahme des Systems findet am 10. November 2016 durch die Erstgutachter Herr Prof. Damm und Herr Prof. Fränzle statt. An diesem Datum ist die Abschlusspräsentation der Projektgruppe *RCCARS*. Das Team wird den gesamten Entwicklungsstand anhand des entwickelten Demonstrators präsentieren, woraufhin den Auftraggebern (Erstgutachtern) die letztendliche Systemabnahme übertragen wird.

13.2 Fazit

13.2.1 Vorgehensmodell

Zum verwendeten Vorgehensmodell *ScrVm* (siehe Kapitelabschnitt 3.1.2) soll an dieser Stelle ein Fazit gezogen werden. Es kann festgehalten werden, dass sich das gewählte Vorgehensmodell *ScrVm* durch die Vorteile im oben genannten Kapitelabschnitt als sehr gut anwendbar für den hier gegebenen Kontext der Projektgruppe erwiesen hat. Insbesondere von der Agilität des Vorgehensmodells konnte profitiert werden. Bedingt durch die sehr geringe Personalkapazität, war es wichtig schnell und effizient auf Änderungen und Vorfälle - wie beispielsweise plötzlicher Personaleinbruch - reagieren zu können. Zudem fiel auf, dass während der Sprints neue Anforderungen aufkamen und diese durch das Vorgehensmodell neu erhoben und in den Product Backlog mit aufgenommen werden konnten. Die gesamte Dynamik von Scrum sowie die Eigenschaften des V-Modells durch ständiges Testen brachten der Projektgruppe einen großen Vorteil. Abschließend kann somit festgehalten werden, dass das Vorgehensmodell *ScrVm* eine gute Wahl und bei einer Projektneuplanung wieder zum Einsatz kommen würde.

13.2.2 Zeit- und Sprintmanagement

In diesem Abschnitt soll das Zeit- und Sprintmanagement betrachtet werden. Für das Zeitmanagement wurde der Projektverlauf in drei verschiedene Phasen eingeteilt. Die Einarbeitungs- und Anforderungsdefinitionsphase sowie in zwei Sprintphasen. Die Abbildung 13.1 vedeutlicht dies noch einmal.

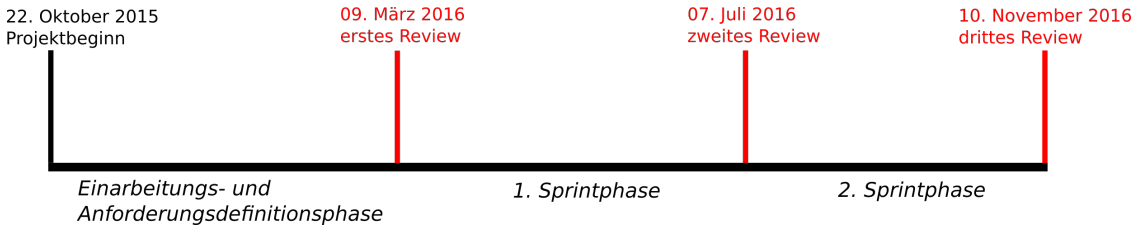


Abbildung 13.1: Zeitleiste, welche den Projektverlauf in Phasen einteilt.

In der Phase vom Beginn des Moduls Projektgruppe bis zum ersten Review befand sich die Projektgruppe in der Einarbeitungs- und Anforderungsdefinitionsphase. In dieser Phase wurde jedem Projektgruppenmitglied ein Referenzprojekt zugewiesen, in das sich eingearbeitet werden sollte. Im Weiteren wurde aus den erforschten Ergebnissen die benötigte Hardware für die Projektgruppe *RCCARS* abgeleitet. Im Anschluss wurden Anforderungen für den Bericht zum ersten Review an das Gesamtsystem erhoben. Rückwirkend lässt sich festhalten, dass die Zeit für das Erheben der Anforderungen zu knapp bemessen worden ist und die Projektgruppe zum Ende der Einarbeitungs- und Anforderungsdefinitionsphase mit Zeitmangel zu kämpfen hatte. Es lässt sich somit resümieren, dass bei erneuter Projektplanung hierfür mehr Zeit eingeplant werden sollte. Dennoch wurden die gesetzten Ziele in der Einarbeitungs- und Anforderungsdefinitionsphase eingehalten. Darüber hinaus sind die mangelnden Ressourcen beim Schreiben des ersten Berichts (siehe *Anforderungsdefinition vom 09. März 2016*) deutlich aufgefallen. Das Übertragen bestimmter Aufgaben auf ein einzelnes Projektgruppenmitglied, wie beispielsweise das Erstellen von Grafiken, wäre von großem Vorteil gewesen

In der Phase zwischen dem ersten und dem zweiten Review, also der ersten Sprintphase wurden die vorhandenen Ressourcen der Projektgruppe aufgeteilt, um einen maximalen Fortschritt erzielen zu können. Zur Verdeutlichung wird die bereits im Kapitel zum Projektmanagement aufgeführte Grafik (siehe Abbildung 13.2) herangezogen.

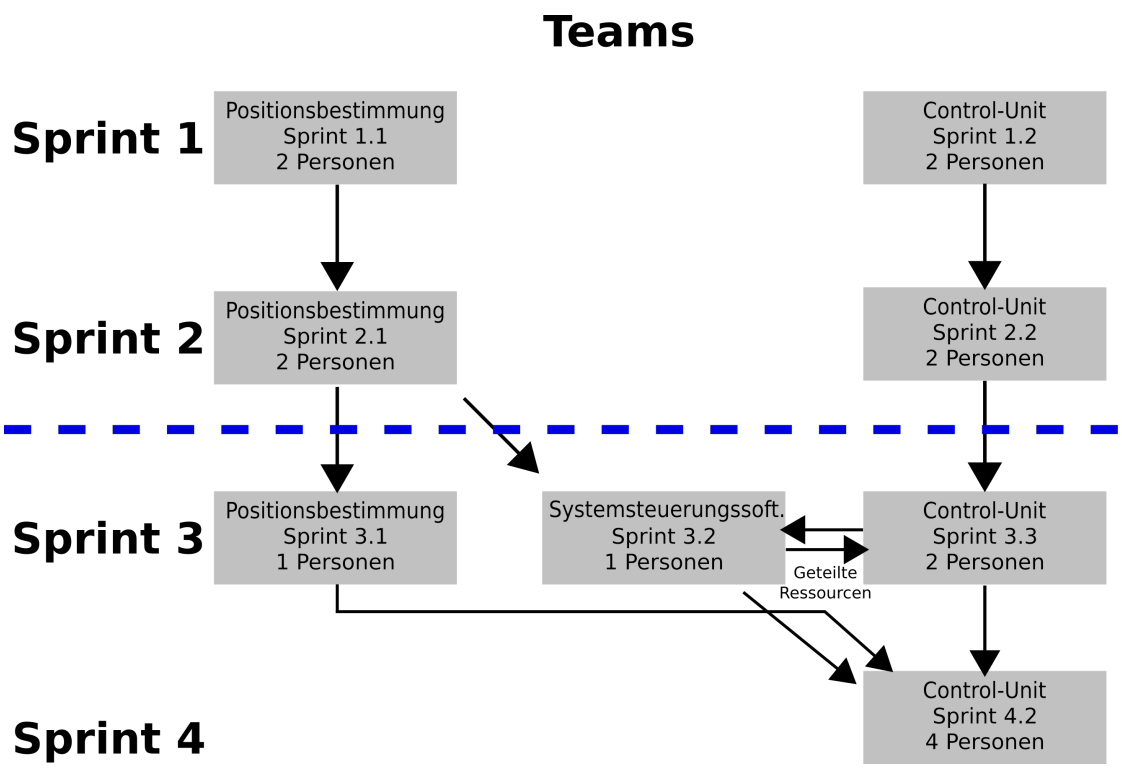


Abbildung 13.2: Sprintplanung vom ersten bis zum dritten Review. Oberhalb der blau gestrichelten Linie findet sich die erste Sprintphase, also die Phase nach dem ersten Review wieder, welche an die Einarbeitungs- und Anforderungsdefinitionsphase anschloss. Die Zeit unterhalb der Linie stellt die Zeit zwischen dem zweitem und dritten Review dar. Die blaue Linie selbst beschreibt den Zeitpunkt des zweiten Reviews (07. Juli 2016).

Die Sprints 1 und 2 wurden jeweils mit etwa sieben Wochen angesetzt. Diese Zeit erwies sich als gerade noch akzeptabel. Bedingt durch Personalmangel wurde festgestellt, dass sich in der folgenden Sprintphase (2. Sprintphase), die sich unterhalb der blau gestrichelten Linie befindet, eine Erhöhung der Sprintdauer anbieten würde. Letztendlich war es in der ersten Sprintphase jedoch nicht möglich die Sprintdauer noch weiter zu erhöhen, da ansonsten keine Zeit mehr für den zweiten Bericht (siehe *Architektur und Schnittstellenspezifikation vom 07. Juni 2016*) übrig geblieben wäre. Letzendlich konnten nicht alle gesetzten Anforderungen in jedem Sprint gehalten werden. Dazu zählt Sprint 1.1, also der erste Sprint des Teams Positionsbestimmung. Innerhalb dieses Sprints war es dem Team nicht möglich, die Ausrichtung des Fahrzeugs korrekt festzustellen, obwohl dies im Sprint-Backlog vorgesehen war. Dies hing mit dem unterschätzen Arbeitsaufwand zu Bewältigung dieser Aufgabe zusammen. Diese nicht erfüllte Anforderung wurde somit höher priorisiert (siehe Kapitelabschnitt 3.5) und in dem zugehörigen Folgesprint aufgenommen und dort auch erfolgreich erfüllt.

Darüber hinaus konnte die Anforderung, dass die Identität der Fahrzeuge anhand der Anordnung der Reflexionsmarker festgestellt werden kann nicht realisiert werden. Die Probleme, welche die Abwicklung dieser Anforderung erschwerten, konnten nur mühsam beseitigt werden. Diese Anforderung wurde höher priorisiert und in Sprint 2.1, also den zweiten Sprint des Teams Positionsbestimmung neu aufgenommen. In Sprint 2.2 wurde dann ein Ansatz implementiert, der es zulässt ausschließlich ein einziges Fahrzeug auf der Rennstrecke identifizieren zu können. Das Ziel war dennoch im selben Sprint einen Ansatz zu implementieren, der es zulässt mehrere Fahrzeuge zu detektieren. Zum Zeitpunkt, als dieser neue Ansatz entwickelt werden sollte brach jedoch die Arbeitskapazität eines Teammitglieds krankheitsbedingt um rund 50 Prozent ein. Dadurch standen dem Team Positionsbestimmung nur noch rund 75 Prozent Kapazität zur Verfügung. Dies führte dazu, dass auch in diesem Sprint die korrekte Umsetzung der zuvor genannten Anforderung nicht möglich war. Das wiederum hatte zur Folge, dass zum Zeitpunkt des zweiten Reviews (siehe blaue Linie in Abbildung 13.2) nur ein Fahrzeug gleichzeitig detektiert werden konnte. Die vollständige Umsetzung eines Ansatzes, der es zulässt die Anforderung SW1.2.5 zu erfüllen und so bis zu 16 Fahrzeuge eindeutig zu identifizieren, wurde in der Zeitphase vom zweiten Review bis zum dritten Review umgesetzt. Im Weiteren war die Positionsbestimmungssoftware zu diesem Zeitpunkt noch nicht effizient genug programmiert.
Das Team Control-Unit konnte die angezielten Anforderungen zum zweiten Review umsetzen.

Es lässt sich somit festhalten, dass der geschätzte Arbeitsaufwand, insbesondere bei der Entwicklung der Positionsbestimmung, höher war als gedacht. Bei erneuter Planung sollte, falls möglich, die Sprintlänge der Positionsbestimmungssoftware erhöht werden oder ein zusätzlicher Sprint zur Optimierung eingeführt werden.

In der zweiten Sprintphase (siehe unterhalb der blauen Linie in Abbildung 13.2) wurde aufgrund der Erfahrungen in der ersten Sprintphase die Sprintlaufzeiten auf rund acht Wochen erhöht. Dies ergab gleichzeitig einen Zeitpuffer zum Ende der Projektgruppe, um den hier vorliegenden Endbericht zu vervollständigen. Im Gegensatz zu den vorherigen Phasen wurde durch die Projektleitung in dieser Sprintphase das Dokumentieren der Softwareentwicklung zeitlich eingeplant und vorgeschrieben, um so zum Ende der Projektgruppe einen größeren Zeitraum zur Vervollständigung des Endberichts zu bieten.
Der Zeitplan konnte jedoch nicht eingehalten werden. Die Fertigstellung und Optimierung der Positionsbestimmungssoftware hat einen so hohen Zeitaufwand in Anspruch genommen, dass die Sprintlänge insgesamt verdoppelt werden musste. Somit viel das

Teammitglied der Positionsbestimmungssoftware, dass ab Sprint vier bei der Entwicklung der Control-Unit eingesetzt werden sollte, aus. Auch die Entwicklung der Systemsteuerungssoftware konnte nicht innerhalb des geplanten Zeitraums eingehalten werden. Die Laufzeit von Sprint 3.2 (Systemsteuerungssoftware) musste insgesamt um zwei Wochen verlängert werden.
Diese unvorhergesehenen zum Teil massiven Verlängerungen der Sprintlaufzeiten führten zu einem enormen Engpass in der Entwicklung der Control-Unit. In der zweiten Sprintphase der Control-Unit ging es um die Implementierung der Regelungsalgorithmen.

Zusammenfassend lässt sich zur zweiten Sprintphase sagen, dass durch die erhöhten Sprintlaufzeiten und dem unterschätzten Implementierungsaufwand die eingeplante Dokumentierung der Softwareentwicklung nicht realisiert werden konnte. Insbesondere bei der Control-Unit trat ein so ernstzunehmender Ressourcenengpass auf, dass das Dokumentieren der Entwicklungsfortschritte vollständig ausgesetzt werden musste. Dies führte wiederum in der Endphase der Projektgruppe zu einem Zeitkonflikt bei der Erstellung des Ihnen hier vorliegenden Endberichts.
Es steht nun zur Diskussion, ob eine andere Projektplanung mehr Pufferzeiten für die Anfertigung der Berichte ermöglicht hätte. Dem kann jedoch entgegengebracht werden, dass die Projektleitung die höchste Priorität auf die Erfüllung der Aufgabenstellung gelegt hat und der zeitliche Rahmen für andere Tätigkeiten entsprechend niedrig gehalten wurde. Ein anderer Zeitplan mit der Hauptaufmerksamkeit auf die Anfertigung der Berichte hätte beispielsweise einen größeren zeitlichen Puffer ermöglicht. Allerdings hätte das Projektziel mit einer solchen Strategie nicht erreicht werden können. Nichts desto trotz muss stetig abgewogen werden, ob eine Reduktion der Zeit für die Implementierung und eine Inkrementierung der Zeit für die Berichte von Vorteil gewesen wäre.

13.2.3 Projektverlauf

In diesem Kapitelabschnitt soll ein kritisches Fazit darüber gezogen, mit welchen Problemen die Projektgruppe zu tun hatte und an welchen Stellen die Projektplanung hätte verbessert werden können.
Bereits in der ersten Implementierungsphase stellte die Projektgruppe fest, dass eine noch tiefere Literaturrecherche in den Referenzprojekten der Universitäten, die sich mit dem gleichen Themengebiet beschäftigt haben, von Vorteil gewesen wäre. Daraus resultierte, dass einige Entwicklungswege zuerst in Sackgassen führten, die durch eine gründlichere Recherche hätten umgangen werden können. Durch ein gründlicheres und vor allem tieferes Einarbeiten in die Referenzpaper hätte in der späteren Entwicklung Zeit eingespart werden können.

Als nächster wichtiger Punkt ist die Anforderungserhebung zu erwähnen. Deren Erstellungsaufwand und vor allem deren Wichtigkeit wurde deutlich unterschätzt. Der Projektgruppe wurde in den Sprintphasen schnell bewusst, welche Widersprüche durch nicht sauber oder vollständig definierte Anforderungen entstehen können. Folgen wie beispielsweise nachträgliche Implementierungen von Funktionen oder deutliche Änderungen in der Funktionsweise des Systems wurden unterschätzt. Auch an dieser Stelle lässt sich festhalten, dass die Projektgruppe daraus gelernt hat und fehlende oder unvollständig definierte Anforderungen stetig nachgebessert hat.

Ein Weiterer und einer der wichtigsten Punkte war das Definieren der Schnittstellen. Der Projektgruppe wurde bewusst, welche Folgen es mit sich bringt die Schnittstellen nicht optimal zu definieren. Der Aufwand in der Nachbesserung an Schnittstellen stellte sich als extrem hoch heraus. Viele Missverständnisse und Nachimplementierungen hätten durch eine noch sauberere Definierung der Schnittstellen vermieden werden können.

Zusätzlich ist der Aufwand des Testens unterschätzt worden. Die Projektgruppe hat von Beginn an viel Wert auf das Testen gelegt und hat diesem Teil des Entwicklungsprozesses große Beachtung geschenkt. Der dabei entstandene zeitliche Aufwand wurde jedoch unterschätzt. Insbesondere die Verwaltung der Testdatenbank, der Unit-Tests und der Testfunktion in *SCADE* nahmen viel Zeit in Anspruch. Nichts desto trotz hat die Projektgruppe versucht dieses Thema so ausgiebig wie möglich mit den vorhandenen Ressourcen und der vorhandenen Zeit zu bearbeiten, da es sich bei der Entwicklung um ein sicherheitskritisches System handelte.

Bei einer selbstkritischen Betrachtung lässt sich festhalten, dass die Projektplanung an einigen Stellen hätte verbessert werden können. Jedoch wurden Fehler erkannt und diese stetig korrigiert. Die gesamte Projektgruppe war lernwillig und hat sich selbst kontinuierlich weiterentwickelt. Zudem soll berücksichtigt werden, dass trotz vieler Schwierigkeiten und Fehlentscheidungen das Projektziel größtenteils erreicht wurde. Insbesondere im Hinblick auf die geringe Anzahl an Projektteilnehmern waren die Fortschritte der Projektgruppe sehr gut.
Schlussendlich kann ein positives Resümee dahingehend gezogen werden, dass der Lerneffekt der Projektgruppe enorm hoch war. Die Projektgruppe hat einen tiefen Einblick in den realen Entwicklungsprozess von der Anforderungserhebung bis zum Abnahmetest des Produkts bekommen und ist dankbar für diese Erfahrung.

13.2.4 Personalmanagement

Zu Beginn des Moduls Projektgruppe bestand die PG *RCCARS* aus insgesamt sechs Leuten, der Mindestanzahl für eine Projektgruppe. Bereits nach einigen Wochen verließ ein Mitglied die Gruppe, sodass fünf Teilnehmer übrig blieben. Diese Teilnehmer waren entschlossen, die Projektgruppe dennoch fortzusetzen. Während der Einarbeitungs- und Anforderungsdefinitionsphase verließ ein weiteren Mitglied die Projektgruppe *RCCARS*. Die Gruppe bestand schließlich noch aus vier Teilnehmern. Ein großes Problem beim Verlassen des zweiten Mitglieds war, dass sich dieses bereits in ein Themengebiet eingearbeitet hatte und die Projektgruppe damit Wissen verlor. An dieser Stelle muss jedoch angemerkt werden, dass das Mitglied den Rest der Projektgruppe in ihrem eingearbeiteten Thema instruiert hat und stets für Fragen zur Verfügung stand.

Die geringe Anzahl an Teilnehmer bekam die Projektgruppe deutlich zu spüren. Neben dem erhöhten Implementierungsaufwand machten sich die mangelnden Ressourcen insbesondere bei „Nebentätigkeiten“ wie beispielsweise dem Verwalten der Testdatenbank und der kontinuierlichen Dokumentenpfelge bemerkbar.Dies hatte Folgen für das Erstellen der Zwischenberichte (siehe Bericht *Anforderungsdefinition vom 09. März 2016* und *Architektur und Schnittstellenspezifikation vom 07. Juni 2016*) und des Endberichts. Um die geringe Anzahl von Teilnehmern auszugleichen wurde das Prinzip der geteilten Ressourcen eingeführt. Dieses Prinzip verfolgt das Ziel eine Ressource gleichzeitig bei verschiedenen Sprints einsetzen zu können. So war es möglich, dass ein Mitglied, dass sich beispielsweise im Bereich der Netzwerkkommunikation auskennt, diese bei allen Subsystemen implementiert, ohne dass sich das eigentlich zuständige Mitglied dieses Subsystems/Sprints in dieses Thema einarbeiten muss. Dieses Prinzip hat sich bewährt und als sehr empfehlenswert herausgestellt.
Weitere Details zum Personalmanagement innerhalb der Sprints sind unter 13.2.2 zu finden.

13.2.5 Projektorganisation

Die Projektgruppe führte während der gesamten Projektlaufzeit wöchentliche Treffen aus. Während der Sprintphasen wurden diese Treffen als *weekly-scrVm-meetings* angesehen (mehr Informationen zum verwendeten Vorgehensmodell siehe 3.1). Jedes Treffen wurde von einem Projektgruppenmitglied organisiert und geleitet. Außerdem gab es einen Protokollanten. Die Zuteilungen rotierten dabei innerhalb der Projektgruppe. Nach jeder Projektgruppensitzung arbeitete der Protokollant ein Protokoll aus, welches in der Versionierungssoftware (siehe Kapitelabschnitt 3.3) für jeden über die gesamte Projektlaufzeit

einsehbar war. Des Weiteren erstellte der Protokollant einen Blog-Eintrag, welcher auf der Projektgruppenwebsite öffentlich zugänglich gemacht wurde. Innerhalb dieser wöchentlichen Treffen wurde der aktuelle Fortschritt der Projektgruppe besprochen. Zusätzlich wurden Probleme aufgezeigt und diese versucht in der Gruppe zu lösen. Darüber hinaus wurde eine Planung aufgestellt,in welcher die Ziele der kommenden Woche festgelegt wurden. Dabei wurden ebenfalls Aufgaben an die jeweiligen Gruppenmitglieder übertragen, die es umzusetzen galt.In der Woche darauf, wurden dann die zuvor gesetzten Ziele auf deren Fortschritt überprüft. So wurde kontinuierlich der Projektfortschritt überwacht und für jeden Teilnehmer einsehbar. Zusätzlich gab es innerhalb der Projektgruppe weitere wöchentliche Treffen zu wechselnden Zeitpunkten.

Wie zuvor im Kapitelabschnitt 3.8 angesprochen, wurde eine Website von einem Projektgruppenmitglied erstellt und verwaltet. Dort wurden Informationen zum Thema *RCCARS*, dem Projektgruppenteam sowie einigen Referenzprojekten veröffentlicht. Außerdem wurde aufgezeigt, an welchen Veranstaltungen die Projektgruppe teilgenommen hatte sowie wöchentliche Projektgruppenupdates über den Blog zur Verfügung gestellt.

Die Projektorganisation beziehungsweise die Projektplanung selbst wurde mit dem Softwaretool Microsoft Project 2016 angefertigt (siehe Kapitelabschnitt 3.3), welche dem Projektleiter bei der Planung unterstützen sollte. Dennoch war die Aufgabe der Projektplanung völlig neu und stellte sich als komplexer heraus als erwartet. Es war nicht immer einfach - auch wenn es sich um eine kleine Projektgruppe handelte -, sowie die vorhanden Aufgaben auf Gruppenmitglieder stets so zu verteilen, dass die Zeit und die vorhandenen Ressourcen optimal genutzt und gleichzeitig die Ziele erreicht werden. An dieser Stelle war es wichtig, frühzeitig die Kompetenzen der Mitglieder der Projektgruppe zu erkennen und die Teilnehmer dementsprechend einzuplanen. Des Weiteren stellte sich schnell heraus, dass der Verzug eines Arbeitspakets eines Projektgruppenmitglieds die gesamte Projektplanung durcheinanderbringen kann. Als Beispiel kann hier der sich ständig erhöhende Zeitaufwand in der Fertigstellung der Positionsbestimmung genannt werden. Durch die regelmäßige Erhöhung der benötigten Zeit musste die Projektplanung stets agil an die auftretenden Verhältnisse angepasst werden und Teilaufgaben auf andere Projektgruppenmitglieder übertragen werden. Auch hier bewährte sich das gewählte Vorgehensmodell (siehe Kapitelabschnitt 13.2.1).

13.2.6 ScrVm-Dokument

Das ScrVm-Planungsdokument ist ein eigenständiges, umfangreiches Dokument, dessen Nutzenfaktor nicht zu unterschätzt war und das in jedem Falle zur Betrachtung des Pro-

jektverlaufs herangezogen wurde. Der Arbeitsaufwand für dieses Dokument, insbesondere der Aufwand zur Pflege, war entsprechend hoch. Der Aufwand für die Verwaltung des Projekts rentierte sich jedoch.
Zusammenfassend lässt sich festhalten, dass das ScrVm-Planungsdokument trotz seines entsprechend hohen Verwaltungsaufwands einen hohen Nutzenfaktor beinhaltete und der Projektleitung sowie den Mitgliedern stets dabei half, den Projektfortschritt sowie die aktuell anliegenden Aufgaben einzusehen. Es war insbesondere bei der Verwaltung der Anforderungen von Vorteil. Durch die agilität des Vorgehensmodell *ScrVm* kam es häufig zu kurzfristigen Änderungen in der Anforderungsdefinition. Es wurden beispielsweise neue Anforderungen aufgenommen und alte Anforderungen zurückgeschoben. Durch das ScrVm-Planungsdokument konnte die Projektgruppe den Überblick über die Anforderungen behalten.

13.2.7 Gruppenklima und Kommunikation

Das Gruppenklima innerhalb der Projektgruppe war angenehm. Die Projektgruppenmitglieder kamen gut miteinander aus und arbeiteten vor Ort, sowie von zu Hause aus stets gut zusammen. Insbesondere während der Phasen vor den Reviews haben die Mitglieder gut zusammengearbeitet und auch Aktivitäten außerhalb der Projektgruppenzeit unternommen. Dazu zählen zum Beispiel Grill- oder Spieleabende. Im Weiteren hat die Projektgruppe geschlossen am Boule-Tunier 2016 der Universität Oldenburg teilgenommen und den zweiten Platz belegt. Dennoch kommt es bei jeder Gruppenarbeit mal zu internen Meinungsverschiedenheiten, welche sich jedoch alle auf angenehme Weise lösen ließen.
Wurden Fristen innerhalb der Projektgruppe durch Mitglieder versäumt, so wurden Sanktionen in Form von der Besorgung von Keksen auferlegt. Diese Sanktionen sorgten zu Beginn der Projektgruppe stets für einen gefüllten Lebensmittelvorrat im Projektgruppenraum und zog dennoch den gewünschten Lerneffekt nach sich.

Die Kommunikation in der Projektgruppe war größtenteils gut, wobei es an dieser Stelle zu Beginn häufig Verbesserungsbedarf gegeben hat. Es kam vor, dass in Sprints, in denen mehr als eine Person arbeitete, die interne Kommunikation nicht optimal war, was letztendlich kurzzeitig das Gruppenklima belastet hat. Die Mitglieder der Projektgruppe lernten jedoch schnell dazu und erlangten so eine deutlich verbesserte interne Kommunikation, was folglich das Gruppenklima verbesserte. Hierbei waren die wöchentlichen Projektgruppentreffen von Vorteil. Kommunikation, neben der persönlichen, verlief über zwei aktive E-Mail Verteiler. Bei dem einen Verteiler wurden die Projektgruppenbetreuer mit erreicht. Der zweite Verteiler diente der internen Kommunikation zwischen den Projektgruppenmitgliedern.

Im Weiteren liefen Kommunikationen zum Teil über Instant Messenger. Komplexe Sachverhalte wurden telefonisch geklärt. Auch die wöchentlichen Protokolle der Projektgruppensitzungen halfen bei der Verständigung zwischen den Teilnehmern.

Im Weiteren soll an dieser Stelle die herausragende Kommunikation zu den Betreuern angemerkt werden. Durch die örtliche Nähe (selbes Gebäude) zu dem Projektgruppenraum funktionierte die Kommunikation persönlich, was von großem Vorteil für die gesamte Projektgruppe war. Zudem waren die Betreuer immer ansprechbar und hilfsbereit, und das auch insbesondere außerhalb der eigentlichen Arbeitszeiten, was großes Lob verdient hat.

13.2.8 Gruppenräume

Der Projektgruppe stand während der gesamten Projektgruppenlaufzeit ein hervorragend ausgestatteter Projektgruppenraum zur Verfügung, welcher rund um die Uhr verfügbar war. Dieser Projektgruppenraum bot genug Platz für alle Mitglieder. Im Weiteren stellte der Raum ausreichend Rechner zur Verfügung, die softwaretechnisch vollständig für die Projektgruppe ausgestattet waren. Bei Fragen oder Problemen zu der Hard- und Software im Projektgruppenraum standen die Systemadministratoren sowie die Betreuer stets zur Verfügung.
Der Raum bot auch einen großen abschließbaren Schrank, in dem alle Unterlagen deponiert werden konnten, was der Projektgruppe die Arbeit erleichterte. Auch Werkzeug wie Lötkolben oder ähnliches standen in ausreichendem Maße zur Verfügung. Zudem musste durch einen eigenen Raum für die Projektgruppe in der Regel keine Rücksicht auf weitere Besucher des Raums genommen werden, wodurch sich die Projektgruppe voll entfalten konnte und die Flexibilität gesteigert wurde.

13.2.9 Erfahrungen

Die gesammelten Erfahrungen innerhalb der Projektgruppe waren bemerkenswert. Die Projektgruppenmitglieder sind sich einig, einen deutlich höheren Lerneffekt im Modul Projektgruppe erlangt zu haben, als je im Studium zuvor. Insbesondere die Nähe zur realen Industrie wurde sehr gut widergespiegelt und verleihte einen tiefen Einblick in reale Entwicklungsprozesse. Es wurde deutlich, wie komplex der Entwurf eines Systems ist und mit was sich alles neben der wesentlichen Implementierung befasst werden muss. Insbesondere die Wichtigkeit der Anforderungserhebenung, des Entwurfs der Systemarchitektur und der Definition der Schnittstellen wurde der PG bewusst.
Allerdings muss angemerkt werden, dass der zeitliche Aufwand der Projektgruppe sehr

hoch war. Insbesondere die Reviews sowie die Anfertigung der jeweiligen Berichte forderten einen hohe zeitliche Investition. Dies wurde durch die geringe Anzahl an Personen innerhalb der Projektgruppe noch einmal verstärkt.

Trotz des hohen zeitlichen Aufwands hat die Projektgruppe *RCCARS* den Teilnehmern sehr viel Spaß bereitet. Die Projektgruppenmitglieder würden im Nachhinein die Projektgruppe erneut belegen und die Abteilung empfehlen. Insbesondere der Entwicklungsfortschritt sowie eine erfolgreiche Verteidigung des Gesamtsystems und die positive Verstärkung von Personen außerhalb der Projektgruppe sorgten für viel Motivation. Die gesammelten Erfahrungen werden den Teilnehmer im zukünftigen Berufsleben hilfreich sein.

13.2.10 Danksagung

An dieser Stelle bedankt sich die Projektgruppe *RCCARS* für die exzellente Betreuung und der reibungslosen Kooperation mit dem OFFIS. Gesonderter Dank geht dabei an Günter Ehmen für die tatkräftige Unterstützung während der gesamten Projektlaufzeit und das auch außerhalb der umgänglichen Arbeitszeiten. Im Weiteren gilt es sich bei Stefan Puch zu bedanken, der stets Hilfestellungen bei der Programmiersprache C gegeben und ausgiebig Korrektur gelesen hat. Zusätzlichen bedankt sich die Projektgruppe bei Prof. Martin Fränzle für den hilfreichen Vortrag im Bereich der Regelungstechnik. Ohne diesen Vortrag hätte die Umsetzung der Regelung deutlich mehr Zeit in Anspruch genommen. Im Weiteren soll die Aufmerksamkeit auch auf Christian Schrader gerichtet werden, der kontinuierlich Unterstützung bei dem Umgang mit dem Softwaretool *SCADE* geboten hat.

Zuletzt bedankt sich die gesamte Projektgruppe *RCCARS* für das tolle Projekt und den reibungslosen Ablauf bei der Carl von Ossietzky Universität Oldenburg sowie bei den Abteilungen hybride Systeme und sicherheitskritische eingebettete Systeme.

13.3 Ausblick

In diesem Kapitelabschnitt wird ein Ausblick über mögliche Optimierungen des Systems gegeben, welche voraussichtlich dazu führen würden, dass das Projektziel in seinem vollen Maße und darüber hinaus hätte erreicht werden können. Die jeweiligen Verbesserungen konnten aus zeitlichen Gründen nicht mehr umgesetzt werden.

13.3.1 Positionsbestimmung

Obwohl die Positionsbestimmung sämtliche Anforderungen, die an diese gestellt wurden, erfüllt, kann die Software noch an einigen Stellen erweitert werden.
Zum Einen ist die Objekterkennung noch nicht vollständig, da bislang nur ein einziges Fahrzeug detektiert werden kann. Der Algorithmus müsste für mehrere Fahrzeuge erweitert werden. Ebenfalls sind Probleme mit der Fahrzeugmarkererkennung aufgetreten, sofern sich diese relativ nah an einander befinden. Dies müsste näher untersucht werden. An dieser Stelle könnte auch eine bessere Konstellation der Lampen evaluiert werden. Es gibt vereinzelte Stellen, die weniger gut ausgeleuchtet sind und somit die Objekterkennung dort behindern.
Die Kamerakalibrierung kann ebenfalls überarbeitet werden. Momentan muss dem Programm jedes Bild mit dem Schachbrettmuster einzeln übergeben werden. Es wäre komfortabler, wenn die Muster direkt aus dem Videostream erkannt werden könnten. Die im Anschluss bestimmten Parameter werden in diesem Stand per Hand von der erstellten output.xml in die Konfigdatei übertragen. Eine automatische Übertragung wäre äußerst praktisch. Diesbezüglich wäre eine Anpassung des FileReaders sinnvoll. Ändert der Admin beispielsweise die Filterparameter, so werden diese nicht in die Konfigurationsdatei übernommen und müssen per Hand nachgetragen werden.
Zudem kann die Berechnung der Transformationsmatrix optimiert werden. Die Verschiebung der Rennstreckenmarker ist noch nicht präzise genug.
Außerdem wäre eine dynamische Filteranpassung, sowie eine automatische Rennstreckenerkennung, inklusive der Erstellung einer csv-Datei die die Rennstrecke repräsentiert, wünschenswert. Außerdem wäre die Implementierung eines Kalman Filters, der das Verhalten des Fahrzeugs voraussagen kann, empfehlenswert.

13.3.2 Wrapper Code

In diesem Kapitelabschnitt werden mögliche Erweiterungen, die den Wrapper Code betreffen, beschrieben.

13.3.2.1 Kommunikation über Bluetooth

In zukünftigen Ausbaustufen (siehe Kapitelabschnitt 2.2) wird die Fahrzeugplatine gegen eine eigens entwickelte Platine ausgetauscht. Um die Übertragung von Regelungswerten weiterhin realisieren zu können, muss die Kommunikation über Bluetooth realisiert werden. Zusätzlich sollte ein Konzept entworfen und umgesetzt werden um wahlweise die CarControl oder die Bluetooth-Kommunikation nutzen zu können.

13.3.2.2 Systemzustände mit SCADE modellieren

Aktuell wurde das Management der Systemzustände der Control-Unit händisch im Wrapper Code implementiert (siehe Kapitelabschnitt 10.3.1.9). Um dies robuster und erweiterbarer zu gestalten und Fehler zu vermeiden, könnten die Systemzustände und dessen Transitionen in *SCADE* modelliert werden. Dies ist im Hinblick auf den sicheren Fehlerzustand zu empfehlen.

13.3.2.3 Prüfroutinen des Watchdogs in SCADE modellieren

Aktuell wurden sämtliche Prüfroutinen des Watchdogs händisch innerhalb der Control-Unit implementiert (siehe Kapitelabschnitt 10.3.1.7). Um an dieser Stelle Fehler in den Prüfroutinen und ggf. einen falschen ”Alarm” zu vermeiden, könnten diese Prüfroutinen in *SCADE* modelliert werden.

13.3.2.4 Dateiname der Konfigurationsdatei

Im aktuellen Zustand wurde der Dateiname der Konfigurationsdatei fest implementiert, sodass die Konfigurationsdatei immer den gleichen Dateinamen tragen muss (siehe Kapitelabschnitt 10.3.1.1). Um für das System unterschiedliche Konfigurationsdateien anlegen zu können, könnte der Dateiname als Parameter beim Ausführen der Control-Unit übergeben werden können.

13.3.3 SCADE Modell

In diesem Kapitelabschnitt werden mögliche Erweiterungen, die das *SCADE* Modell betreffen, beschrieben.

13.3.3.1 Längsregelung

Bei der Längsregelung ist es wichtig zukünftig das zugehörige Kennfeld weiter zu evaluieren. Das Kennfeld ist zur Zeit nur teilweise gefüllt. Eine weitere Füllung sowie bessere Evaluation der Stellwerte könnte möglicherweise die Performance des Systems weiter steigern. Man könnte zusätzlich versuchen die Regelgüte durch weitere Einstellungen der P, I und D-Anteile des Reglers zu optimieren.

13.3.3.2 Dynamische Bremszonen

Im Weiteren sollten die aktuell vorhandenen statischen Bremszonen (siehe Kapitelabschnitt 10.3.3.8) dynamisch angepasst werden. An dieser Stelle könnte beispielsweise durch den zuvor angesprochenen Geschwindigkeitsregler (siehe Kapitelabschnitt 13.3.3.1)

innerhalb der Bremszonen kein Bremssignal vorgegeben werden, sondern eine Zielgeschwindigkeit zu der das Fahrzeug runter geregelt wird. Das hätte den Vorteil, dass wenn das Fahrzeug mit einer ausreichend niedrigen Geschwindigkeit in eine Kurve fährt, es nicht zusätzlich vorher noch bremst. Durch die dynamischen Bremszonen würde sich die Performance der Regelung voraussichtlich steigern lassen.

13.3.3.3 Veränderung der Trajektorie

Ein Weiterer wichtiger Punkt ist die Verlegung der Trajektorie. Diese sollte zukünftig nicht die Mittellinie repräsentieren, sondern die Ideallinie, wie sie üblicherweise im Motorsport gefahren wird. Dadurch könnten höhere Kurvengeschwindigkeiten gefahren werden. Hierbei ist jedoch anzumerken, dass die Regelung dabei eine hohe Güte aufweisen muss, da die Ideallinie dicht an den Fahrbahnbegrenzungen vorbei führen würde. Die Gefahr einer Bandenkollision wäre damit deutlich gesteigert.
Die Trajektorie sollte durch die hohe Kollisionsgefahr mit den Fahrbahnbegrenzungen deswegen erst verlegt werden, wenn ein neues Verfahren zur Bandenkollisionserkennung implementiert wurde (siehe Kapitelabschnitt 13.3.3.5). Bei dem aktuellen Verfahren zur Kollisionserkennung wird lediglich die reale Begrenzung virtuell verbreitert. Dies ist ein übliches Verfahren in der Robotik.

13.3.3.4 Model Predictive Control

Einer der wohl wichtigsten Optimierungsansätze wäre die Implementierung einer modellprädiktiven Regelung (Model Predictive Control, kurz MPC). Wie bereits im Kapitelabschnitt 11 beschrieben, ist das gesamte System mit Latenzen belastet. Diese Latenzen sowie die allgemeine Massenträgheit des Fahrzeugs, auch wenn diese gering ist, führt dazu, dass auf Posen verzögert reagiert wird und Lenkbefehle verzögert umgesetzt werden. Diesem Problem könnte mit einer Model Predictive Control entgegen gewirkt werden.
Da das in Kapitel 10.3.3.14 beschriebene Modell nur das Verhalten eines langsamen Fahrzeugs zuverlässig darstellen kann, sollte für genauere Simulationen ein anderes Fahrzeugmodell verwendet werden. Alternativen wären das Bicyle- bzw. SingleTrackModell.

13.3.3.5 Bandenkollisionserkennung

Im aktuellen Entwicklungsstand werden für die Bandenkollisionserkennung lediglich die Koordinaten der Fahrzeugmitte verwendet. In Verbindung dazu wurde, wie in Kapitelabschnitt 10.3.3.3 beschrieben, die reale Bande virtuell verbreitert. Da die Fahrzeugmitte selbst jedoch nie mit der Bande in Berührung kommen kann, wurden die Rennstreckenbanden künstlich verbreitert für das *SCADE* Modell. Zum Anderen ist es nicht möglich

die Kamera auf den Millimeter genau zu kalibrieren. Je nach Position auf der Rennstrecke sind Abweichungen von 2 bis 10 mm möglich. Die Kollisionserkennung sieht diese Abweichungen jedoch nicht vor, sodass gelegentlich fälschlicherweise Kollisionen erkannt werden.
Eine Verbesserung kann durch die Verwendung der Fahrzeugkanten statt des Fahrzeugmittelpunktes erzielt werden. Da die Mitte und der Ausrichtungswinkel des Fahrzeugs ϕ bekannt sind, können die Fahrzeugkoordinaten als Stützvektor a verwendet werden. Der Richtungsvektor kann durch $b = (\cos(\phi)\sin(\phi))^T$ beschrieben werden. Normiert man den Richtungsvektor auf die halbe Fahrzeuglänge, so zeigt der Vektor der $f = a + b$ auf die Front des Fahrzeugs. Analog dazu der Vektor h=a-b auf das Heck. Wählt man nun die Position der Fahrzeugfront wiederum als Stützvektor, so kann die Position des vorderen rechten Kotflügels mit Hilfe des zu b orthogonalen Vektors $b_2 = (\sin(\phi) - \cos(\phi))^T$, welcher auf die Länge der halben Fahrzeugbreite normiert wurde, berechnet werden. Es gilt dann $f_r = f + b_2$ bzw. für den linken Kotflügel $f_l = f - b_2$. Die Bestimmung der Position der hinteren Kotflügel ist komplett analog dazu.
Innerhalb des *SCADE* Modells kann somit geprüft werden, ob ein Fahrzeugkotflügel die Banden berührt.

13.3.4 CarControl

Im aktuellen Entwicklungszustand befindet sich die CarControl in einem funktionsfähigen Zustand. Sämtliche Funktionen, welche benötigt werden, wurden umgesetzt. Eine Verbesserung, welche sich bereits in Planung befindet, wäre der Austausch der Fahrzeugplatine.

13.3.4.1 Austausch der Fahrzeughardware

In zukünftigen Ausbaustufen (siehe Kapitelabschnitt 2.2) wird die Hardware bzw. die Platine des Fahrzeuges gegen eine eigens entwickelte Platine ausgetauscht. Hierdurch ermöglicht sich die Kommunikation zwischen der Control-Unit und dem Fahrzeug einzig über einen Bluetooth-Stick, welcher am Rechner angeschlossen wird. Somit wird ebenfalls die bidirektionale Kommunikation direkt mit dem Fahrzeug ermöglicht und weitere Kontrolle über das Fahrzeug gewonnen. Die CarControl im jetzigen Zustand wird somit vollständig ersetzt und ist zu dem Fahrzeug mit der ausgetauschten Hardware nicht mehr kompatibel.

13.3.5 Kommunikation

Die Realisierung der Kommunikation wurde in Kapitelabschnitt 10.2 beschrieben. Für zukünftige Projektgruppen könnte der Inhalt des Netzwerkdatenpakets angepasst werden und der Kommunikationskanal über Multicast statt Broadcast realisiert werden.

13.3.5.1 Inhalt des Netzwerkdatenpakets

Aktuell werden sämtliche Variablen aller Nachrichtentypen über das Netzwerk übertragen (siehe Kapitelabschnitt 10.2.1), sodass ein Netzwerkdatenpaket in etwa 500 Bytes groß ist. Wenn drei Komponenten miteinander kommunizieren, so mag dies eine geringe Menge an Daten sein. Im Anbetracht auf die Tatsache, dass in weiteren Ausbaustufen dieses Projektes (siehe Kapitelabschnitt 2.2) mehrere Fahrzeuge die Rennstrecke befahren sollen, vergrößert sich das Datenaufkommen entsprechend.

Eine mögliche Verbesserung wäre die Übertragung nur der Variablen bzw. dessen Informationen, welche zum gesendeten Nachrichtentypen gehören. Somit könnte die Größe des Netzwerkpaketes um einige Bytes reduziert werden und somit mehr Bandbreite für weitere verschickte Daten zur Verfügung stehen.

13.3.5.2 Nutzung von Multicast

Wie in Kapitelabschnitt 4.5 und 10.2 beschrieben, ist die Kommunikation zwischen den Systemkomponenten über UDP-Sockets realisiert. Hierzu wird Broadcast als Kommunikationsweg genutzt, sodass sämtliche Systemkomponenten die Netzwerkdatenpakete erhalten. So muss nicht jeder Systemkomponente zu Beginn des Betriebs alle IP-Adressen der übrigen Systemkomponenten bekannt sein. Dies wäre bei maximal 18 Systemkomponenten (Systemsteuerung, Positionsbestimmung und 16 Mal die Control-Unit) ein erhöhter Aufwand. Sollte eine der Systemkomponenten versehentlich mit einem öffentlichen Netzwerk verbunden sein, wird dies gegebenenfalls mit Netzwerkdatenpaketen geflutet und überlastet.

Eine mögliche Verbesserung, um das versehentliche Fluten eines öffentlichen Netzwerkes zu verhindern, wäre das Nutzen von Multicast. Bei dem Kommunikationsweg über Broadcast werden Netzwerkdatenpakete ausnahmslos an alle Netzwerkteilnehmer verschickt. Beim Multicast werden diese nur an eine Gruppe (Multicast-Domain) versendet, welche sich aus interessierten Empfängern zusammensetzt. Sämtliche Empfänger müssen sich über eine Multicast-Adresse beim Sender anmelden und werden der Gruppe hinzugefügt. Dem Sender ist dann bekannt, an welche(n) Empfänger die Netzwerkdatenpakete gesen-

det werden sollen. Übrige Netzwerkteilnehmer, welche sich nicht beim Sender anmelden, erhalten die Netzwerkdatenpakete nicht. So kann ein Fluten des Netzwerks verhindert werden.

13.3.6 Systemsteuerungssoftware

Eine bereits vorgesehene Erweiterung der Systemsteuerungssoftware ist die Visualisierung der Fahrzeugposen, welche von der Positionsbestimmung geliefert werden. So wäre es einfacher möglich die Funktionsweise der Positionsbestimmung mit Hilfe der Systemsteuerungssoftware zu kontrollieren. Auf der Benutzeroberfläche (siehe Kapitelabschnitt 10.4.1.1) wurde bereits die Ebene ”Car Locations” angelegt, welche für diese Zwecke vorgesehen ist. Die Realisierung ist beispielsweise mit Hilfe der Java-Bibliothek *Java 2D* [Oraa] möglich, welche die erforderlichen Funktionen bietet, um Fahrzeuge und die Rennstrecke zu skizzieren.
Um die Erweiterbarkeit der Software weiter zu verbessern, könnte die Generierung der Optionsdialoge der Settings-Ebene automatisiert werden, so dass die Ergänzung neuer Konfigurationsparameter vereinfacht wird. Ein möglicher Ansatz wäre das Auslesen einer XML-Datei, in welche alle Parameter mit ihren Namen, Beschreibungen, Wertebereichen und Defaultwerten eingetragen werden, im Zusammenspiel mit der Implementierung eines entsprechenden Parsers.

14 Anhang

14.1 Fehlercodetabelle

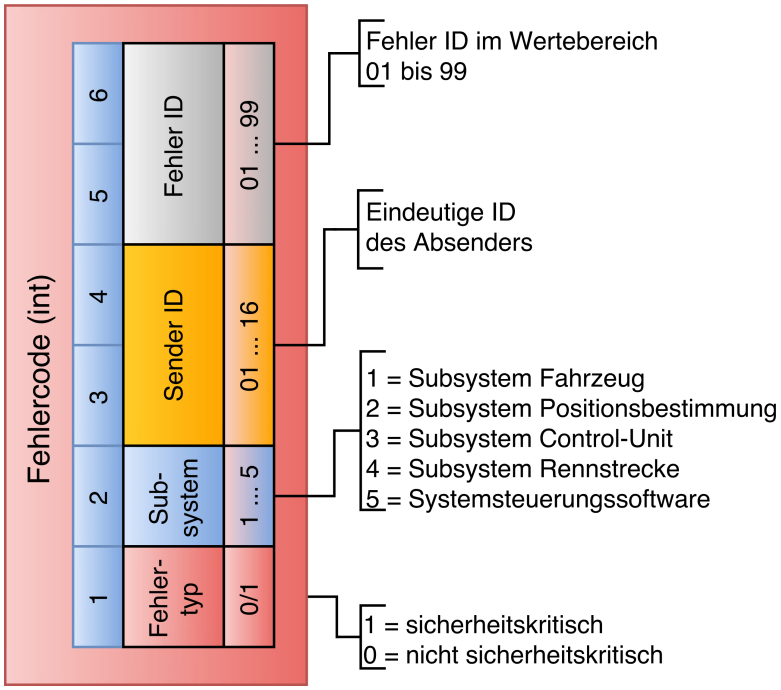


Abbildung 14.1: Aufbau des Fehlercodes.

kritisch	Subsys.	Sender-ID	Fehler-ID	
0	2	00	0	Kamerainitialisierung fehlgeschlagen
0	2	00	1	Keine Balken oder Marker gefunden
0	2	00	2	Keine oder zu wenig Rennstreckenmarker gefunden
0	2	00	3	Zu viele Rennstreckenmarker gefunden
0	2	00	4	Anzahl der initialisierten Fahrzeuge stimmt nicht mit der Anzahl der gefundenen Balken überein
0	2	00	5	Realzeitanforderung konnte nicht eingehalten werden
1	2	00	6	Rennstrecke oder Gerüst wurden bewegt

kritisch	Subsys.	Sender-ID	Fehler-ID	
0	2	00	7	Kamera liefert weniger als 100 fps
1	2	00	8	Positionsbestimmung wurde beendet
1	2	00	9	Fahrzeugpose seit mehreren Durchläufen nicht feststellbar.
0	2	00	10	Fahrzeug nicht gefunden
1	2	00	11	Es konnten keine Konturen im Bild erkannt werden. Dies deutet auf den Ausfall der Beleuchtungseinheit hin.
1	2	00	12	Verbindung zur Kamera verloren
1	3	01…16	01	Kein Fahrzeug detektiert
1	3	01…16	02	Indizierungsfehler - Out of Bounds (SCADE)
1	3	01…16	03	Sonstiger Fehler (SCADE)
1	3	01…16	04	Fahrzeug auf nicht befahrbaren Bereich
1	3	01…16	05	Fehler bei dem Feststellen in welchem Abteil sich das Fahrzeug auf der Rennstrecke befindet.
1	3	01…16	06	Fahrzeug fährt nicht gegen den Uhrzeigersinn.
1	3	01…16	07	KEINER
1	3	01…16	10	Fehler beim Indizieren des Throttle Kennfelds (SpeedMapping (operator)).
1	3	01…16	20	Fehler beim Finden der Trajektorie. (Indizierungsfehler)
1	3	01…16	21	Fehler beim Finden der Trajektorie. (Keine Zieltrajektorie gefunden. Es standen keine im Array, welches vom Wrapper Code kommt)
1	3	01…16	22	Fehler beim Finden der Trajektorie. (Sonstiger Fehler)
1	3	01…16	23	Fehler beim Finden der Trajektorie. (Gefundene Trajektorie(n) passen nicht zum aktuellen Streckenabschnitt)
1	3	01…16	24	KEINER
1	3	01…16	25	Fehler beim Mappen des lokalen Winkelfehlers zum globalen Winkelfehler (CalculateAngle)
1	3	01…16	26	Kein Wert im Kennfeld für entsprechende Kurve vom Steering Controller gefunden (IndexingCharacteristivDiagram)

kritisch	Subsys.	Sender-ID	Fehler-ID	
1	3	01... 16	27	Sonstiger Fehler beim ziehen eines Kennfeldwerts vom Steering Controller (IndexingCharacteristivDiagram)
1	3	01... 16	51	Sicherheitskritischen Fehler von Subsystem Fahrzeug empfangen
1	3	01... 16	52	Sicherheitskritischen Fehler von Subsystem Systemsteuerungssoftware empfangen
1	3	01... 16	53	Sicherheitskritischen Fehler von Subsystem Control-Unit empfangen.
1	3	01... 16	54	Sicherheitskritischen Fehler von Subsystem Rennstrecke empfangen.
1	3	01... 16	55	Sicherheitskritischen Fehler von der Systemsteuerungssoftware empfangen.
1	3	01... 16	56	Not-Aus von der Systemsteuerungssoftware empfangen.
1	3	01... 16	57	Fehler bei der Initialisierung der Systemzustände.
1	3	01... 16	58	Rennstreckendatei nicht gefunden.
1	3	01... 16	59	Konfigurationsdatei nicht gefunden!.
1	3	01... 16	60	Der Socket für den UDP-Receiver konnte nicht geöffnet werden.
1	3	01... 16	61	Der Port für den UDP-Receiver konnte nicht an den Socket gebunden werden.
1	3	01... 16	62	Der Socket für den UDP-Sender konnte nicht geöffnet werden.
1	3	01... 16	63	Fehler beim Bauen des Netzwerkpaketes. Es wurde vor dem Bauen nicht zurückgesetzt.
1	3	01... 16	64	Fehler beim Senden von Netzwerkpaketen.
1	3	01... 16	65	Schnittstelle zur CarControl konnte nicht konfiguriert werden.
1	3	01... 16	66	Schnittstelle zur CarControl konnte nicht geöffnet werden.
1	3	01... 16	67	Verbindung zur CarControl liefert falsches Feedback.
1	3	01... 16	68	Verbindung zur CarControl konnte nicht hergestellt werden.
1	3	01... 16	69	Fehler beim Senden des Regelungswertes.

kritisch	Subsys.	Sender-ID	Fehler-ID	
1	3	01. . . 16	70	Fehler beim Wechsel in den Systemzustand ”Inspektion”.
1	3	01. . . 16	71	Fehler beim Wechsel in den Systemzustand ”Initialisierung”.
1	3	01. . . 16	72	Fehler beim Wechsel in den Systemzustand ”Standby”.
1	3	01. . . 16	73	Fehler beim Wechsel in den Systemzustand ”Rennbetrieb”.
1	3	01. . . 16	74	Fehler beim Wechsel in den Systemzustand ”Fehler”.
1	3	01. . . 16	75	Fehler beim Wechsel in den Systemzustand ”Aus”.
1	3	01. . . 16	76	Die letzten empfangenen Fahrzeugdaten sind zu alt. Erlaubtes Alter sind wurde überschritten.
1	3	01. . . 16	77	Plausibilitätsfehler der Fahrzeugdaten. Die Fahrzeugdaten wurden nicht von der Positionsbestimmung versendet.
1	3	01. . . 16	78	Plausibilitätsfehler der Fahrzeugdaten. Die Empfangszeit der Fahrzeugdaten liegt vor dem Alter der Fahrzeugdaten.
1	3	01. . . 16	79	Plausibilitätsfehler der Fahrzeugdaten. Die aktuell empfangenen Fahrzeugdaten sind älter als die letzten Fahrzeugdaten.
1	3	01. . . 16	80	Plausibilitätsfehler der Fahrzeugdaten. Die Fahrzeugdaten enthielten zu oft keine Fahrzeugpose.
1	3	01. . . 16	81	Plausibilitätsfehler der Fahrzeugdaten. Die Änderung der Koordinaten des Fahrzeuges auf der X-Achse war größer als erlaubt.
1	3	01. . . 16	82	Plausibilitätsfehler der Fahrzeugdaten. Die Änderung der Koordinaten des Fahrzeuges auf der Y-Achse war größer als erlaubt.
1	3	01. . . 16	83	Plausibilitätsfehler der Fahrzeugdaten. Die Änderung des Ausrichtungswinkels des Fahrzeuges war größer als erlaubt.
1	3	01. . . 16	84	Plausibilitätsfehler der Fahrzeugdaten. Das Fahrzeug bewegt sich unerlaubt.

kritisch	Subsys.	Sender-ID	Fehler-ID	
1	3	01… 16	85	Plausibilitätsfehler der Fahrzeugdaten. Das Fahrzeug bewegt sich im Rennbetrieb nicht.
1	3	01… 16	86	Die letzten Kontrolldaten sind älter als das erlaubte Alter.
1	3	01… 16	87	Plausibilitätsfehler der Kontrolldaten. Die Kontrolldaten wurden nicht von der Systemsteuerungssoftware versendet.
1	3	01… 16	88	Plausibilitätsfehler der Kontrolldaten. Der Empfangszeitpunkt der Kontrolldaten liegt vor dem Alter der Kontrolldaten.
1	3	01… 16	89	Plausibilitätsfehler der Kontrolldaten. Die aktuell empfangenen Kontrolldaten sind älter als die letzten Kontrolldaten.
1	3	01… 16	90	Plausibilitätsfehler der Kontrolldaten. Die Systemsteuerungssoftware hat einen unerlaubten Systemzustandswechsel angefordert.
1	3	01… 16	91	Plausibilitätsfehler der Fehlerdaten. Empfangenes Not-Aus wurde nicht von der Systemsteuerungssoftware versendet.
1	3	01… 16	92	Plausibilitätsfehler der Ausgaben des SCADE-Modells. Der Regelungswert für die Querführung ist nicht im Rahmen zwischen 0 und 4095.
1	3	01… 16	93	Plausibilitätsfehler der Ausgaben des SCADE-Modells. Der Regelungswert für die Längsführung ist nicht im Rahmen zwischen 0 und 4095.
1	3	01… 16	94	Es konnte nicht festgestellt werden, welche Trajektorienpunkte am nähesten am Fahrzeug liegen..
0	5	00	01… 16	Fehlerdatenpaket mit unkritischem Fehler von einer Control-Unit mit der entspr. ID (1… 16) empfangen.
1	5	00	01… 16	Fehlerdatenpaket mit kritischem Fehler von einer Control-Unit mit der entspr. ID (1… 16) empfangen.

kritisch	Subsys.	Sender-ID	Fehler-ID	
0	5	00	20	Fehlerdatenpaket mit unkritischem Fehler von der Positionsbestimmung empfangen
1	5	00	20	Fehlerdatenpaket mit kritischem Fehler von der Positionsbestimmung empfangen
1	5	00	21	Fehlerdatenpaket von der Systemsteuerungssoftware selbst empfangen
1	5	00	22	Fehlerdatenpaket von unbekanntem Absender empfangen
1	5	00	30	unbekanntes Netzwerkdatenpaket empfangen
1	5	00	40	Gesamtrealzeitbedingung verletzt (Bildaufnahme bis Ausgabe der Steuerungsdaten)
1	5	00	41	interne Realzeitbedingung der Systemsteuerungssoftware verletzt
1	5	00	50	manueller Notstopp ausgelöst
1	5	00	51	unbekannte Benutzereingabe
1	5	00	52	unerlaubte Benutzereingabe während der Inspektion
1	5	00	53	unerlaubte Benutzereingabe während der Initialisierung
1	5	00	54	unerlaubte Benutzereingabe im Standby-Zustand
1	5	00	55	unerlaubte Benutzereingabe im Rennbetrieb
1	5	00	60	Zeitdatenpaket während der Initialisierung empfangen
1	5	00	61	während der Initialisierung Kontrolldaten von mehr Control-Units als erwartet empfangen
1	5	00	62	während der Initialisierung Kontrolldaten von unbekannter Control-Unit empfangen
1	5	00	65	Zeitdatenpaket im Standby-Zustand empfangen
1	5	00	66	mindestens ein Fahrzeug hat sich im Standby-Zustand bewegt
1	5	00	67	im Standby-Zustand wurden Kontrolldaten von mindestens einer unerlaubten Control-Unit empfangen
1	5	00	68	Liste der von der Positionsbestimmungssoftware erkannten Fahrzeuge hat sich im Standby-Zustand verändert

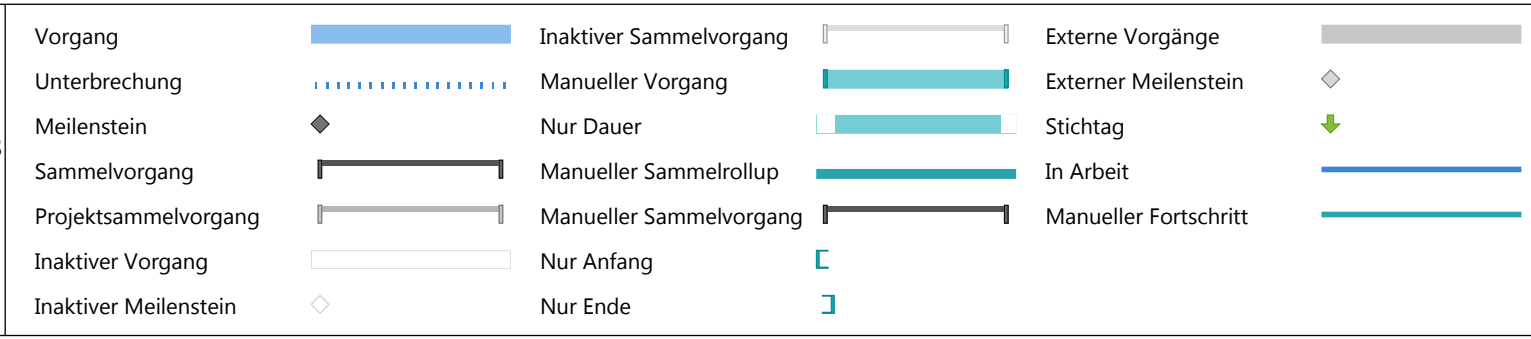
kritisch	Subsys.	Sender-ID	Fehler-ID	
1	5	00	69	mindestens ein Fahrzeug hat sich im Standby-Zustand gedreht
1	5	00	70	im Rennbetrieb wurden Kontrolldaten von mindestens einer unerlaubten Control-Unit empfangen
1	5	00	71	Liste der von der Positionsbestimmungssoftware erkannten Fahrzeuge hat sich im Rennbetrieb verändert
1	5	00	72	mindestens ein Fahrzeug mit unerlaubter oder unplausibler Position oder Bewegung während des Rennbetriebs
1	5	00	90	unvorgesehener SystemControl-Thread-Interrupt
1	5	00	99	unerlaubter Systemzustand

Tabelle 14.1: Fehlercodetabelle.

14.2 Projektplan-Export

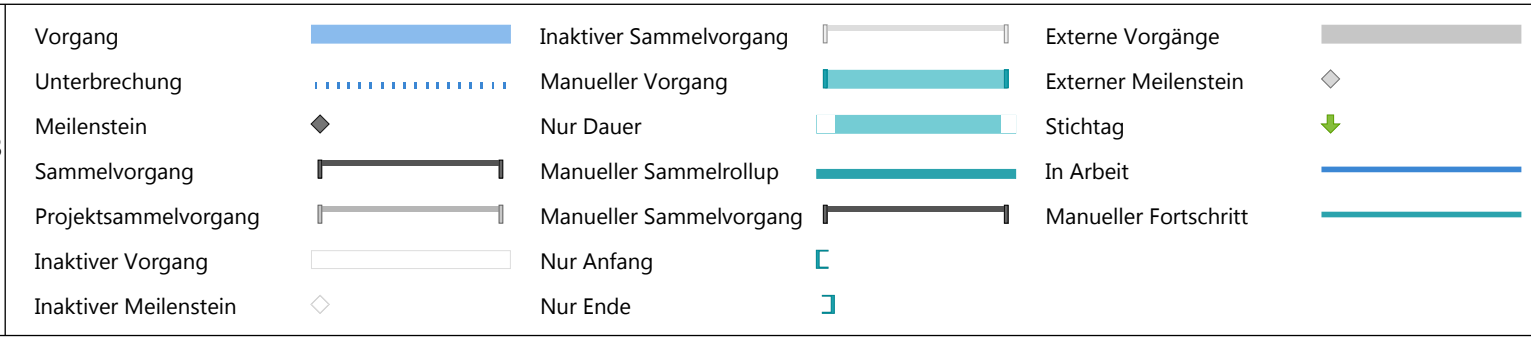
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
1	Urlaub	*Urlaub gesamte PG*	*14 Tage*	*Son 20.12.15*	*Son 03.01.16*	
2	*Prüfungsphase*	*PG Mitglieder nur eingeschränkt für die PG tätig*	*40 Tage*	*Mit 20.01.16*	*Mon 29.02.16*	
3	*Urlaub*	*Urlaub Michael*	*4 Tage*	*Mon 28.03.16*	*Fre 01.04.16*	
4	*Urlaub*	*Urlaub Anatolij*	*6 Tage*	*Mon 28.03.16*	*Son 03.04.16*	
5	*Urlaub*	*Urlaub Michael*	*11 Tage*	*Fre 15.07.16*	*Die 26.07.16*	
6						
7	Öffentlichkeitsarbeit	Profildatein für die Website ausfüllen	3 Tage	Mit 18.11.15	Sam 21.11.15	
8	Öffentlichkeitsarbeit	Erste Version der Website erstellen	4 Tage	Sam 21.11.15	Mit 25.11.15	7
9	Öffentlichkeitsarbeit	Website online stellen	0 Tage	Mit 25.11.15	Mit 25.11.15	8
10	Projektmanagement	Festlegung des Projektmanagementsystems	14 Tage	Mit 11.11.15	Mit 25.11.15	
11	Bericht	Definition System (Subsysteme, Komponenten)	7 Tage	Mit 25.11.15	Mit 02.12.15	
12	State of the art	Ausarbeitung Paper von Vorgängerprojekten	20 Tage	Mit 28.10.15	Die 17.11.15	
13	Hardware	Kamera- (inkl. Zubehör) & IR Lampenangebot (inkl Zubehör) vorlegen	0 Tage	Mit 09.12.15	Mit 09.12.15	12
14	Hardware	Kamera- (inkl. Zubehör) & IR Lampenangebot (inkl Zubehör) bestellen	42 Tage	Mit 09.12.15	Mit 20.01.16	13
15	Hardware	Gestell für Kamera+IR-Lampe aufbauen	42 Tage	Mit 20.01.16	Mit 02.03.16	14
16	Bericht	Projektziel definieren (Erstes Szenario)	0 Tage	Mit 18.11.15	Mit 18.11.15	12
17	Hardware	Computerangebot vorlegen	0 Tage	Mit 18.11.15	Mit 18.11.15	12
18	Hardware	Computer bestellen	14 Tage	Mit 18.11.15	Mit 02.12.15	17
19	Hardware	Computer installieren	28 Tage	Mit 09.12.15	Mit 06.01.16	18
20	Vortrag	Realzeitbetriebssysteme Vortrag	0 Tage	Mit 16.12.15	Mit 16.12.15	

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



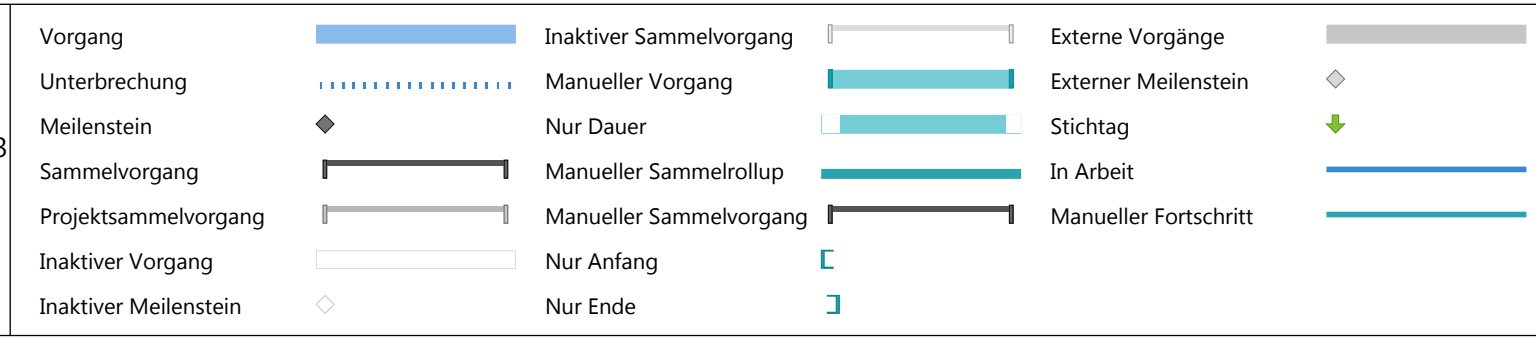
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
21	Vortrag	SCADE & PID Regler Vortrag	0 Tage	Mit 06.01.16	Mit 06.01.16	
22	Öffentlichkeitsarbeit	Erstellung Poster	7 Tage	Mit 13.01.16	Mit 20.01.16	
23	Öffentlichkeitsarbeit	Schülerinformationstag	0 Tage	Mit 20.01.16	Mit 20.01.16	22
24	Hardware	STM-DAC+RS232/USB Steuersw	26 Tage	Mit 27.01.16	Mon 22.02.16	
25	Hardware	Fernbedienung und STM Board verbinden	11 Tage	Die 23.02.16	Sam 05.03.16	24
26	Hardware	PC-SW Tastenauswertung mit STM	11 Tage	Die 23.02.16	Sam 05.03.16	24
27	Hardware	Ersten Prototypen fertigstellen	0 Tage	Mon 07.03.16	Mon 07.03.16	25;26
28						
29	Arbeitspaket 1	Einleitung erstellen	6 Tage	Mit 27.01.16	Die 02.02.16	
30	Arbeitspaket 1	State of the Art Teil erstellen	6 Tage	Mit 27.01.16	Die 02.02.16	
31	Arbeitspaket 1	LTV erstellen	6 Tage	Mit 27.01.16	Die 02.02.16	
32	Arbeitspaket 1	Aufgabenstellung schreiben	6 Tage	Mit 27.01.16	Die 02.02.16	
33	Arbeitspaket 1	Szenario fertigstellen	6 Tage	Mit 27.01.16	Die 02.02.16	
34	Arbeitspaket 1	Grafiken erstellen (Rennstrrecke, Aufbau (bemaßt)), Signalfluss	10 Tage	Mit 27.01.16	Sam 06.02.16	11
35	Arbeitspaket 1	Berechnungen für Höhe der Kamera+Licht abschließen	10 Tage	Mit 27.01.16	Sam 06.02.16	
36	Arbeitspaket 1	Arbeitspaket 1 abgeschlossen	0 Tage	Sam 06.02.16	Sam 06.02.16	29;30;31;32;3
37						
38	Arbeitspaket 2	Systembeschreibung abschließen	6 Tage	Son 07.02.16	Sam 13.02.16	36
39	Arbeitspaket 2	Arbeitspaket 2 abgeschlossen	0 Tage	Sam 13.02.16	Sam 13.02.16	38
40						
41	Arbeitspaket 3	Anforderungen definieren	8 Tage	Son 14.02.16	Mon 22.02.16	39

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



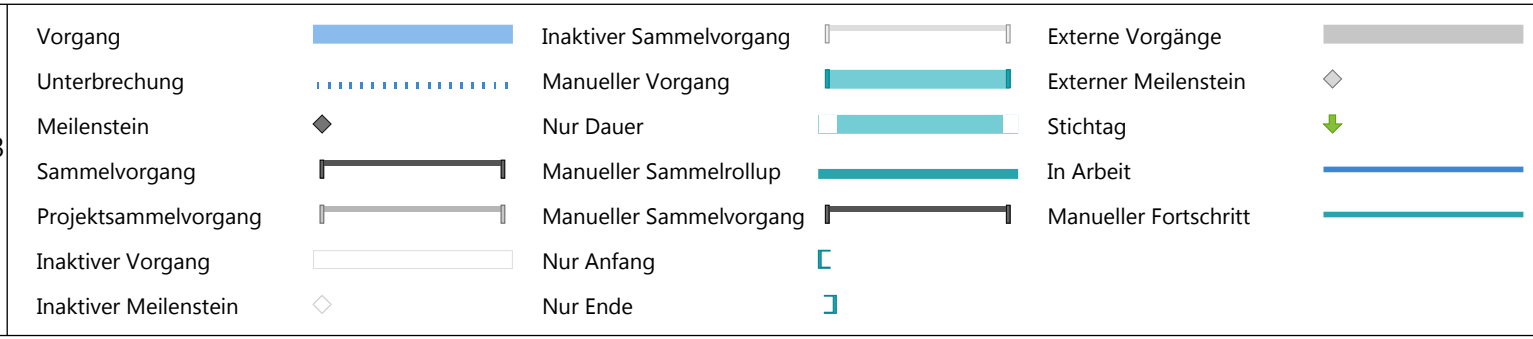
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
42	Arbeitspaket 3	Anforderungen beschreiben	8 Tage	Son 14.02.16	Mon 22.02.16	39
43	Arbeitspaket 3	Präsentationsfolien erstellen	12 Tage	Die 23.02.16	Son 06.03.16	
44	Arbeitspaket 3	Arbeitspaket 3 abgeschlossen	0 Tage	Mon 22.02.16	Mon 22.02.16	41;42
45						
46	Arbeitspaket 4	Testen Kapitel erstellen	4 Tage	Die 23.02.16	Sam 27.02.16	44
47	Arbeitspaket 4	Arbeitspaket 4 abgeschlossen	0 Tage	Sam 27.02.16	Sam 27.02.16	46
48						
49	Arbeitspaket 5	Aktueller Stand & Ausblick erstellen	2 Tage	Son 28.02.16	Die 01.03.16	47
50	Arbeitspaket 5	Projektmanagementteil schreiben	2 Tage	Son 28.02.16	Die 01.03.16	10
51	Arbeitspaket 5	Arbeitspaket 5 abgeschlossen	0 Tage	Die 01.03.16	Die 01.03.16	49;50
52	Bericht	Anforderungsdefinition Erstversion beendet	0 Tage	Mit 02.03.16	Mit 02.03.16	36;39;44;47;5
53	Probereview	Probereview	0 Tage	Mon 07.03.16	Mon 07.03.16	51
54	Bericht	Anforderungsdefinition Druck & Abgabe	0 Tage	Mit 09.03.16	Mit 09.03.16	
55	**REVIEW**	**Erstes Review**	**0 Tage**	**Mon 07.03.16**	**Mon 07.03.16**	**27;52;43;18**
56						
57	Schnittstellendefinition	Schnittstellendefinition & Softwareachritekturentwurf	29 Tage	Don 17.03.16	Fre 15.04.16	
58	Sprint 1.1	Installation & Einarbeitung OpenCV	31 Tage	Don 17.03.16	Son 17.04.16	15
59	Sprint 1.1	Realisierung HW 2	29 Tage	Don 17.03.16	Fre 15.04.16	
60	Sprint 1.1	Realisierung SW 1.1	29 Tage	Don 17.03.16	Fre 15.04.16	
61	Sprint 1.1	Realisierung SW 1.2.1	29 Tage	Don 17.03.16	Fre 15.04.16	
62	Sprint 1.1	Testfalldefinition & Dokumentation SW 1.1	43 Tage	Don 17.03.16	Fre 29.04.16	
63	Sprint 1.1	Testfalldefinition & Dokumentation SW 1.2.1	43 Tage	Don 17.03.16	Fre 29.04.16	

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



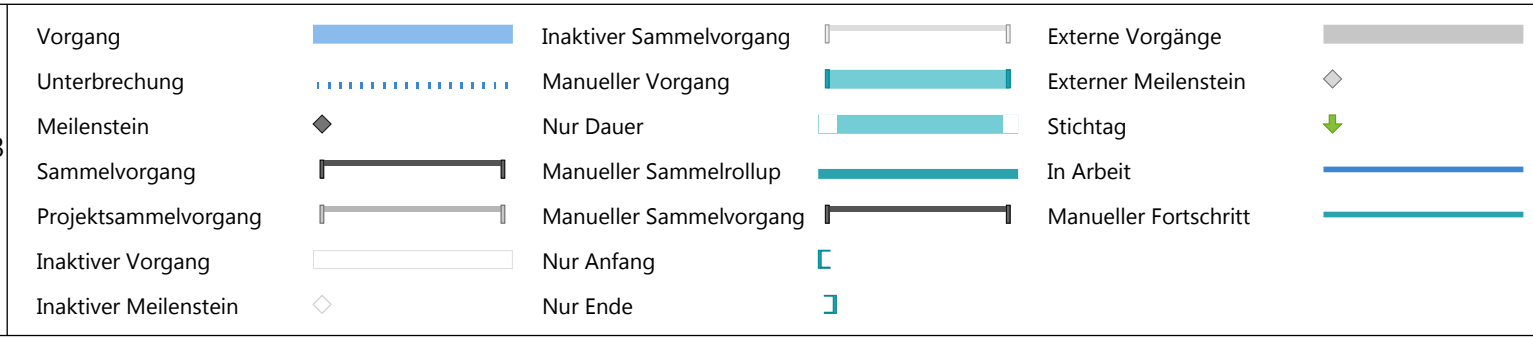
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
64	Sprint 1.1	Realisierung SW 1.2.3	14 Tage	Fre 15.04.16	Fre 29.04.16	
65	Sprint 1.1	Realisierung SW 1.2.4	14 Tage	Fre 15.04.16	Fre 29.04.16	
66	Sprint 1.1	Realisierung SW 1.2.5	14 Tage	Fre 15.04.16	Fre 29.04.16	
67	Sprint 1.1	Testfalldefinition & Dokumentation SW 1.2.3	14 Tage	Fre 15.04.16	Fre 29.04.16	
68	Sprint 1.1	Testfalldefinition & Dokumentation SW 1.2.4	14 Tage	Fre 15.04.16	Fre 29.04.16	
69	Sprint 1.1	Testfalldefinition & Dokumentation SW 1.2.5	14 Tage	Fre 15.04.16	Fre 29.04.16	
70	Sprint 1.1	Kontrolle & Ergebnisdokumentation der Testfälle	6 Tage	Fre 29.04.16	Don 05.05.16	62;63;67;68;6
71	Sprint 1.1	Kapitel 4.2 erstellen	6 Tage	Fre 29.04.16	Don 05.05.16	
72	Sprint 1.1	Sprint 1.1 abgeschlossen	0 Tage	Don 05.05.16	Don 05.05.16	70;71
73						
74	Sprint 1.2	Installation & Einarbeitung SCADE	17 Tage	Die 29.03.16	Fre 15.04.16	
75	Sprint 1.2	Realisierung HW 3	17 Tage	Die 29.03.16	Fre 15.04.16	
76	Sprint 1.2	Realisierung NHW 1	24 Tage	Die 29.03.16	Fre 22.04.16	
77	Sprint 1.2	Realisierung SW 2.2	24 Tage	Die 29.03.16	Fre 22.04.16	
78	Sprint 1.2	Testfalldefinition & Dokumentation SW 2.2	14 Tage	Fre 15.04.16	Fre 29.04.16	
79	Sprint 1.2	Realisierung SW 2.1.1	7 Tage	Fre 29.04.16	Fre 06.05.16	
80	Sprint 1.2	Testfalldefinition & Dokumentation SW 2.1.1	7 Tage	Fre 29.04.16	Fre 06.05.16	
81	Sprint 1.2	Kontrolle & Ergebnisdokumentation der Testfälle	6 Tage	Fre 06.05.16	Don 12.05.16	78;80
82	Sprint 1.2	Kapitel 4.3 erstellen	6 Tage	Fre 06.05.16	Don 12.05.16	
83	Sprint 1.2	Sprint 2.1 abgeschlossen	0 Tage	Don 12.05.16	Don 12.05.16	81;82
84						
85	Sprint 2.1	Kapitel 3.1 erstellen (Stichwörter/grob)	7 Tage	Fre 06.05.16	Fre 13.05.16	

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



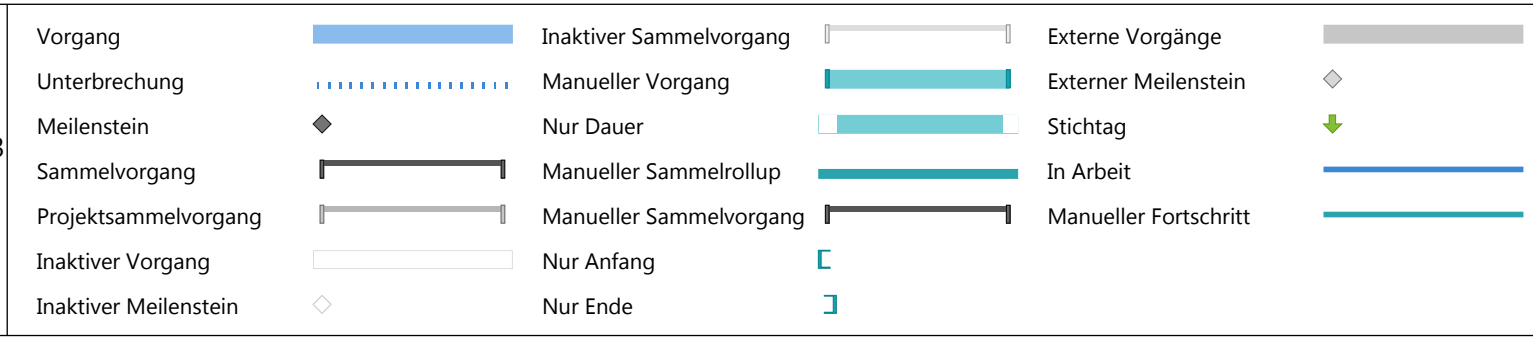
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
86	Sprint 2.1	Kapitel 3.2 erstellen (Stichwörter/grob)	7 Tage	Fre 06.05.16	Fre 13.05.16	
87	Sprint 2.1	Kapitel 3.4 erstellen (Stichwörter/grob)	7 Tage	Fre 06.05.16	Fre 13.05.16	
88	Sprint 2.1	Realisiering SW1.5 (zuerst bearbeiten)	28 Tage	Fre 13.05.16	Fre 10.06.16	
89	Sprint 2.1	Realisiering SW1.6	28 Tage	Fre 13.05.16	Fre 10.06.16	
90	Sprint 2.1	Realisiering SW1.2.4	28 Tage	Fre 13.05.16	Fre 10.06.16	
91	Sprint 2.1	Realisiering SW1.2.5	28 Tage	Fre 13.05.16	Fre 10.06.16	
92	Sprint 2.1	Testfalldefinition & Dokumentation SW 1.2.4	7 Tage	Fre 10.06.16	Fre 17.06.16	
93	Sprint 2.1	Testfalldefinition & Dokumentation SW 1.2.5	7 Tage	Fre 10.06.16	Fre 17.06.16	
94	Sprint 2.1	Testfalldefinition & Dokumentation SW 1.5	7 Tage	Fre 10.06.16	Fre 17.06.16	
95	Sprint 2.1	Testfalldefinition & Dokumentation SW 1.6	7 Tage	Fre 10.06.16	Fre 17.06.16	
96	Sprint 2.1	Kontrolle & Ergebnisdokumentation der Testfälle	7 Tage	Fre 17.06.16	Fre 24.06.16	94;95;93;92
97	Sprint 2.1	Sprint 2.1 abgeschlossen	0 Tage	Fre 24.06.16	Fre 24.06.16	96
98						
99	Sprint 2.2	Kapitel 3.3 erstellen (Stichwörter/grob)	7 Tage	Fre 13.05.16	Fre 20.05.16	
100	Sprint 2.2	Kapitel 3.5 erstellen (Stichwörter/grob)	7 Tage	Fre 13.05.16	Fre 20.05.16	
101	Sprint 2.2	Kapitel 3.6 erstellen (Stichwörter/grob)	7 Tage	Fre 13.05.16	Fre 20.05.16	
102	Sprint 2.2	Realisiering SW2.4 (zuerst bearbeiten)	21 Tage	Fre 20.05.16	Fre 10.06.16	
103	Sprint 2.2	Realisiering SW 1.3	21 Tage	Fre 20.05.16	Fre 10.06.16	
104	Sprint 2.2	Realisiering SW 2.1.7	21 Tage	Fre 20.05.16	Fre 10.06.16	
105	Sprint 2.2	Realisiering SW 2.1.8.1	21 Tage	Fre 20.05.16	Fre 10.06.16	
106	Sprint 2.2	Realisiering SW 2.1.8.2	21 Tage	Fre 20.05.16	Fre 10.06.16	
107	Sprint 2.2	Realisiering SW 2.1.8.5	21 Tage	Fre 20.05.16	Fre 10.06.16	

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



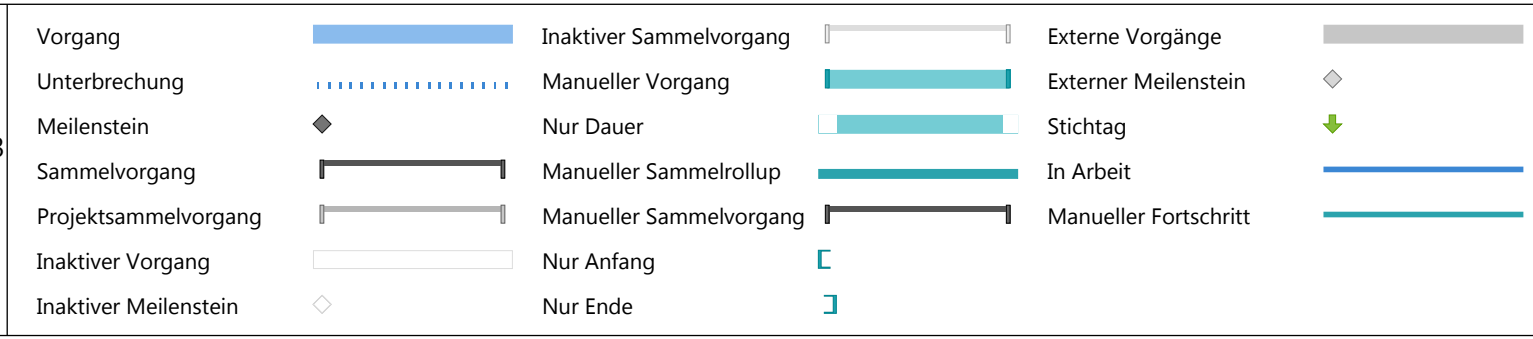
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
108	Sprint 2.2	Realisiering SW 2.1.8.6	21 Tage	Fre 20.05.16	Fre 10.06.16	
109	Sprint 2.2	Realisiering SW 2.1.8.8	21 Tage	Fre 20.05.16	Fre 10.06.16	
110	Sprint 2.2	Realisiering SW 2.1.9	21 Tage	Fre 20.05.16	Fre 10.06.16	
111	Sprint 2.2	Testfalldefinition & Dokumentation SW 1.3	7 Tage	Fre 10.06.16	Fre 17.06.16	
112	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.7	7 Tage	Fre 10.06.16	Fre 17.06.16	
113	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.8.1	7 Tage	Fre 10.06.16	Fre 17.06.16	
114	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.8.2	7 Tage	Fre 10.06.16	Fre 17.06.16	
115	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.8.5	7 Tage	Fre 10.06.16	Fre 17.06.16	
116	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.8.6	7 Tage	Fre 10.06.16	Fre 17.06.16	
117	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.8.8	7 Tage	Fre 10.06.16	Fre 17.06.16	
118	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.1.9	7 Tage	Fre 10.06.16	Fre 17.06.16	
119	Sprint 2.2	Testfalldefinition & Dokumentation SW 2.4	7 Tage	Fre 10.06.16	Fre 17.06.16	
120	Sprint 2.2	Kontrolle & Ergebnisdokumentation der Testfälle	7 Tage	Fre 17.06.16	Fre 24.06.16	111;112;113;1
121	Sprint 2.2	Sprint 2.2 abgeschlossen	0 Tage	Fre 24.06.16	Fre 24.06.16	120
122						
123	Schulung	Scade Schulung I	0 Tage	Die 15.03.16	Die 15.03.16	
124	Schulung	Scade Schulung II	0 Tage	Don 24.03.16	Don 24.03.16	123
125	Schulung	Scade Schulung III	0 Tage	Fre 15.04.16	Fre 15.04.16	124
126						
127	Bericht - Arbeitspaket 1	Kapitel Systemarchitektur	3 Tage	Fre 24.06.16	Mon 27.06.16	
128	Bericht - Arbeitspaket 2	Kapitel Schnittstellen	2 Tage	Mon 27.06.16	Mit 29.06.16	127
129	Bericht - Arbeitspaket 3	Kapitel Hardwareevaluation	2 Tage	Mit 29.06.16	Fre 01.07.16	

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



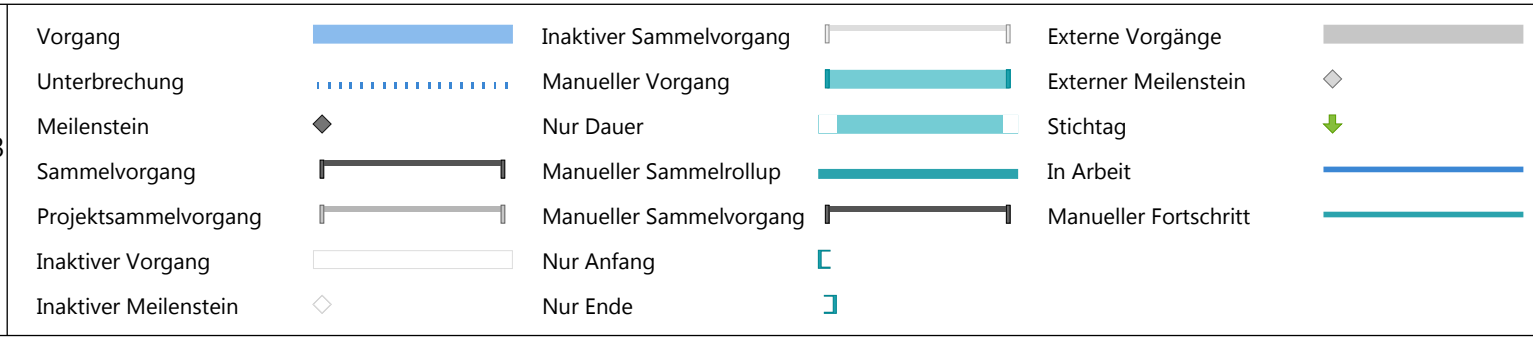
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
130	Bericht - Arbeitspaket 4	Kapitel Testen	2 Tage	Fre 01.07.16	Son 03.07.16	
131	Bericht - Arbeitspaket 4	Kapitel Einleitung	2 Tage	Fre 01.07.16	Son 03.07.16	
132	Bericht - Arbeitspaket 5	Kapitel Aktueller Stand & Ausblick	1 Tag	Mon 27.06.16	Die 28.06.16	
133	Bericht - Arbeitspaket 1	Zweiten Bericht vervollständigt	0 Tage	Don 07.07.16	Don 07.07.16	74;83;72;85;8
134	Präsentation	Präsentationsfolien erstellen	4 Tage	Mit 29.06.16	Son 03.07.16	
135	Präsentation	Präsentationsfolien überarbeiten	4 Tage	Son 03.07.16	Don 07.07.16	134
136						
137	Probereview	Probereview	0 Tage	Son 03.07.16	Son 03.07.16	134
138	Bericht	Zweiter Bericht Druck & Abgabe	0 Tage	Don 07.07.16	Don 07.07.16	133
139	**REVIEW**	**Zweites Review**	**0 Tage**	**Don 07.07.16**	**Don 07.07.16**	**138**
140						
141	Sprint 3.1	Realisiering SW 1.2.5	81 Tage	Mon 11.07.16	Fre 30.09.16	
142	Sprint 3.1	Realisiering SW 1.4	81 Tage	Mon 11.07.16	Fre 30.09.16	
143	Sprint 3.1	Realisiering SW 1.6.2	81 Tage	Mon 11.07.16	Fre 30.09.16	
144	Sprint 3.1	Realisiering SW4	81 Tage	Mon 11.07.16	Fre 30.09.16	
145	Sprint 3.1	Realisiering SW5	81 Tage	Mon 11.07.16	Fre 30.09.16	
146	Sprint 3.1	Realisiering NSW1.1	81 Tage	Mon 11.07.16	Fre 30.09.16	
147	Sprint 3.1	Realisiering NSW1.2	81 Tage	Mon 11.07.16	Fre 30.09.16	
148	Sprint 3.1	Realisiering NSW1.7	81 Tage	Mon 11.07.16	Fre 30.09.16	
149	Sprint 3.1	Aufräumarbeiten	81 Tage	Mon 11.07.16	Fre 30.09.16	
150	Sprint 3.1	Implementierung korrekter Fehlercodes	81 Tage	Mon 11.07.16	Fre 30.09.16	
151	Sprint 3.1	Testfalldefinition & Dokumentation	7 Tage	Fre 30.09.16	Fre 07.10.16	141;142;143;1

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



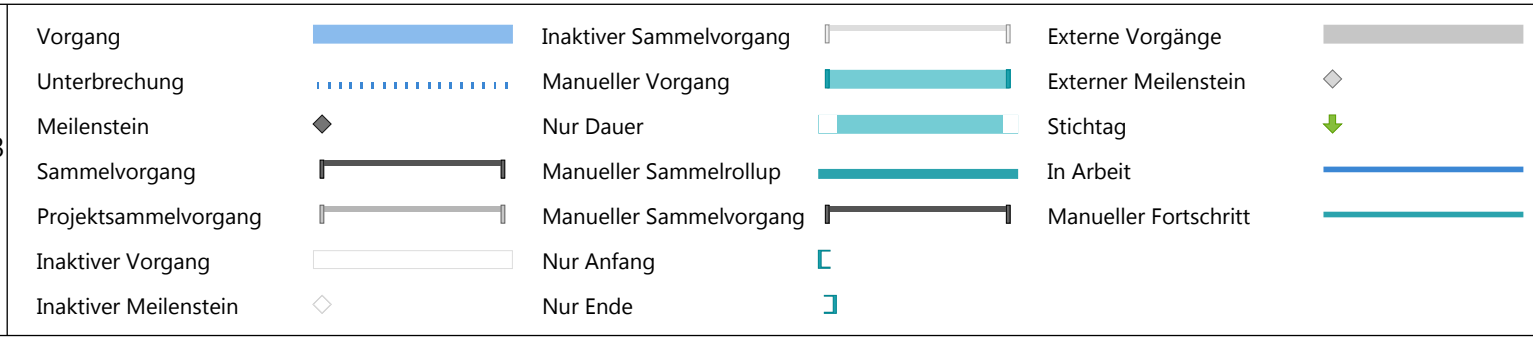
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
152	Sprint 3.1	Kontrolle & Ergebnisdokumentation der Testfälle	7 Tage	Fre 30.09.16	Fre 07.10.16	141;142;143;1
153	Sprint 3.1	Dokumentation der Entwicklung im Bericht	7 Tage	Fre 30.09.16	Fre 07.10.16	141;142;143;1
154	Sprint 3.1	Sprint 3.1 abgeschlossen	0 Tage	Fre 07.10.16	Fre 07.10.16	153;152
155						
156	Sprint 3.2	Realisiering SW 3	32 Tage	Mon 11.07.16	Fre 12.08.16	
157	Sprint 3.2	Realisiering SW 5	32 Tage	Mon 11.07.16	Fre 12.08.16	
158	Sprint 3.2	Realisiering SW 5	32 Tage	Mon 11.07.16	Fre 12.08.16	
159	Sprint 3.2	Realisiering NSW 1.7	32 Tage	Mon 11.07.16	Fre 12.08.16	
160	Sprint 3.2	Testfalldefinition & Dokumentation	7 Tage	Fre 12.08.16	Fre 19.08.16	156;157;158;1
161	Sprint 3.2	Kontrolle & Ergebnisdokumentation der Testfälle	7 Tage	Fre 19.08.16	Fre 26.08.16	160
162	Sprint 3.2	Dokumentation der Entwicklung im Bericht	7 Tage	Fre 19.08.16	Fre 26.08.16	160
163	Sprint 3.2	Sprint 3.3 abgeschlossen	0 Tage	Fre 26.08.16	Fre 26.08.16	162;161
164						
165	Sprint 3.3	Realisiering SW 2.1.10.3	32 Tage	Mon 11.07.16	Fre 12.08.16	
166	Sprint 3.3	Realisiering SW 2.1.10.4	32 Tage	Mon 11.07.16	Fre 12.08.16	
167	Sprint 3.3	Realisiering SW 2.1.2	32 Tage	Mon 11.07.16	Fre 12.08.16	
168	Sprint 3.3	Realisiering SW 2.1.3	32 Tage	Mon 11.07.16	Fre 12.08.16	
169	Sprint 3.3	Realisiering SW 2.1.5	32 Tage	Mon 11.07.16	Fre 12.08.16	
170	Sprint 3.3	Recherchen zu Fahrzeugmodellen	32 Tage	Mon 11.07.16	Fre 12.08.16	
171	Sprint 3.3	Umsetzung Koordinaten und Kursprädiktion	32 Tage	Mon 11.07.16	Fre 12.08.16	
172	Sprint 3.3	Testfalldefinition & Dokumentation	7 Tage	Fre 12.08.16	Fre 19.08.16	165;166;167;1
173	Sprint 3.3	Kontrolle & Ergebnisdokumentation der Testfälle	7 Tage	Fre 19.08.16	Fre 26.08.16	172

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



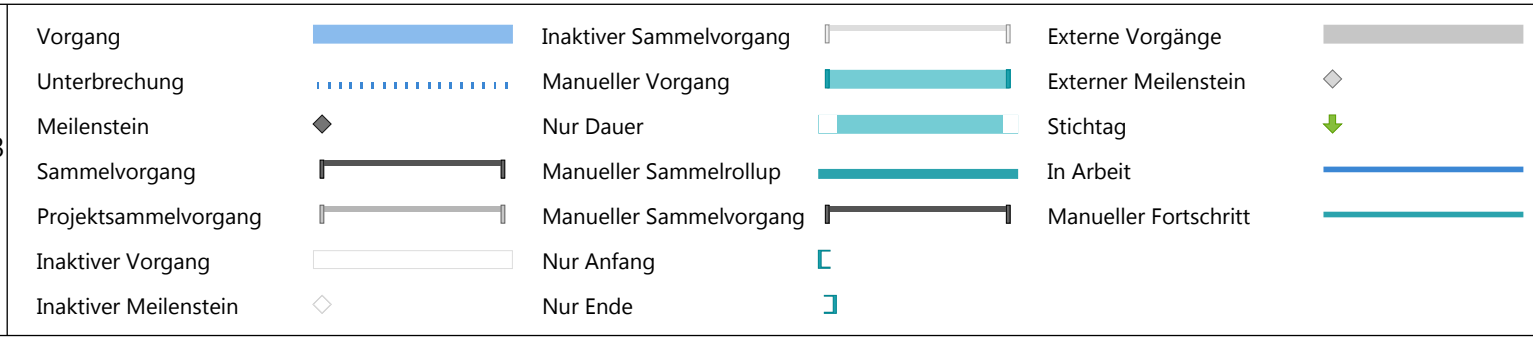
Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
174	Sprint 3.3	Dokumentation der Entwicklung im Bericht	7 Tage	Fre 19.08.16	Fre 26.08.16	172
175	Sprint 3.3	Sprint 3.2 abgeschlossen	0 Tage	Fre 26.08.16	Fre 26.08.16	174;173
176						
177	Sprint 4.1	Realisiering SW 2.1.10	28 Tage	Fre 26.08.16	Fre 23.09.16	
178	Sprint 4.1	Realisiering SW 2.3	28 Tage	Fre 26.08.16	Fre 23.09.16	
179	Sprint 4.1	Realisiering SW 2.1.3	28 Tage	Fre 26.08.16	Fre 23.09.16	
180	Sprint 4.1	Realisiering SW 2.1.7	28 Tage	Fre 26.08.16	Fre 23.09.16	
181	Sprint 4.1	Realisiering SW 2.1.8	28 Tage	Fre 26.08.16	Fre 23.09.16	
182	Sprint 4.1	Realisiering SW 2.1.10.7	28 Tage	Fre 26.08.16	Fre 23.09.16	
183	Sprint 4.1	Realisiering SW4	28 Tage	Fre 26.08.16	Fre 23.09.16	
184	Sprint 4.1	Realisiering SW5	28 Tage	Fre 26.08.16	Fre 23.09.16	
185	Sprint 4.1	Realisiering SW 2.1.10.9	28 Tage	Fre 26.08.16	Fre 23.09.16	
186	Sprint 4.1	Realisiering NSW1.3	28 Tage	Fre 26.08.16	Fre 23.09.16	
187	Sprint 4.1	Realisiering NSW1.4	28 Tage	Fre 26.08.16	Fre 23.09.16	
188	Sprint 4.1	Realisiering NSW1.5	28 Tage	Fre 26.08.16	Fre 23.09.16	
189	Sprint 4.1	Realisiering NSW1.6	28 Tage	Fre 26.08.16	Fre 23.09.16	
190	Sprint 4.1	Realisiering NSW1.7	28 Tage	Fre 26.08.16	Fre 23.09.16	
191	Sprint 4.1	Implementierung Fahrzeugmodell	28 Tage	Fre 26.08.16	Fre 23.09.16	
192	Sprint 4.1	Testfalldefinition & Dokumentation	7 Tage	Fre 23.09.16	Fre 30.09.16	177;191;182;1
193	Sprint 4.1	Kontrolle & Ergebnisdokumentation der Testfälle	7 Tage	Fre 30.09.16	Fre 07.10.16	192
194	Sprint 4.1	Dokumentation der Entwicklung im Bericht	7 Tage	Fre 30.09.16	Fre 07.10.16	192
195	Sprint 4.1	Sprint 4.1 abgeschlossen	0 Tage	Fre 07.10.16	Fre 07.10.16	194;193

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

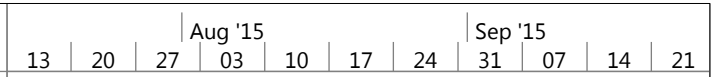


Nr.	Gruppe	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
196						
197	Präsentation	Präsentationsfolien erstellen	11 Tage	Fre 07.10.16	Die 18.10.16	
198	Präsentation	Präsentationsfolien überarbeiten	1 Tag	Die 18.10.16	Mit 19.10.16	197
199	Probereview	Probereview	0 Tage	Die 18.10.16	Die 18.10.16	197
200	Bericht	Zweiter Bericht Druck & Abgabe	0 Tage	Mit 19.10.16	Mit 19.10.16	133
201	**REVIEW**	**Drittes Review**	**0 Tage**	**Don 20.10.16**	**Don 20.10.16**	**154;175;163;1**

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	*Erledigt*	*Anatolij Fandrich;Michael Buk*	*Nein*	*NV*
Nein	*Erledigt*	*Anatolij Fandrich;Michael Buk*	*Nein*	*NV*
Nein	*Erledigt*	*Michael Bukowski*	*Nein*	*NV*
Nein	*Erledigt*	*Anatolij Fandrich*	*Nein*	*NV*
Nein	*Erledigt*	*Michael Bukowski*	*Nein*	*NV*
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Michael Bukowski	Nein	NV
Ja	Erledigt	Michael Bukowski	Nein	NV
Nein	Erledigt	Tom Reske	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Bukowski;Nikolai Bräuer;Tom	Nein	NV
Ja	Erledigt	Anatolij Fandrich	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Betreuer;Mi	Nein	NV
Ja	Erledigt	Anatolij Fandrich	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Nikolai Bräuer	Nein	NV

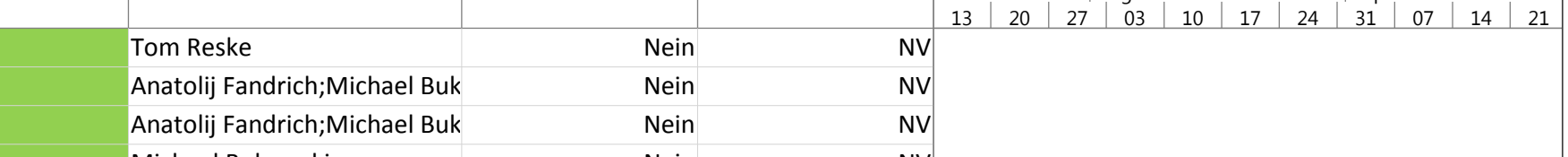


Proj

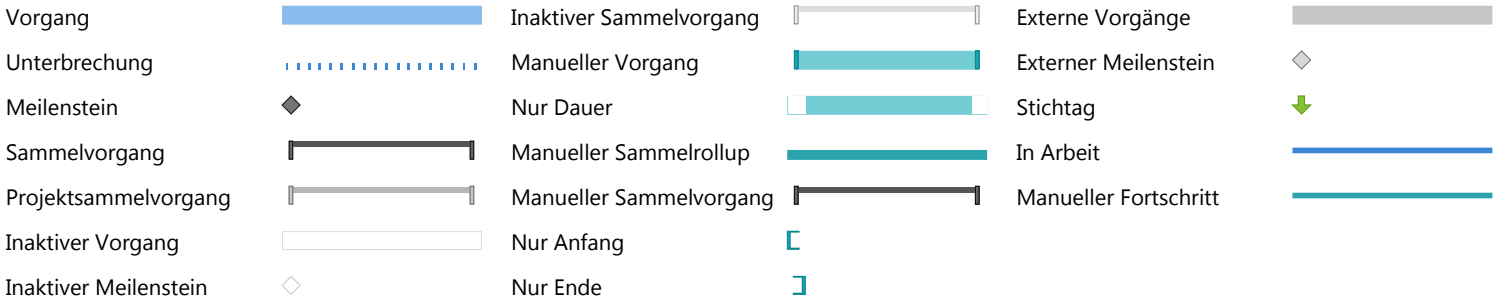
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang
Unterbrechung
Meilenstein
Sammelvorgang
Projektsammelvorgang
Inaktiver Vorgang
Inaktiver Meilenstein
Inaktiver Sammelvorgang
Manueller Vorgang
Nur Dauer
Manueller Sammelrollup
Manueller Sammelvorgang
Nur Anfang
Nur Ende
Externe Vorgänge
Externer Meilenstein
Stichtag
In Arbeit
Manueller Fortschritt

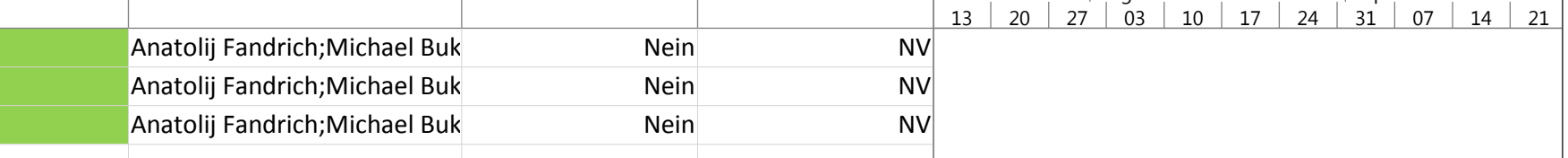
Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Ja	Erledigt	Tom Reske	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Michael Bukowski	Nein	NV
Nein	Erledigt	Michael Bukowski	Nein	NV
Nein	Erledigt	Michael Bukowski	Nein	NV
Ja	Erledigt	Michael Bukowski	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Betreuer	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Bukowski;Nikolai Bräuer	Nein	NV
Nein	Erledigt	Michael Bukowski	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Betreuer;Mi	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt		Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16



Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt		Nein	NV
Nein	Erledigt	Tom Reske	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Tom Reske	Nein	NV
Ja	Erledigt		Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt		Nein	NV
Nein	**Erledigt**	**Anatolij Fandrich;Michael Bu**	**Nein**	**Mit 09.03.16**
Nein	Erledigt	Anatolij Fandrich;Michael Bukowski;Nikolai Bräuer;Tom	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang
Unterbrechung
Meilenstein
Sammelvorgang
Projektsammelvorgang
Inaktiver Vorgang
Inaktiver Meilenstein
Inaktiver Sammelvorgang
Manueller Vorgang
Nur Dauer
Manueller Sammelrollup
Manueller Sammelvorgang
Nur Anfang
Nur Ende
Externe Vorgänge
Externer Meilenstein
Stichtag
In Arbeit
Manueller Fortschritt

Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Ja	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV

Aug '15 | Sep '15

13 | 20 | 27 | 03 | 10 | 17 | 24 | 31 | 07 | 14 | 21

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Nikolai Bräu	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV

Aug '15 Sep '15
13 20 27 03 10 17 24 31 07 14 21

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Ja	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Betreuer;Mi	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Betreuer;Mi	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Betreuer;Mi	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV

Aug '15 Sep '15
13 20 27 03 10 17 24 31 07 14 21

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang
Unterbrechung
Meilenstein
Sammelvorgang
Projektsammelvorgang
Inaktiver Vorgang
Inaktiver Meilenstein
Inaktiver Sammelvorgang
Manueller Vorgang
Nur Dauer
Manueller Sammelrollup
Manueller Sammelvorgang
Nur Anfang
Nur Ende
Externe Vorgänge
Externer Meilenstein
Stichtag
In Arbeit
Manueller Fortschritt

Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Tom Reske	Nein	NV
Nein	Erledigt	Tom Reske	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Ja	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Ja	NV
Ja	**Erledigt**	**Anatolij Fandrich;Michael Bu**	**Ja**	**Don 07.07.16**
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV

Aug '15 Sep '15
13 20 27 03 10 17 24 31 07 14 21

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	In Bearbeitung	Anatolij Fandrich	Nein	NV
Ja	In Bearbeitung	Anatolij Fandrich	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Ja	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV

Aug '15 Sep '15
13 20 27 03 10 17 24 31 07 14 21

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

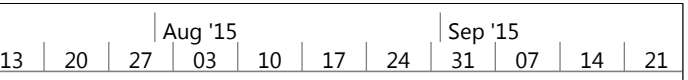
Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Ja	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Nikolai Bräuer	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Nein	Erledigt	Michael Bukowski;Tom Reske	Nein	NV
Ja	Erledigt	Michael Bukowski;Tom Reske	Nein	NV

Aug '15 Sep '15
13 20 27 03 10 17 24 31 07 14 21

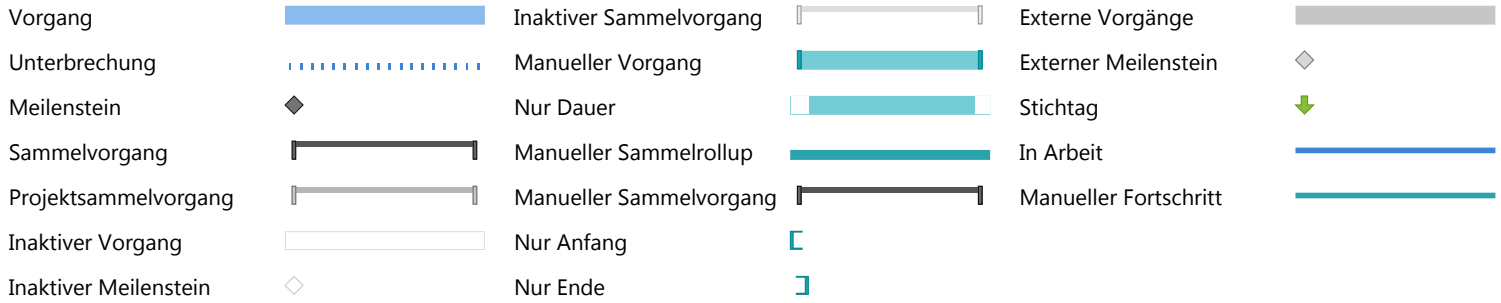
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

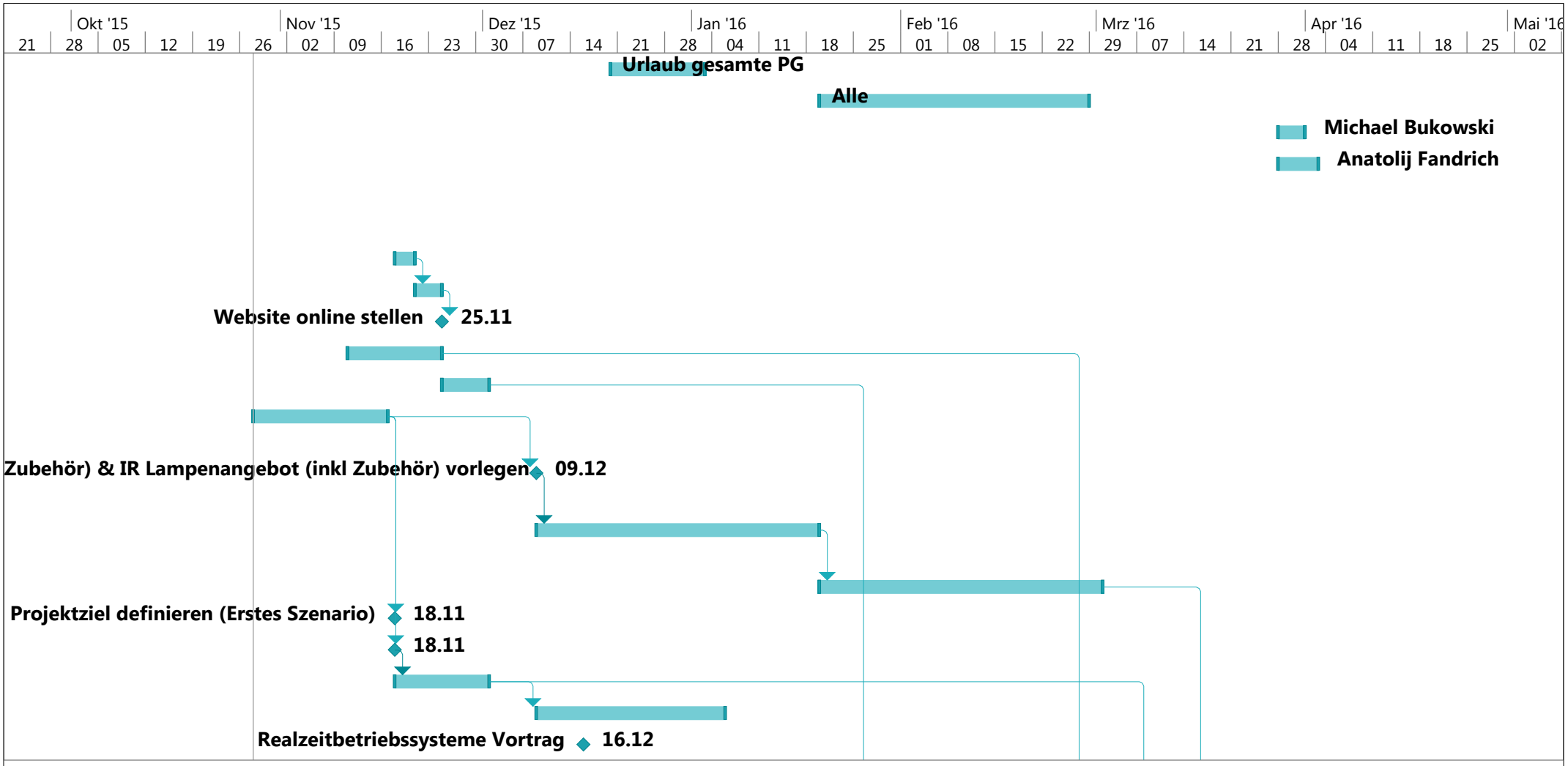
Vorgang
Unterbrechung
Meilenstein
Sammelvorgang
Projektsammelvorgang
Inaktiver Vorgang
Inaktiver Meilenstein
Inaktiver Sammelvorgang
Manueller Vorgang
Nur Dauer
Manueller Sammelrollup
Manueller Sammelvorgang
Nur Anfang
Nur Ende
Externe Vorgänge
Externer Meilenstein
Stichtag
In Arbeit
Manueller Fortschritt

Meilenstein	Zustand	Ressourcennamen	Kritisch	Stichtag
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Nein	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	Anatolij Fandrich;Michael Buk	Nein	NV
Ja	Erledigt	**Anatolij Fandrich;Michael Bu**	**Ja**	**Don 20.10.16**



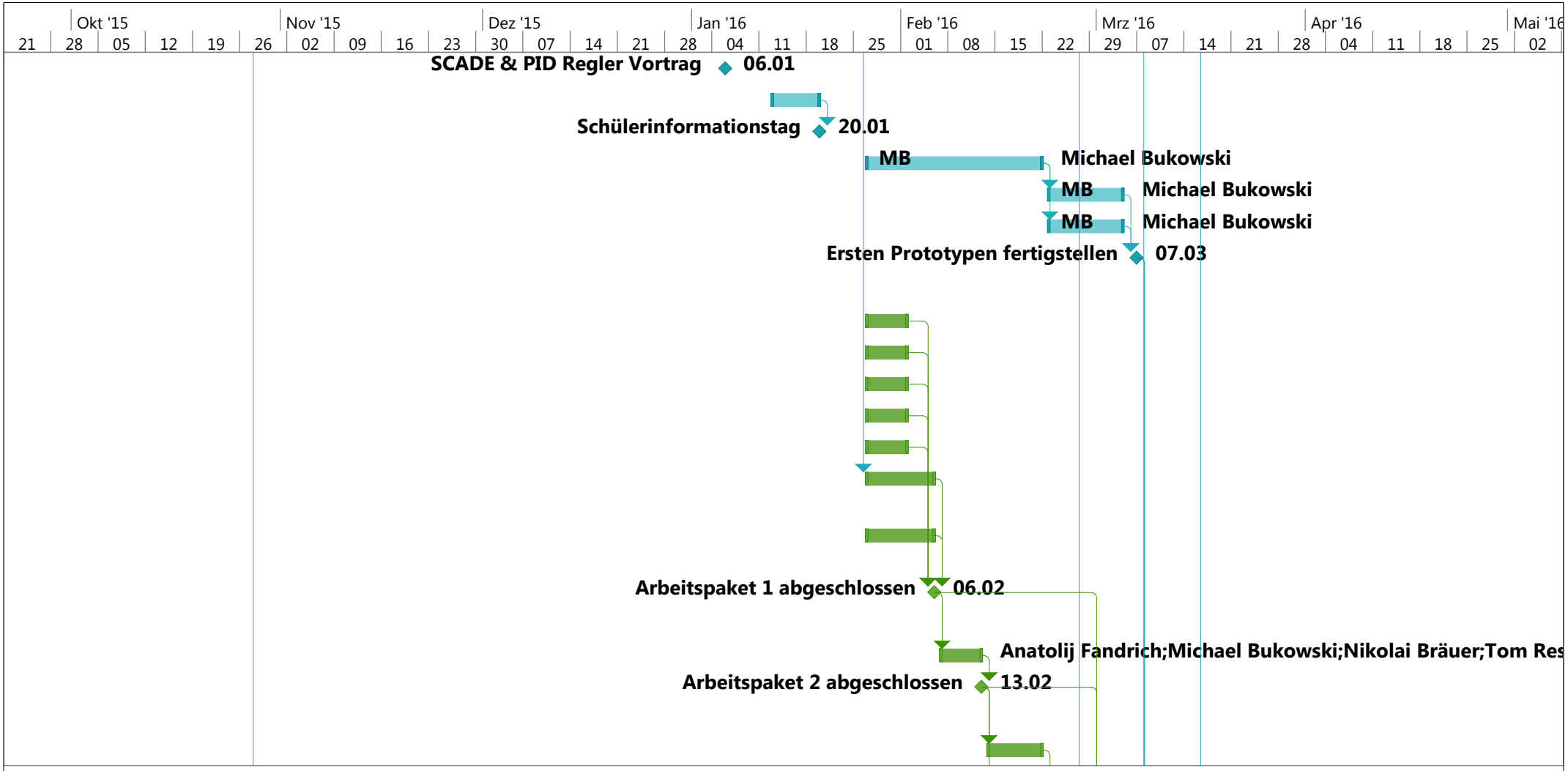
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16





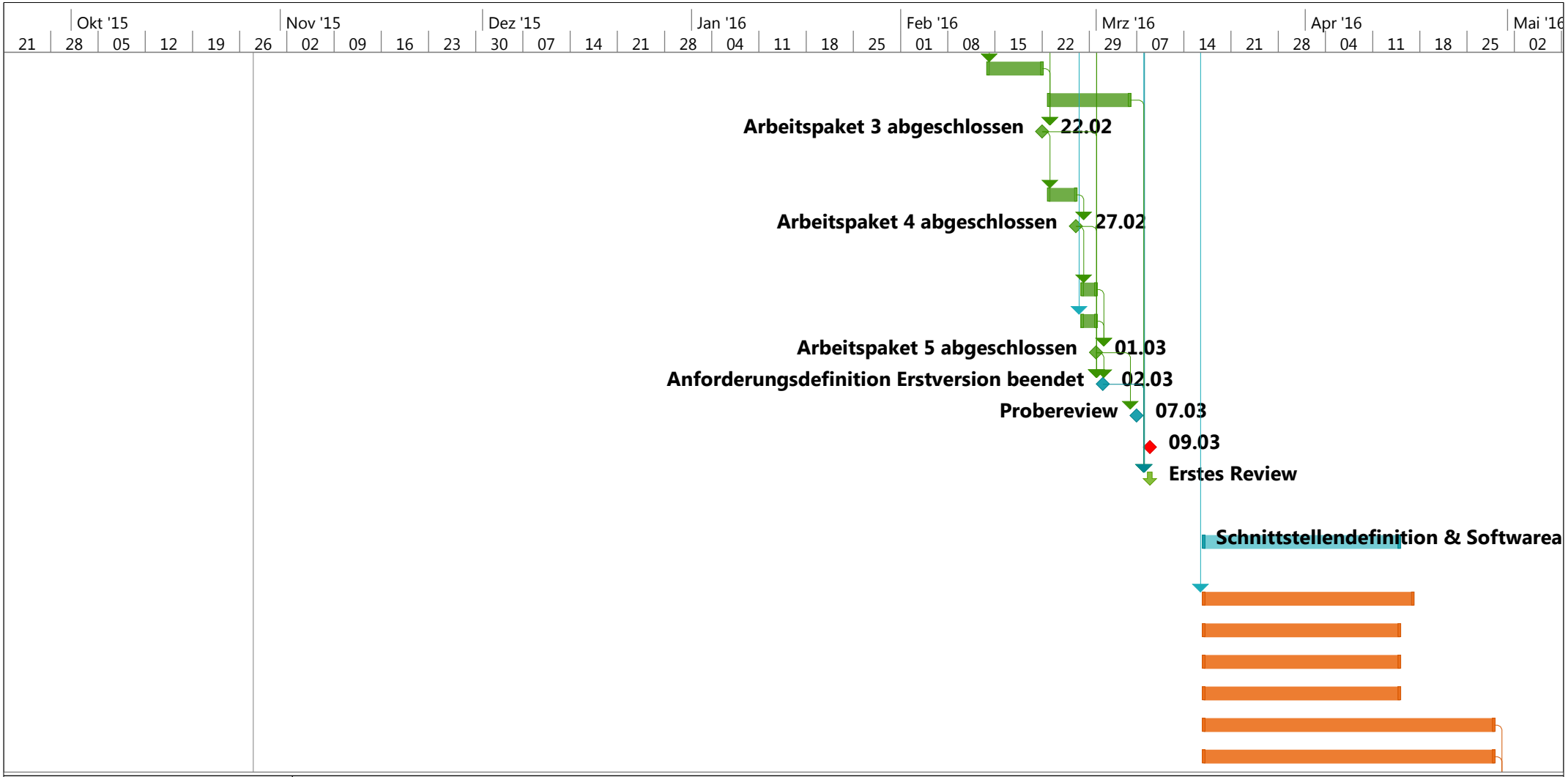
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang		Inaktiver Sammelvorgang		Externe Vorgänge	
Unterbrechung		Manueller Vorgang		Externer Meilenstein	
Meilenstein		Nur Dauer		Stichtag	
Sammelvorgang		Manueller Sammelrollup		In Arbeit	
Projektsammelvorgang		Manueller Sammelvorgang		Manueller Fortschritt	
Inaktiver Vorgang		Nur Anfang			
Inaktiver Meilenstein		Nur Ende			



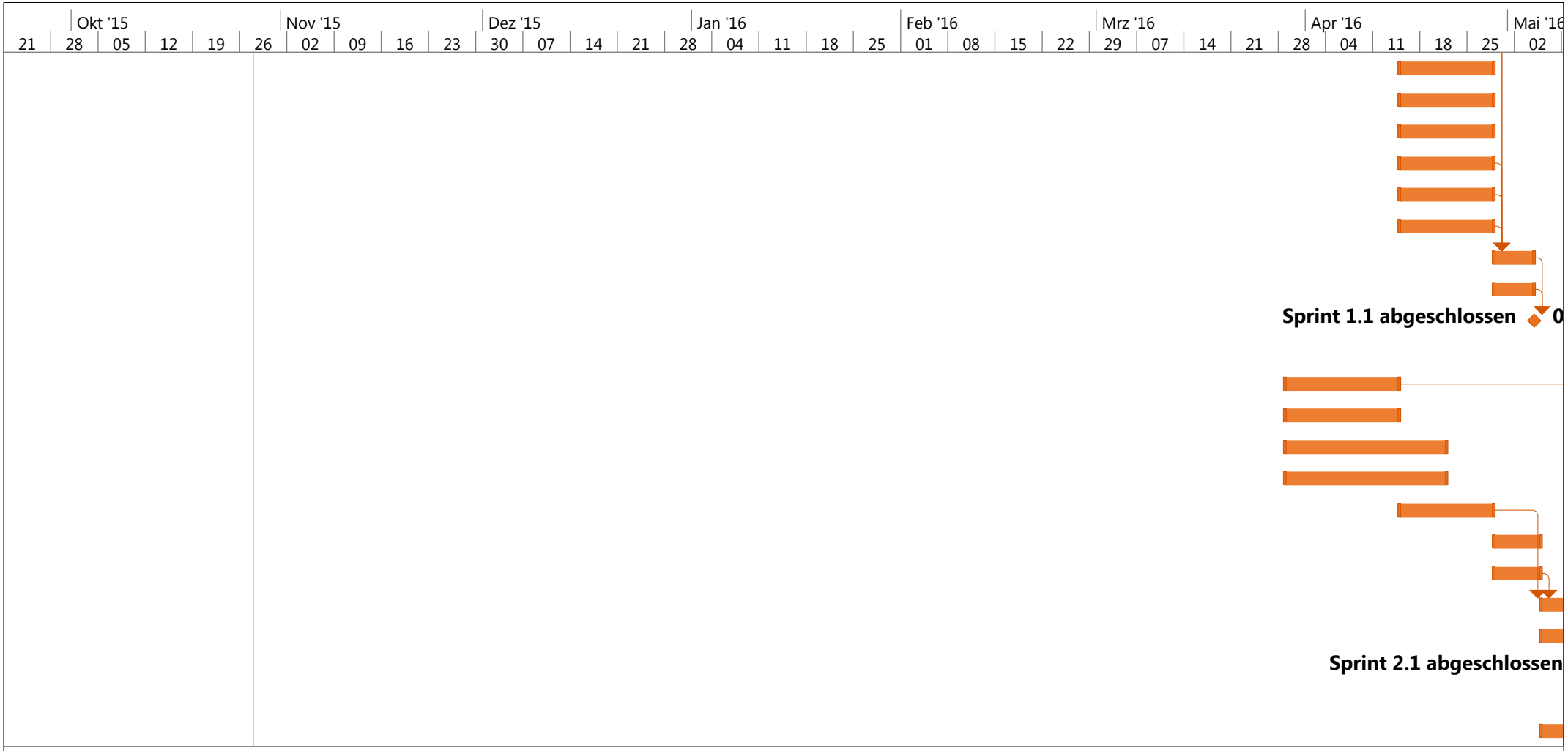
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



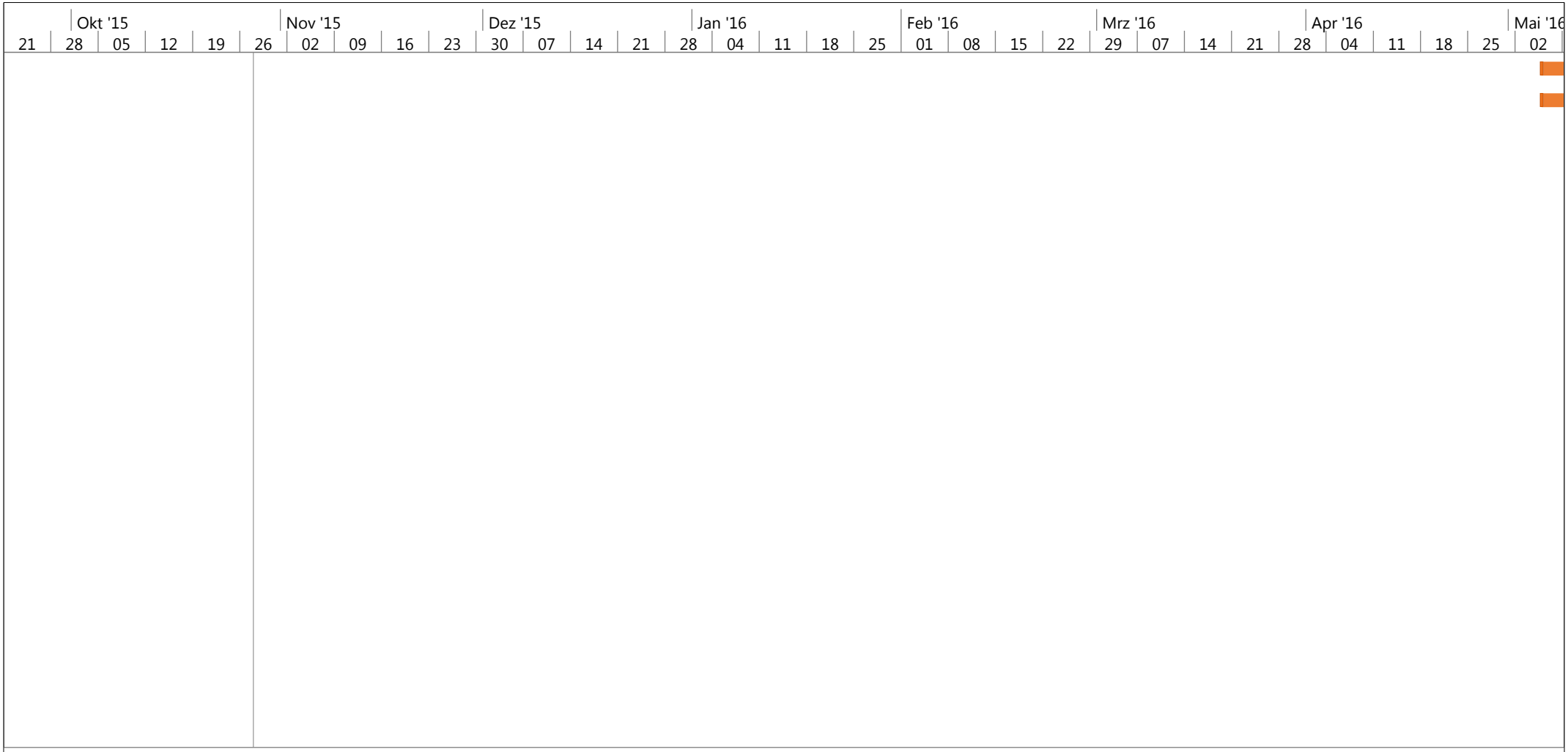
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang		Inaktiver Sammelvorgang		Externe Vorgänge	
Unterbrechung		Manueller Vorgang		Externer Meilenstein	
Meilenstein		Nur Dauer		Stichtag	
Sammelvorgang		Manueller Sammelrollup		In Arbeit	
Projektsammelvorgang		Manueller Sammelvorgang		Manueller Fortschritt	
Inaktiver Vorgang		Nur Anfang			
Inaktiver Meilenstein		Nur Ende			



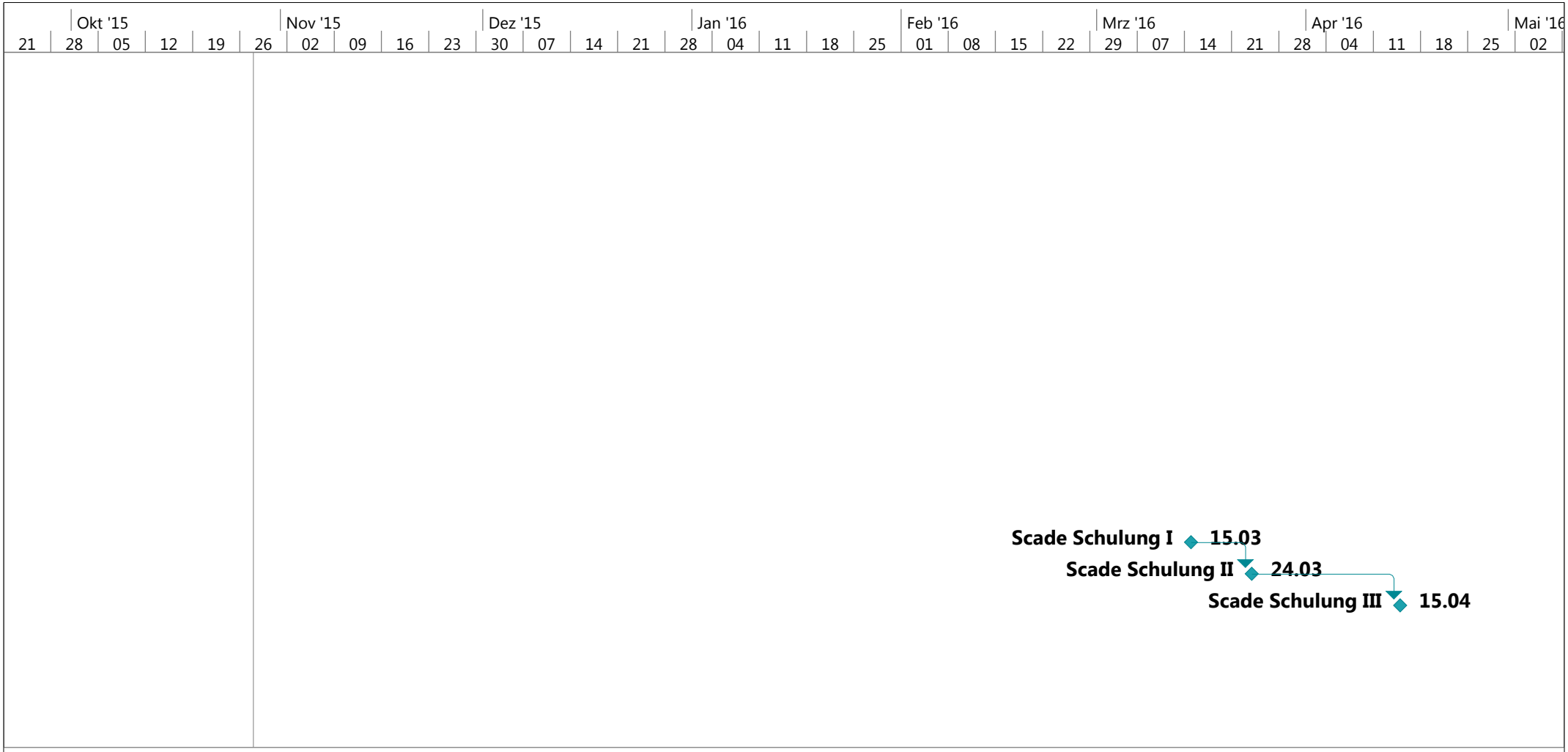
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



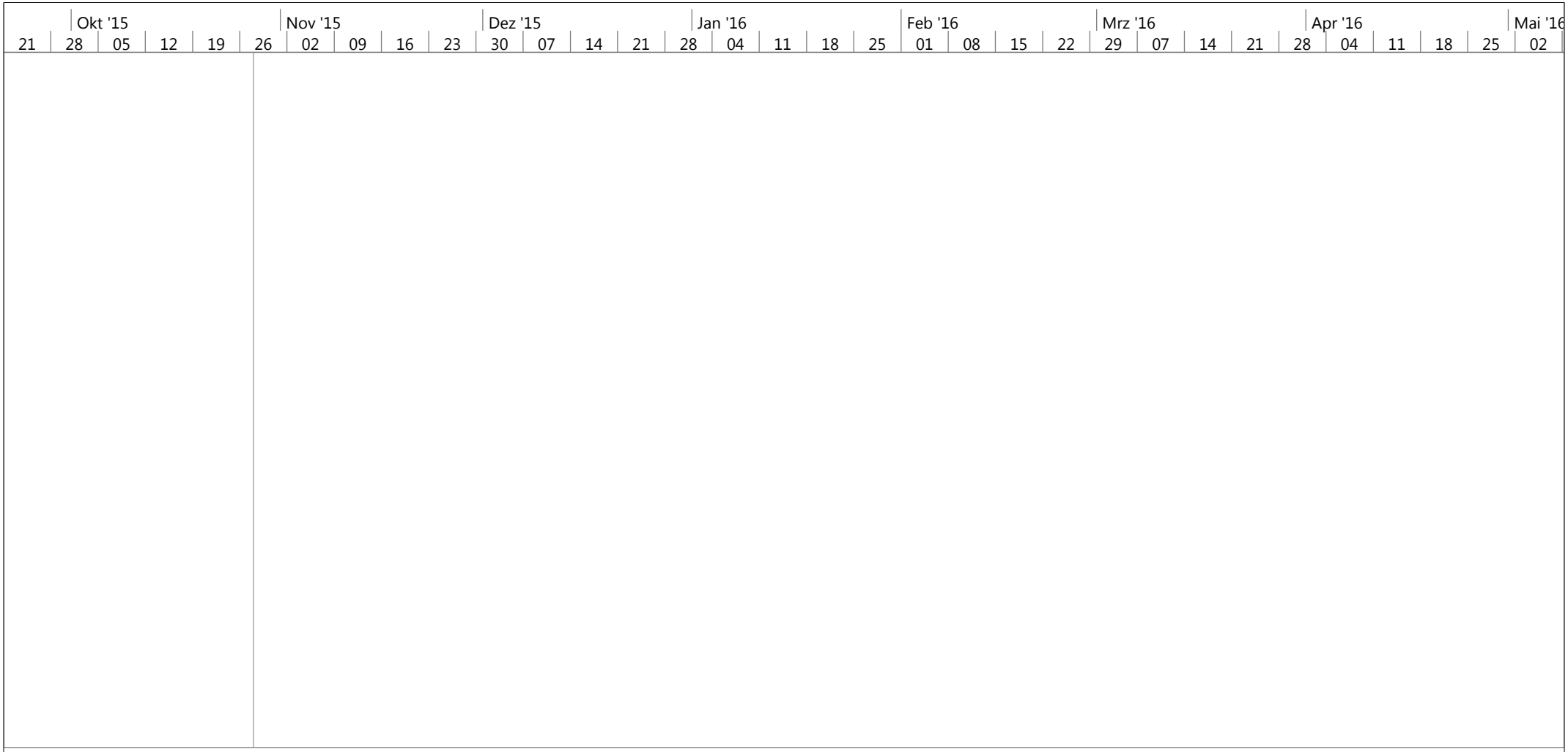
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



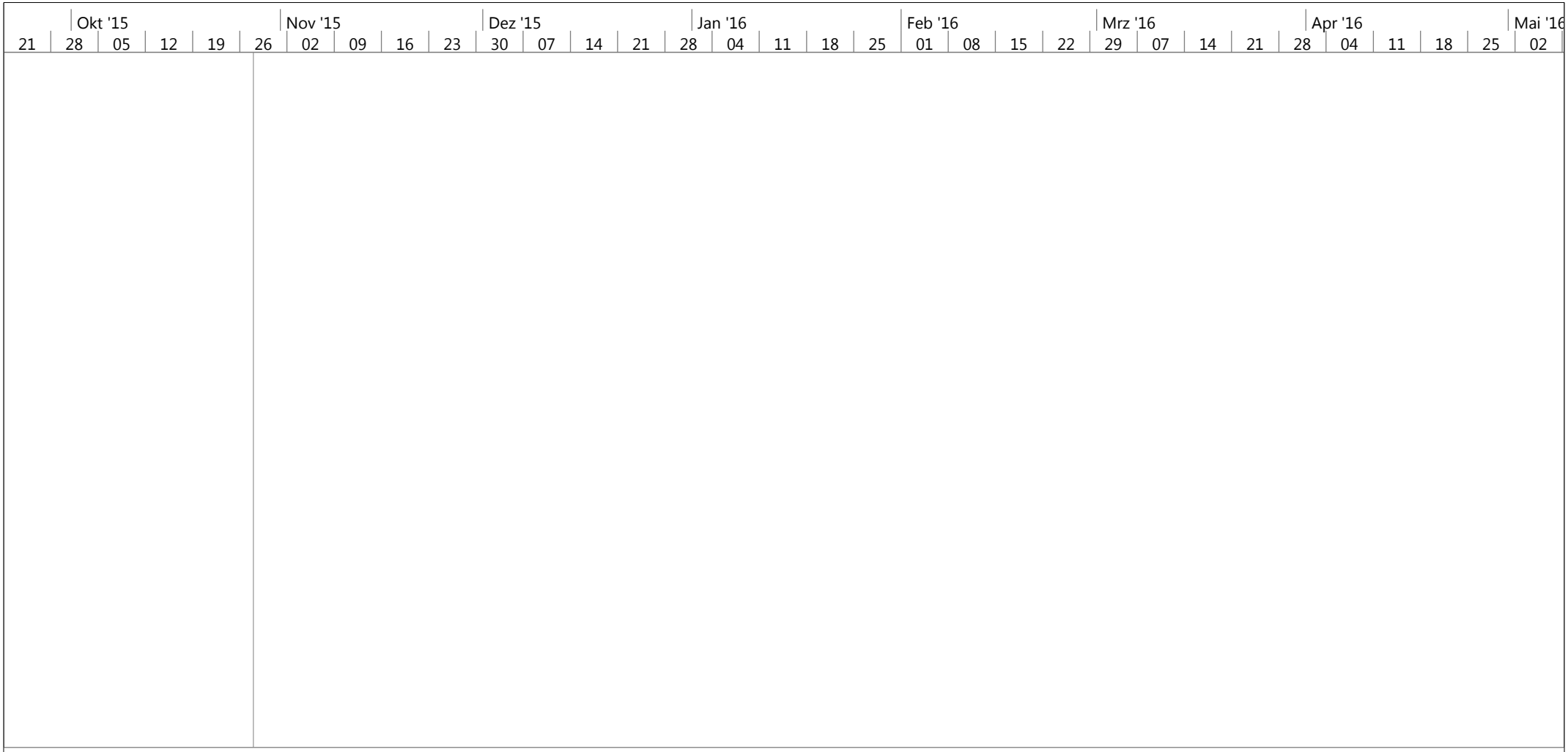
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



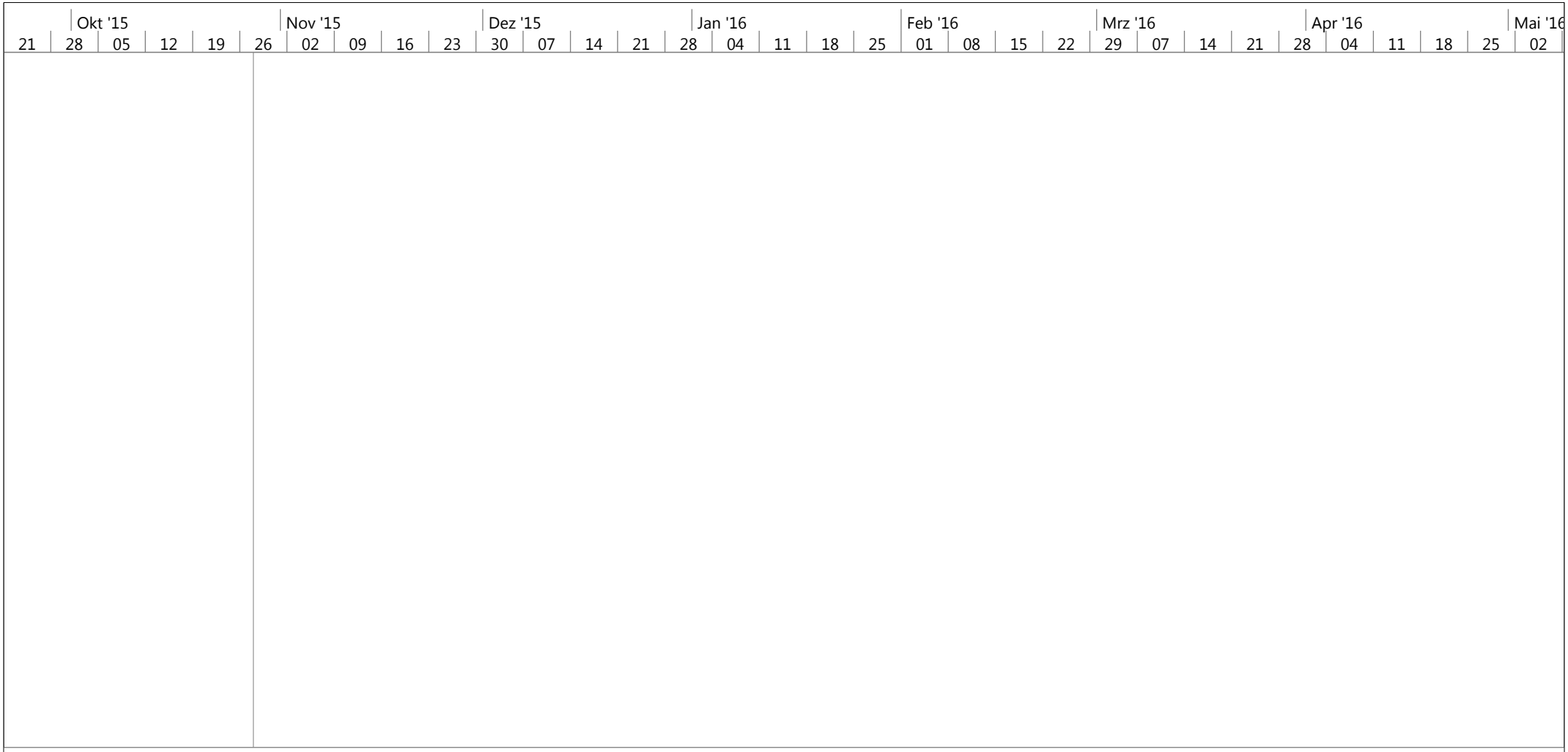
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

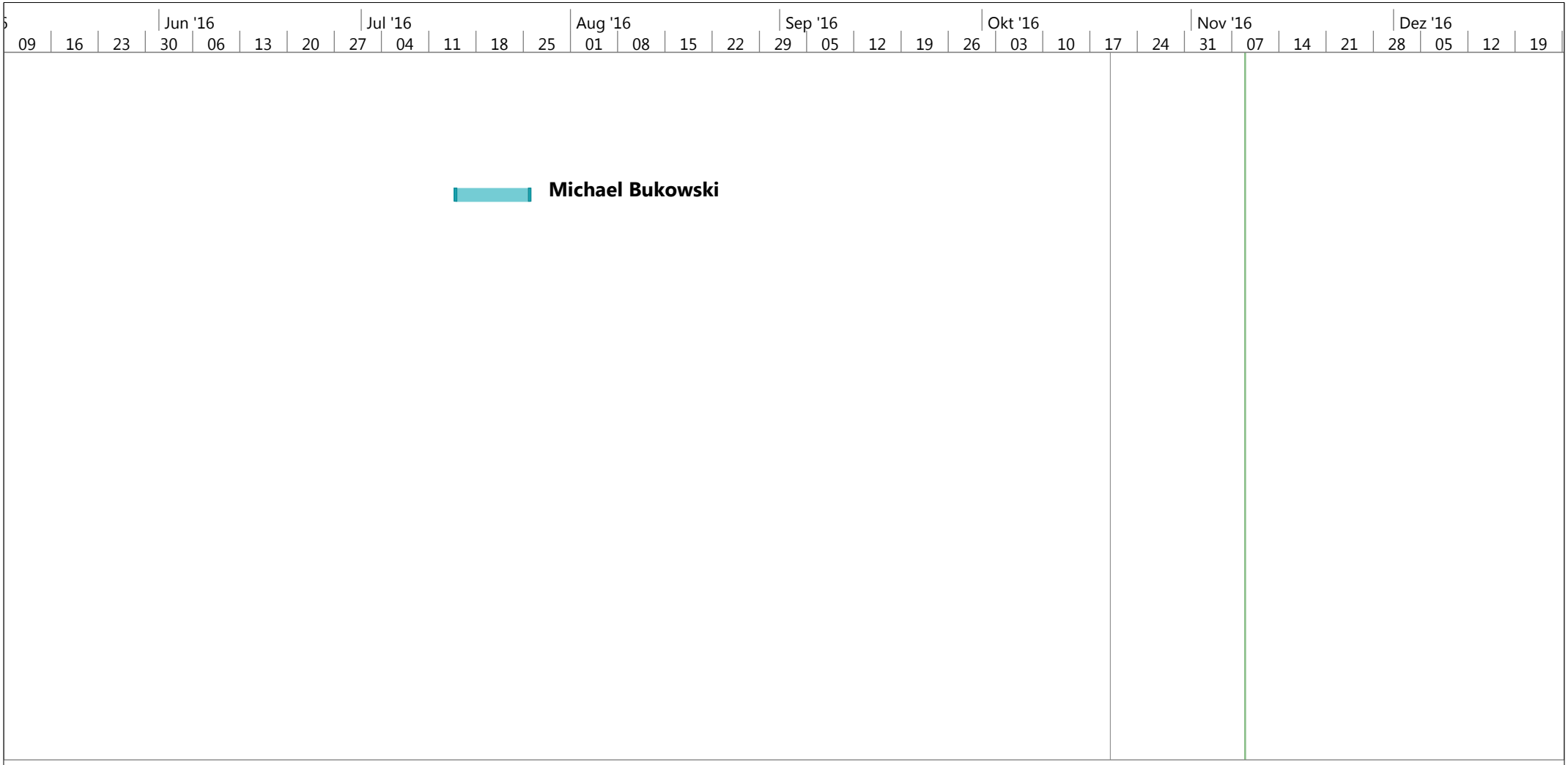


Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

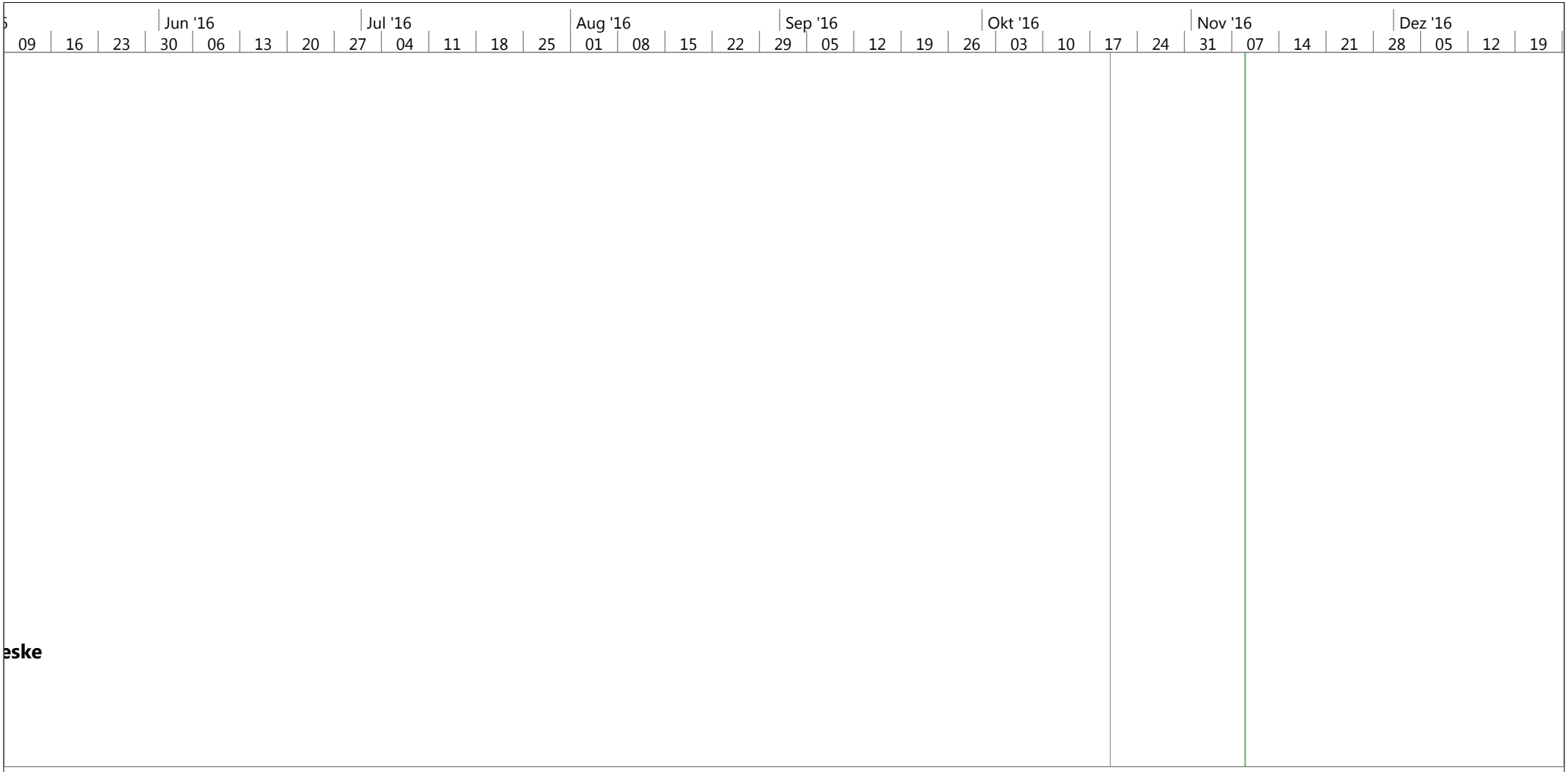
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



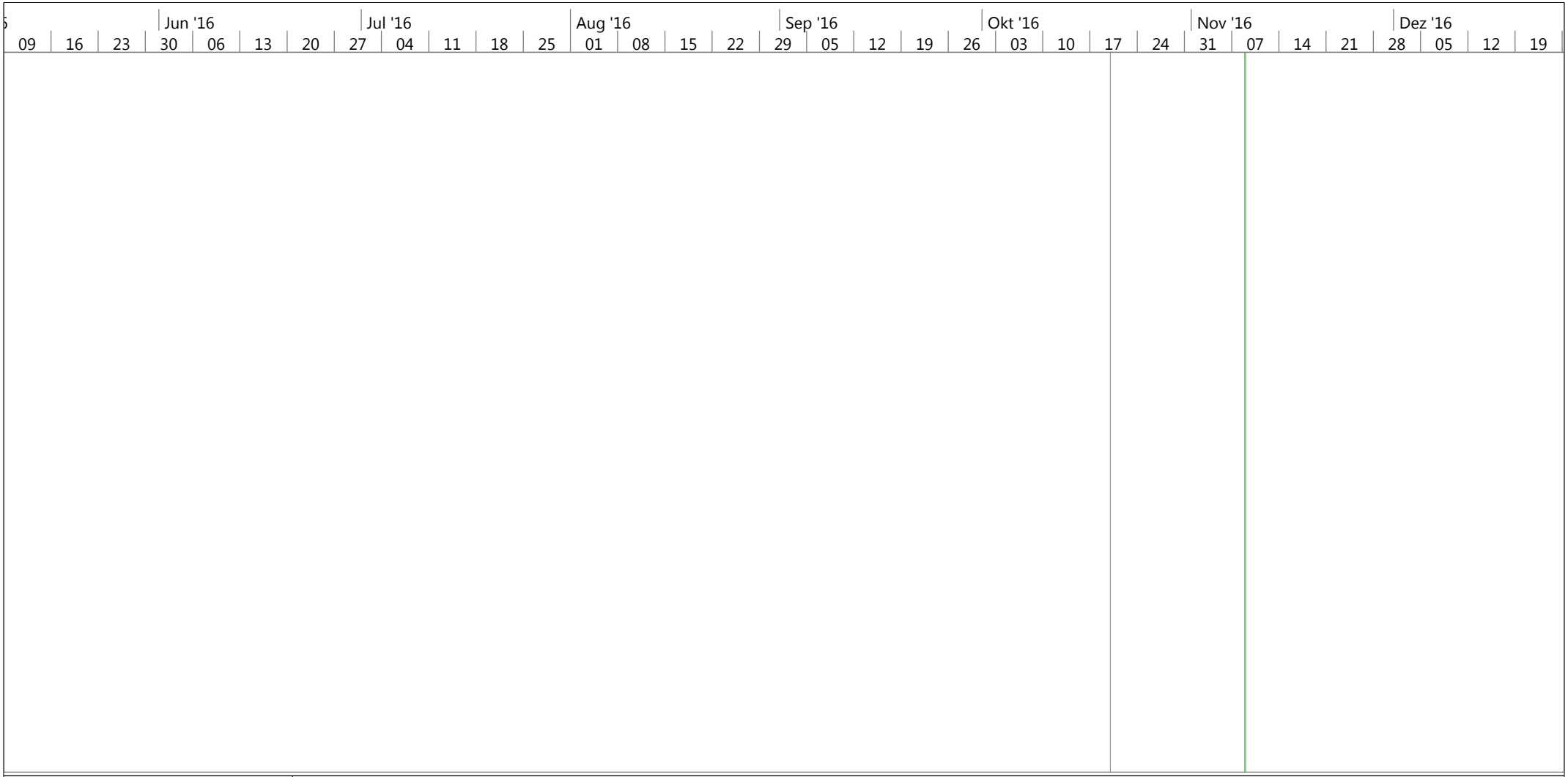
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

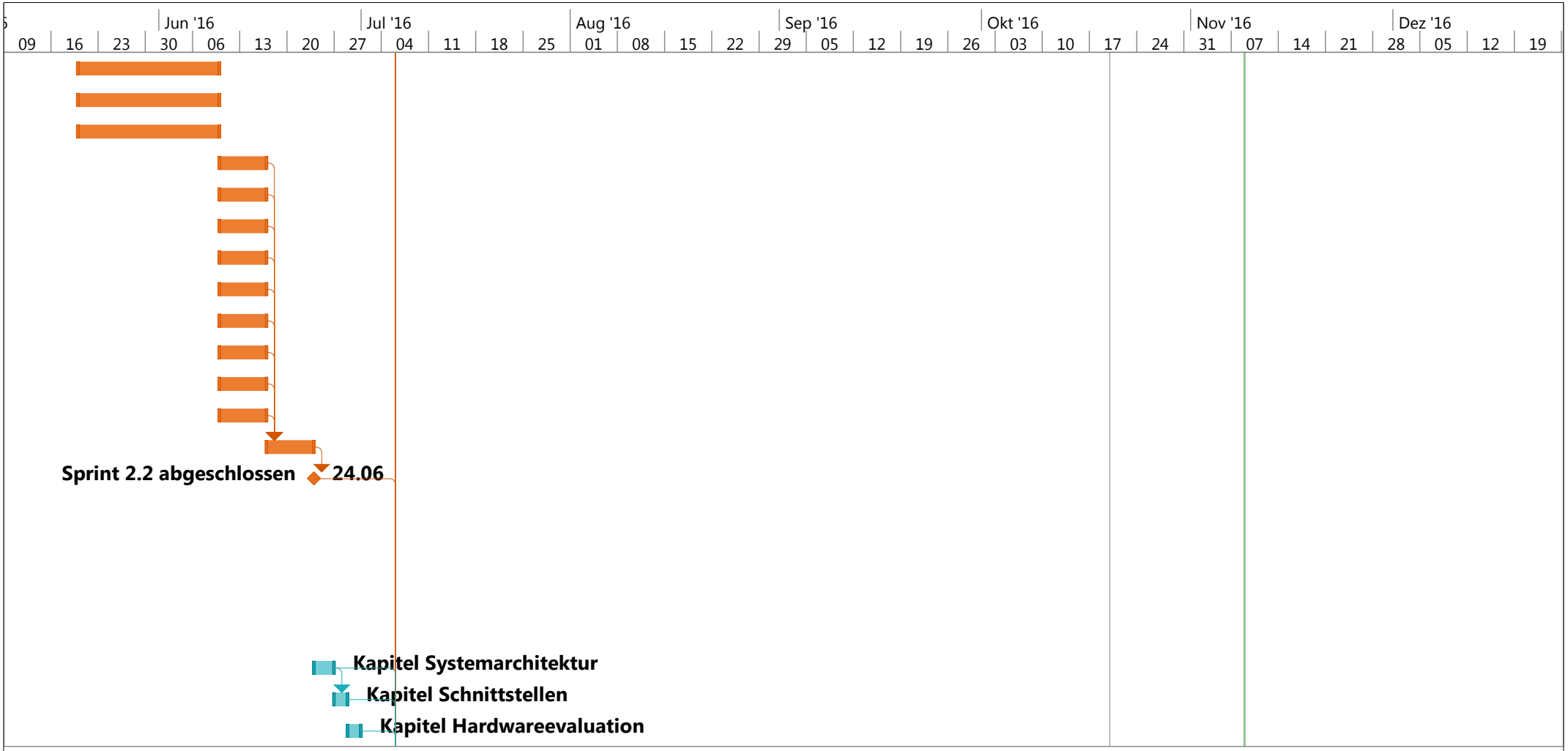
Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

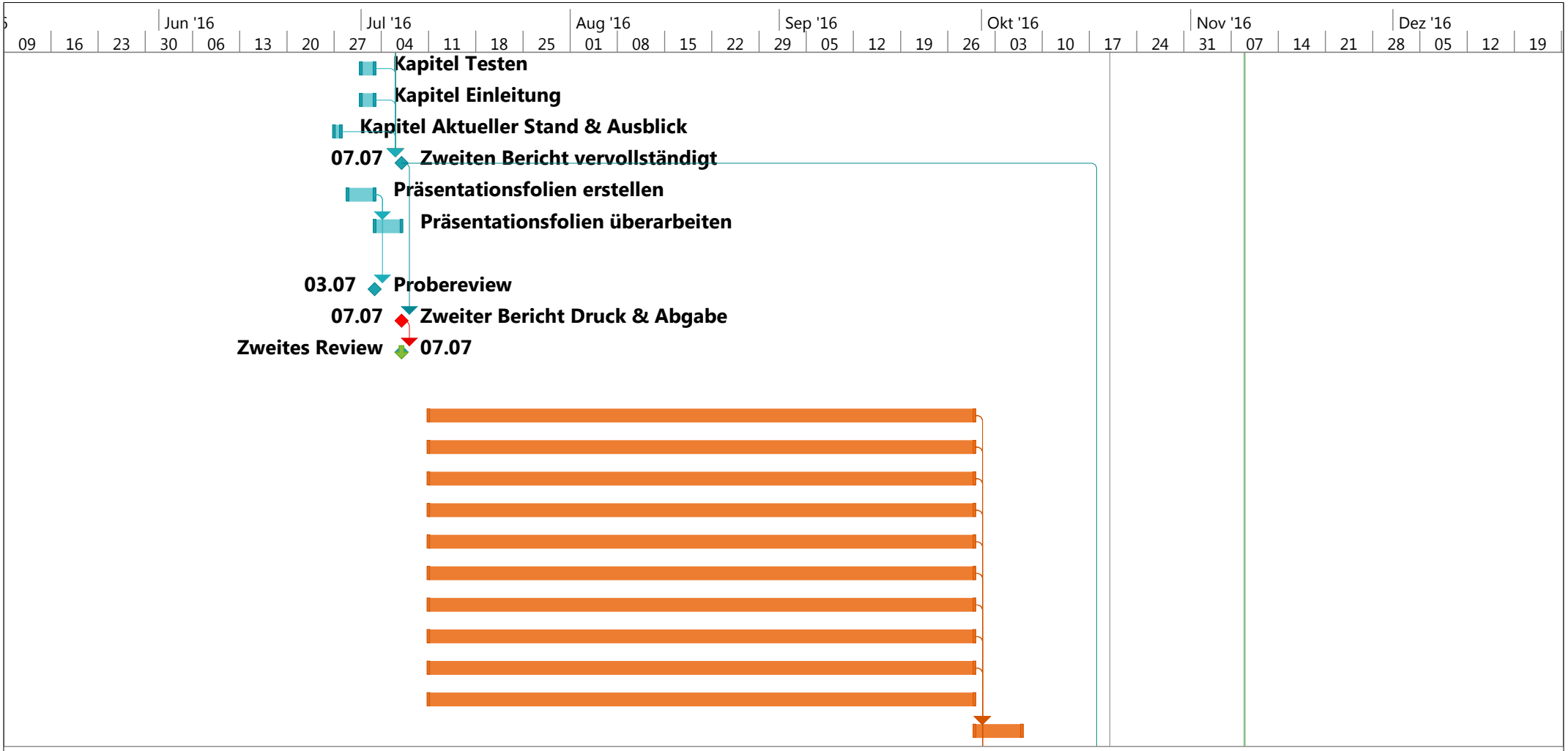
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang		Inaktiver Sammelvorgang		Externe Vorgänge	
Unterbrechung		Manueller Vorgang		Externer Meilenstein	
Meilenstein		Nur Dauer		Stichtag	
Sammelvorgang		Manueller Sammelrollup		In Arbeit	
Projektsammelvorgang		Manueller Sammelvorgang		Manueller Fortschritt	
Inaktiver Vorgang		Nur Anfang			
Inaktiver Meilenstein		Nur Ende			



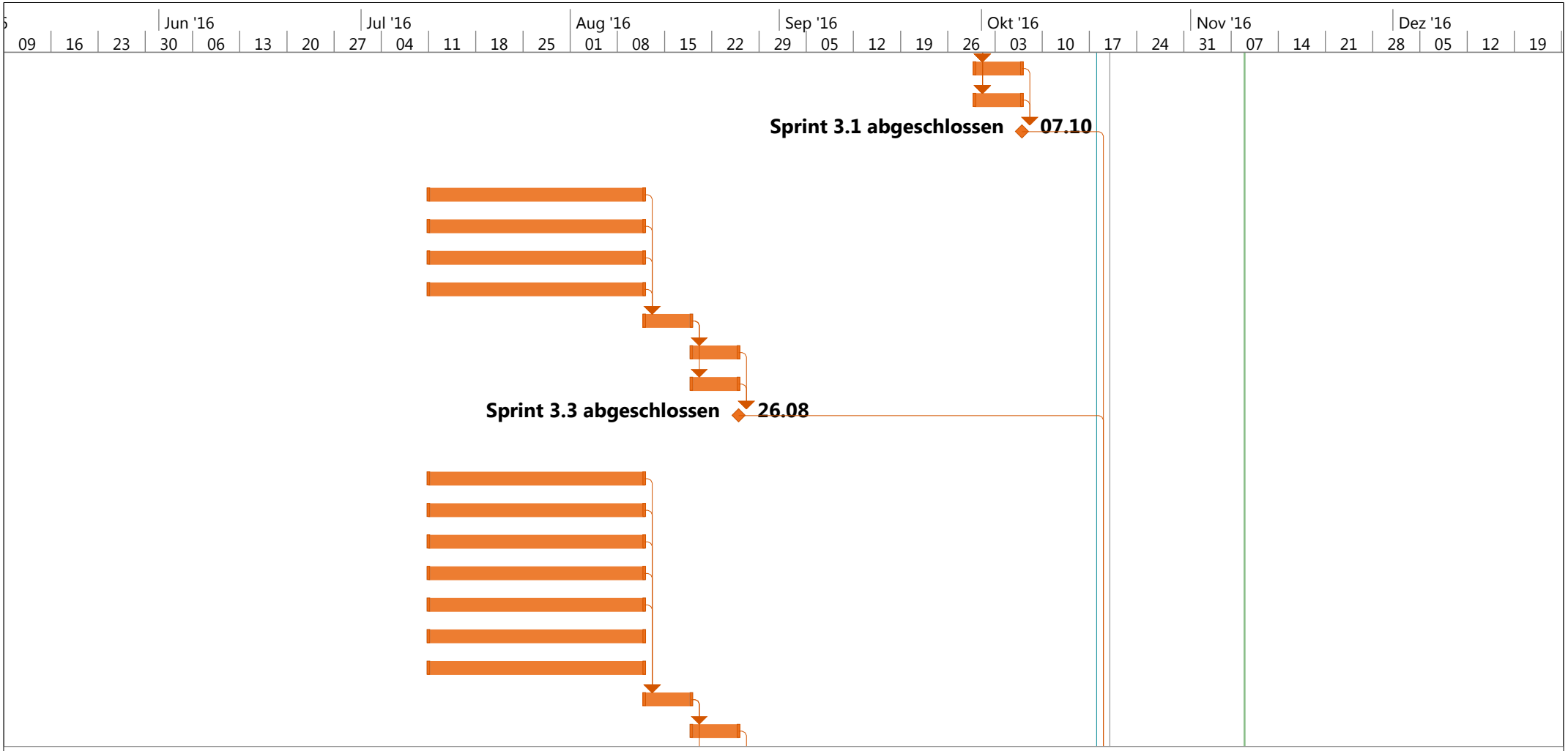
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang		Inaktiver Sammelvorgang		Externe Vorgänge	
Unterbrechung		Manueller Vorgang		Externer Meilenstein	
Meilenstein		Nur Dauer		Stichtag	
Sammelvorgang		Manueller Sammelrollup		In Arbeit	
Projektsammelvorgang		Manueller Sammelvorgang		Manueller Fortschritt	
Inaktiver Vorgang		Nur Anfang			
Inaktiver Meilenstein		Nur Ende			



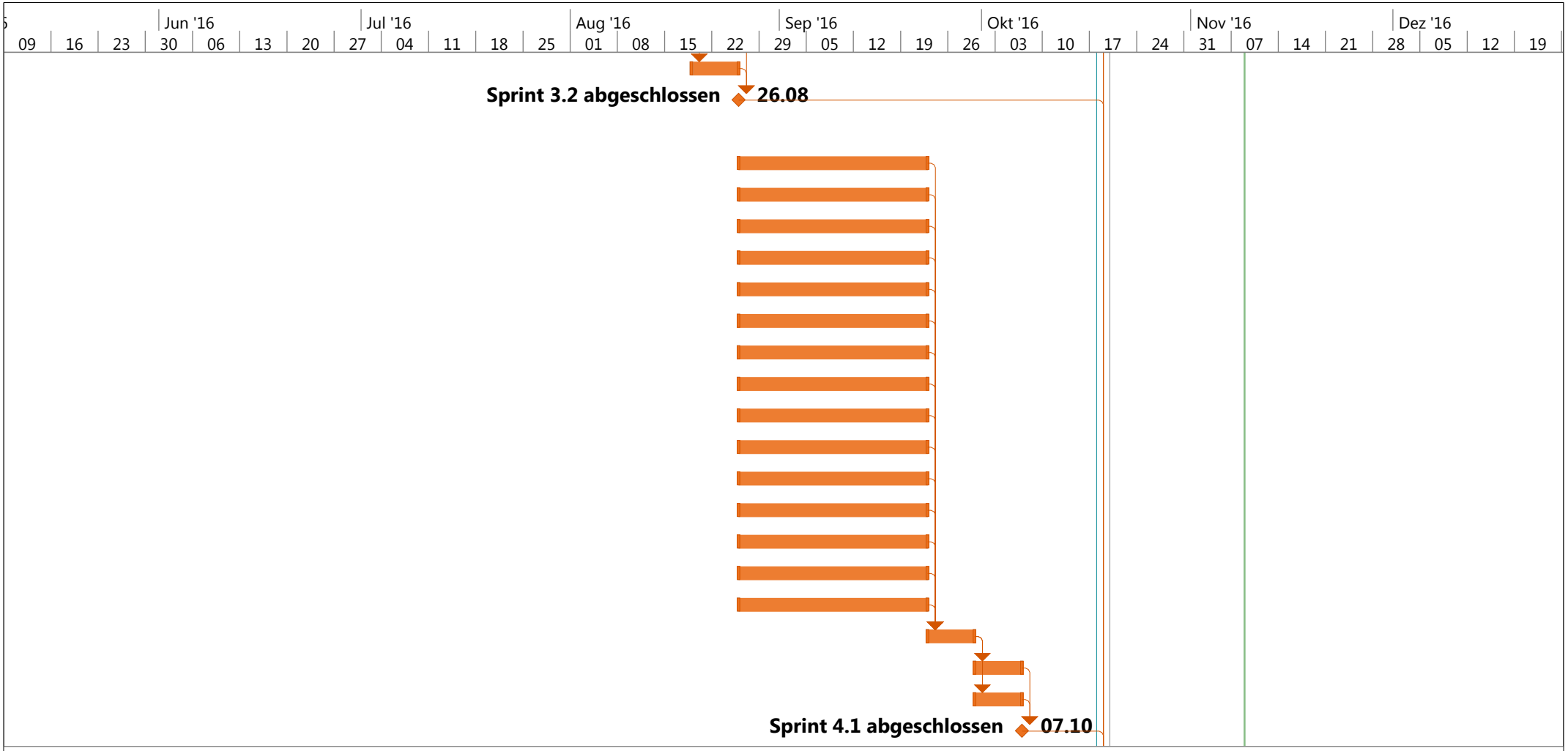
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



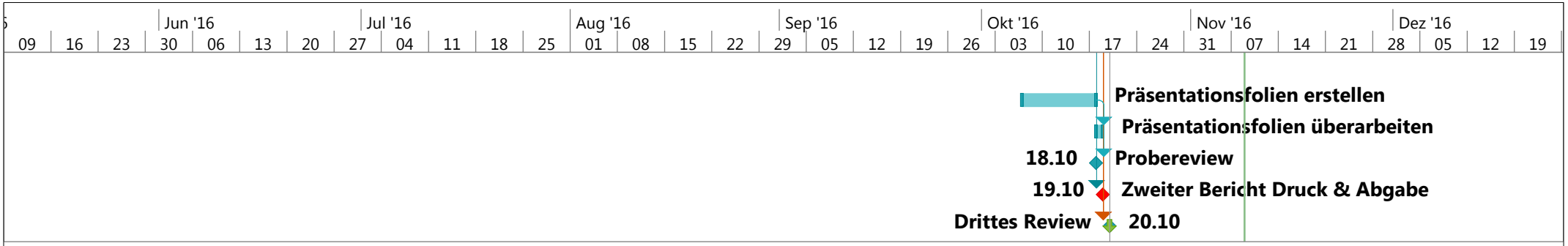
Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	



Projekt: Projektplan_RCCars_3.3
Datum: Mit 09.11.16

Vorgang	Inaktiver Sammelvorgang	Externe Vorgänge
Unterbrechung	Manueller Vorgang	Externer Meilenstein
Meilenstein	Nur Dauer	Stichtag
Sammelvorgang	Manueller Sammelrollup	In Arbeit
Projektsammelvorgang	Manueller Sammelvorgang	Manueller Fortschritt
Inaktiver Vorgang	Nur Anfang	
Inaktiver Meilenstein	Nur Ende	

14.3 Betriebs- und Wartungshandbuch

14.3.1 Allgemeine Hinweise

Aktuell werden sowohl die Positionsbestimmung, als auch die Control-Unit und die Systemsteuerungssoftware auf dem gleichen Rechner ausgeführt. Es muss beachtet werden, dass sowohl die IR-Beleuchtung, als auch das Netzteil der USB-Verlängerung zur Kamera nur eingesteckt sind, wenn der Rechner eingeschaltet ist.
Folgende Schritte sind zur Inbetriebnahme des Systems zu erledigen:

- Netzstecker der IR-Beleuchtung anschließen
- Netzstecker der USB-Verlängerung zur Kamera anschließen
- Rechner starten/hochfahren
- Systemsteuerungssoftware starten (siehe Abschnitt 14.3.4)
- Positionsbestimmung starten (siehe Abschnitt 14.3.2)
- Control-Unit starten (siehe Abschnitt 14.3.3)

Folgende Schritte sind zum Abschalten des Systems zu erledigen:

- Systemsteuerungssoftware beenden
- Control-Unit beenden (wenn nicht automatisch geschehen)
- Positionsbestimmung mit der Taste ’q’ beenden (wenn nicht automatisch geschehen)
- Wenn genutzt, dass hterm schließen
- Die USB Kabel der CarControl abziehen (Reihenfolge muss nicht beachtet werden)
- Rechner herunterfahren
- Netzstecker USB-Verlängerung zur Kamera abziehen
- Netzstecker der IR-Beleuchtung abziehen

14.3.2 Positionsbestimmung

- Direkte Sonnenlichteinstrahlung auf die Rennstrecke verhindern.
- Position der Rennstreckenmarker überprüfen und ggf. korrigieren.
- ggf. Kamera am Rechner bzw. an der USB-Verlängerung anschließen (blaues USB-Kabel)
- Verbindung zur Kamera prüfen. Lämpchen auf der Kamera sollte grün leuchten.
- Fahrzeug einschalten und in Fahrtrichtung auf der Start-/Ziellinie aufstellen.
- Positionsbestimmungssoftware starten
  - Datei 'PositionDetection' auf der Konsole ausführen
  - Wichtig: Die Konfigurationsdatei "positionsbestimmung.config" muss sich im gleichen Verzeichnis befinden!

14.3.2.1 Kalibrierung

- Mit dem Flycapture Programm die Framerate runterstellen, damit die Bilder besser belichtet werden.
- config.xml im CameraCalibration Debug Ordner öffnen und anpassen wieviele Bilder man aufnehmen möchte. Für eine gute Kalibrierung müssen mindestens 20 Bilder aufgenommen werden. In der selben Datei kann auch die Anzahl der Schachkacheln und die Größe dieser angepasst werden, falls ein anderes oder größeres Muster verwendet wird.
- Die Anzahl der Bilder, die in der Configdatei stehen, momentan 46, mit dem Flycapture2 Programm aufnehmen. Dabei muss das Schachbrettmuster in verschiedenen Positionen und Winkeln aufgenommen werden.
- Im Debug Ordner befindet sich die Datei images.xml. Dort müssen die Bilder alle eingetragen werden.
- CameraCalibration Programm starten und warten.
- Am Ende kann die Kalibrierung beurteilt werden. Es sollten oben, unten, links und rechts im Bild schwarze Flächen zu sehen sein. Außerdem sollte der average Reprojection Error so nah wie möglich an 0 liegen.

- In der output.xml Datei stehen dann die Distortionsparameter und die Kameramatrix. Diese Parameter in die positionsbestimmung.config übernehmen.
- Den Rest übernimmt die Positionsbestimmung selbst.

14.3.2.2 Konfiguration

Die Positionsbestimmung kann unterschiedlich konfiguriert werden.

- Möchte man den Filter ausprobieren und anpassen, so muss setupFilter auf 1 gesetzt werden. Es können dann die Schieberegler verschoben werden um den Filter anzupassen. Wichtig ist, dass man die geänderten Werte per Hand in die config File eintragen muss. Die Filtereinstellungen mit 'k' bestätigen. Im Anschluss kann man sich den Videostream anschauen. Ebenfalls wichtig: Ist setupFilter gesetzt, so wird die Systemsteuerungssoftware immer ignoriert und es wird immer ein Bild ausgegeben. Somit können die Realzeitanfordungen auch nicht erfüllt werden. Das Programm muss mit 'q' beendet werden, da sonst die Verbindung zur Kamera nicht richtig getrennt wird und somit beim nächsten Starten unbrauchbar ist.
- Ist ignoreSSS auf 0 muss man die Positionsbestimmung einfach nur starten. Den Rest übernimmt die SSS. Andernfalls Programm sicher mit 'q' beenden!
- Ist showVideoStream auf 1, kann die Realzeitanforderung nicht eingehalten werden. Die ControlUnit wird permanent Fehler ausgeben, da die Fahrzeugdaten zu alt sind.

14.3.3 Control-Unit

CarControl

- Die drei Mini-USB-Buchsen mittels passender USB-Kabel mit dem Rechner verbinden. Gegebenenfalls einen USB-Hub nutzen.
  - Während des Anschließens unter \dev prüfen, welche Schnittstelle welchen Namen erhält.
- Die Fernsteuerung einschalten, damit die Verbindung zwischen dieser und dem Fahrzeug hergestellt wird.
- Bei Bedarf können Debug-Ausgaben über das Tool "hterm" ausgelesen und angepasst werden
  - Hterm öffnen

  - Port der Debug-Schnittstelle auswählen (/dev/tty/USB0 oder /dev/tty/USB1)
  - Connect-Button (oben links) drücken
  - ”Newline at” sowohl für empfangene, als auch gesendete Daten auf ”CR+LF” stellen
  - Für mögliche Einstellungen der Debug-Ausgaben bitte in Kapitelabschnitt 14.5 sehen

- Die Modi der CarControl werden mit der blauen Taste des Entwicklungsboards gewechselt (siehe Kapitelabschnitt 10.3.2.2)
- Das Entwicklungsboard wird durch Drücken der schwarzen Taste auf dem Entwicklungsboard neu gestartet (HardReset). Durch drücken des schwarzen Tasters auf der CarControl wird ein SoftReset durchgeführt.
- Durch drücken des roten Tasters auf der CarControl wird die CarControl in den sicheren Fehlerzustand überführt.

Wrapper Code

- Prüfen, welcher Name der Schnittstelle der CarControl zugeteilt wurde.
- Software der Control-Unit starten
  - Datei ”ControlUnit” auf der Konsole ausführen
  - Wichtig: Die Konfigurationsdatei ”DATEINAME” muss sich im gleichen Verzeichnis befinden! Die Parameter der Konfigurationsdatei können - wie in Tabelle 14.3 in Kapitelabschnitt 14.5 beschrieben - angepasst werden. Hierzu gehört auch die Angabe der Schnittstelle der CarControl und der Dateiname der Rennstreckendatei.

14.3.4 Systemsteuerungssoftware

- Die Systemsteuerungssoftware über die Konsole starten.
  - Befehl zum Starten: ”java -jar SSSvX.Y.jar”
  - Hinweis: Die Bibliothek `libNetworkMessengerSharedLib.so` für die Netzwerkkommunikation muss sich im selben Verzeichnis wie die ausführbare Jar-Datei der Systemsteuerungssoftware befinden.
- Falls nötig können die Systemparameter über den ”Settings”-Tab der Systemsteuerungssoftware angepasst werden.

- Erwartet Anzahl der am Rennen teilnehmenden Fahrzeuge eingeben.
- Manuelle Inspektion durchführen:
  - Einhaltung aller Rahmenbedingungen gemäß Kapitelabschnitt 5.1 kontrollieren.
  - Positionsbestimmungssoftware starten (siehe auch Abschnitt 14.3.2)
  - Regelungssoftware starten und CarControl auf vollständig autonomes Fahren stellen (siehe auch Abschnitt 14.3.3)
- Die manuelle Inspektion über den ”System Control”-Tab der Systemsteuerungssoftware mit dem Button ”Confirm Inspection” bestätigen.
  - Systemparameter können bis zur nächsten Inspektion nicht mehr verändert werden.
  - Falls es bei der letzten Ausführung der Systemsteuerungssoftware zu einem Fehler kam, muss die Datei ErrorStore.txt bzw. deren Inhalt gelöscht, werden bevor die Inspektion erfolgreich bestätigt werden kann. Der Dateipfad wird ggf. auf der Benutzeroberfläche der Systemsteuerungssoftware ausgegeben.
- Die Initialisierung über den Button ”Start Initialization” starten und darauf warten, dass die Positionsbestimmung und alle Control-Units melden, dass sie bereit sind. Eventuell müssen im Rahmen der Initialisierung noch Einstellungen der Positionsbestimmungssoftware vorgenommen werden (siehe Abschnitt 14.3.2). Der Standby-Zustand wird nach erfolgreicher Initialisierung automatisch erreicht.
- Im Standby-Zustand über den Button ”Start Race” den Rennbetrieb starten.
- Wenn das Rennen beendet werden soll kann das System während des Rennbetriebs über den Button ”Stop Race” wieder in Standby-Zustand versetzt werden.
- Der Systembetrieb kann unabhängig vom Systemzustand jederzeit beendet werden indem die Systemsteuerungssoftware geschlossen wird. Beim Beenden kommt es abhängig von den gewählten Systemparametern (siehe ”Keep alife frequence”) zu einer kurzen Verzögerung, da die Systemsteuerungssoftware den anderen Systemkomponenten wiederholt mitteilt, dass der Systembetrieb beendet wird.
- Im Standby-Zustand kann das System über den Button ”Perform Inspection” erneut in den Inspektionszustand versetzt werden. Dort können auch wieder die Systemparameter und die Anzahl der am Rennen teilnehmenden Fahrzeuge verändert werden.

- Der Button ”EMERGENCY STOP” muss unverzüglich betätigt werde, falls eine Rahmenbedingung verletzt wird oder wenn das Fahrzeug nicht sicher autonom gesteuert wird. Dies ist unabhängig vom Systemzustand jederzeit möglich.
- Systemnachrichten können im unteren Drittel der Benutzeroberfläche im ”System Control Log”-Tab abgelesen werden und werden, falls der Debugging-Modus aktiv ist, in einer Logdatei im Verzeichnis ”Log” gespeichert.
- Fehlermeldungen können im ”Error Log”-Tab abgelesen werden und werden in einer Logdatei im Verzeichnis ”Log” gespeichert.

14.4 Konfigurationsparameter der Positionsbestimmung

Auflistung sämtlicher Konfigurationsparameter der Positionsbestimmung

Variable	Beschreibung	Datentyp	Wertebereich
showDebugOutputs	Wird auf 1 gesetzt, falls Debugausgaben in der Konsole angezeigt werden sollen.	int	0/1
showVideoStream	Wird auf 1 gesetzt, falls der Videostream in einem Fenster angezeigt werden soll.	int	0/1
printPoseToDisk	Wird auf 1 gesetzt, falls eine bestimmte Anzahl an Fahrzeugposen auf der Festplatte gespeichert werden soll.	int	0/1
poseCount	Gibt die Anzahl der Posen an, die auf der Festplatte gespeichert werden sollen.	int	0 . . . 1000
ignoreSSS	Wird auf 1 gesetzt, falls die Positionsbestimmungssoftware nicht auf Befehle warten soll.	int	0/1
setupFilter	Wird auf 1 gesetzt, falls Anpassungen an den Filtereinstellungen vorgenommen werden sollen.	int	0/1
MAX_THRESHOLD_TYPE	Obere Grenze der Schwellwert Typ Auswahl.	int	4
MAX_THRESHOLD_VALUE	Obere Grenze des ausgewählten Schwellenwert-Filters.	int	255
MAX_BLUR_K_VALUE	Obere Grenze des Blur-Verfahrens.	int	19
DEFAULT_WINDOW_WIDTH	Standardbreite der erzeugten Fenster.	int	1024

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter

Variable	Beschreibung	Datentyp	Wertebereich
DEFAULT_WINDOW_HEIGHT	Standardhöhe der erzeugten Fenster.	int	768
thresholdType	Standard Schwellenwert Verfahren nach dem Initialisieren des FahrzeugSchiebe-reglerfensters.	int	0 . . . MAX_THRESHOLD_TYPE
lowerBound1	Standard unterer Schwellenwert Wert nach dem Initalisieren des Fahrzeug-Schiebereglerfensters	int	0 . . . MAX_THRESHOLD_VALUE
lowerBound2	Standard oberer Schwellenwert Wert nach dem Initalisieren des Fahrzeug-Schiebereglerfensters	int	0 . . . MAX_THRESHOLD_VALUE
blurValue	Standard Blur Wert nach dem Initalisieren des Fahrzeug-Schiebereglerfensters	int	0 . . . MAX_BLUR_K_VALUE
lowerBound2	Unterer Wert des inRange Filter nach dem Initalisieren des Fahrzeug Schieberegler-fensters	int	0 . . . MAX_THRESHOLD_VALUE
upperBound2	Oberer Wert des inRange Filter nach dem Initalisieren des Fahrzeug Schieberegler-fensters	int	0 . . . MAX_THRESHOLD_VALUE
fx	Fokuslänge in x Richtung in Pixeln.	double	0 . . . 1280
cx	Bildmitte x Koordinate.	double	0 . . . 1280
fy	Fokuslänge in y Richtung in Pixeln.	double	0 . . . 1024

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter			
Variable	**Beschreibung**	**Datentyp**	**Wertebereich**
cy	Bildmitte y Koordinate.	double	0 … 1024
k1	Radiale Verzerrung Parameter 1.	double	-10 … 10
k2	Radiale Verzerrung Parameter 2.	double	-10 … 10
p1	Tangiale Verzerrung Parameter 1.	double	-10 … 10
p2	Tangiale Verzerrung Parameter 2.	double	-10 … 10
k3	Radiale Verzerrung Parameter 3.	double	-10 … 10

Tabelle 14.2: Auflistung sämtlicher Bildverarbeitungsparameter

14.5 Konfigurationsparameter der ControlUnit

Auflistung sämtlicher Konfigurationsparameter der ControlUnit

Variable	Beschreibung	Datentyp	Wertebereich
MAX_BLUR_K_VALUE	Obere Grenze des Blur-Verfahrens.	int	19
carID	ID des Fahrzeuges, welches gesteuert werden soll.	int	0 ... 8
trimmingThrottle	Wert für die Trimmung der Längsführung.	int	0... 4096
trimmingSteering	Wert für die Trimmung der Querführung.	int	0... 4096
racetrackFilename	Dateiname der Rennstreckendatei	char	A-Z, a-z
RoadLength	Rundenlänge in Metern.	float	6.427
TargetSpeed	Zielgeschwindigkeit auf geraden Streckensegmenten	float	$\geq$0.0
TargetSpeedCurve	Zielgeschwindigkeit in Kurven	float	$\geq$0.0
TargetSpeedWarmUp	Zielgeschwindigkeit auf geraden Streckensegmenten im WarmUp	float	$\geq$0.0
TargetSpeedCurveWarmUp	Zielgeschwindigkeit in Kurven im WarmUp	float	$\geq$0.0
RadiusCircle	Initialer Radius des Kreis, der um das Fahrzeug gelegt wird zur Zieltrajektorienfeststellung	int	$\geq$0
RadiusCircleMin	Minimaler Radius des Kreis, der um das Fahrzeug gelegt wird zur Zieltrajektorienfeststellung	int	$\geq$0

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter			
Variable	**Beschreibung**	**Datentyp**	**Wertebereich**
RadiusCircleMax	Maximaler Radius des Kreis, der um das Fahrzeug gelegt wird zur Zieltrajektorienfeststellung	int	$\geq$0
WarmUpRounds	Anzahl an Aufwärmrunden, welche das Fahrzeug zusätzlich zu der fest implementierten Aufwärmrunde fährt.	int	$\geq$0
serialport	Name der seriellen Schnittstelle zur CarControl	char	Bsp. /dev/ttyUSB0
InitializationMode	Modus, in welchen die CarControl sich im Systemzustand "Initialisierung" befinden soll.	int	0 ... 4
StandbyMode	Modus, in welchen die CarControl sich im Systemzustand "Standby" befinden soll.	int	0 ... 4
RaceMode	Modus, in welchen die CarControl sich im Systemzustand "Rennbetrieb" befinden soll.	int	0 0 ... 4
connectionCheckInterval	Intervall in welchem die Verbindung zur CarControl geprüft werden soll (in ms). Bei Wert *-1* wird nicht geprüft	float	$\geq$0.0

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter			
Variable	**Beschreibung**	**Datentyp**	**Wertebereich**
regulationValueSendingInterval	Intervall in welchem Regelungswerte an die CarControl gesendet werden sollen (in ms).	float	$\geq$0.0
receiveFromLocalhostOnly	Schalter zum Einstellen, ob Netzwerkpakete nur vom Localhost empfangen werden sollen.	int	0 / 1
sendToLocalhostOnly	Schalter zum Einstellen, ob Netzwerkpakete nur an den Localhost gesendet werden.	int	0 /1
broadcastPort	Port, welches für den Broadcast via UDP genutzt werden soll.	int	0 . . . 65535
BarrierCollisionDetection	Schalter für das ein- und ausschalten der Bandenkollisionserkennung	int	0 / 1
vehicledataWatchdog	Schalter für das ein- und ausschalten der Plausibilitätsprüfung der Fahrzeugdaten	int	0 / 1
vehicledataAllowedDelay	Erlaubter Zeitabstand zwischen zwei Fahrzeugdten (in ms).	int	$\geq$0
vehicledataAllowedCoordDiff	Erlaubte Differenz auf der X- und Y-Achse zwischen zwei Fahrzeugdaten (in cm) im Systemzustand ”Rennbetrieb”.	float	$\geq$0.0

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter

Variable	Beschreibung	Datentyp	Wertebereich
vehicledataAllowedCoordDiffInStandby	Erlaubte Differenz auf der X- und Y-Achse zwischen zwei Fahrzeugdaten (in cm) im Systemzustand ”Standby”.	float	$\geq$0.0
vehicledataAllowedAlignmentDiff	Erlaubte Differenz des Ausrichtungswinkels zwischen zwei Fahrzeugdaten (in cm) im Systemzustand ”Rennbetrieb”.	float	$\geq$0.0
vehicleDataAllowedMissingData	Erlaubte Anzahl an Fahrzeugdaten hintereinander, in welchen das Fahrzeug nicht detektiert wurde.	int	$\geq$0
controldataWatchdog	Schalter für das ein- und ausschalten der Plausibilitätsprüfung der Kontrolldaten	int	0 / 1
controldataAllowedDelay	Erlaubter Zeitabstand zwischen zwei Kontrolldaten (in ms).	int	$\geq$0
faultdataWatchdog	Schalter für das ein- und ausschalten der Plausibilitätsprüfung der Fehlerdaten	int	0 / 1
ignoreSSS	Ignorieren der Systemsteuerungssoftware bzw. Ausführung der Control-Unit ohne diese.	int	0 / 1
ignoreCarControl	Ignorieren der seriellen Schnittstelle zur CarControl bzw. nicht-initialisierung dieser).	int	0 / 1

Fortsetzung Auflistung sämtlicher Bildverarbeitungsparameter			
Variable	**Beschreibung**	**Datentyp**	**Wertebereich**
consoleOutput	Schalter die Konsolenausgaben. Bei Wert 0 keine Konsolenausgaben. Bei Wert 1 textuelle Konsolenausgaben. Bei Wert 2 InfoBoard.	int	0 . . . 2
networkReceiver	Schalter für das ein- und ausschalten des Netzwerkempfangs (Thread).	int	0 / 1
watchdog	Schalter für das ein- und ausschalten des Watchdogs (Thread).	int	0 / 1
scade	Schalter für das ein- und ausschalten des SCADE Modells (Thread).	int	0 / 1
serialCommunication	Schalter für das ein- und ausschalten der seriellen Kommunikation (Thread).	int	0 / 1
timeMeasurement	Schalter für das ein- und ausschalten der Zeitmessung (Thread).	int	0 / 1
logging	Schalter für das ein- und ausschalten für das Logging von Regelungswerten des SCADE Modells.	int	0 / 1
DebugIn1		float	≥ 0
DebugIn2		float	≥ 0
DebugIn3		float	≥ 0

Tabelle 14.3: Auflistung sämtlicher Bildverarbeitungsparameter

14.6 Konfigurationsparameter der Systemsteuerungssoftware

Auflistung sämtlicher Konfigurationsparameter der Systemsteuerungssoftware

Variable	Beschreibung	Datentyp	Wertebereich
expectedNumberOfCars	Speichert die vom Benutzer vorgegebene Anzahl vom Fahrzeugen, welchen am Rennen teilnehmen sollen.	int	1… 16
carsMismatchTolerance	Fehlertoleranzkonstante, welche festlegt wie hoch eine vorübergehende Abweichung der aktuell erkannten aktiven Fahrzeugen, im Vergleich zum anfangs festgelegten Satz an aktiven Fahrzeugen, sein darf.	int	0…
carCoordTolerance	Fehlertoleranzkonstante, welche festlegt wie sehr die x- / y-Koordinaten (in cm) der Fahrzeuge im Standby-Zustand variieren dürfen.	double	0.0… 237.0
carAngleTolerance	Fehlertoleranzkonstante, welche festlegt wie sehr die Ausrichtung (in Grad) der Fahrzeuge im Standbymodus variieren darf.	double	0.0… 360.0
carStopDelayTolerance	Fehlertoleranzkonstante, welche festlegt wie lange (in Millisekunden) sich die Fahrzeuge noch bewegen dürfen nachdem der Rennmodus gestoppt wurde.	long	0.1…

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter

Variable	Beschreibung	Datentyp	Wertebereich
maxVehicleDataProcTime	Konstante, welche festlegt wie viel Zeit (in Sekunden) maximal vom der Erfassung der Fahrzeugposen durch die PB bis zur Ausgabe der Fahrzeugsteuerungsdaten an das Fahrzeug vergehen darf. Die benötigte Zeit wird in einem Timedata-Package von der CU des jeweiligen Fahrzeugs ausgegeben.	double	0.0…
keepAliveDelay	Konstante, welche festlegt nach wie viel Zeit (in Millisekunden) die SSS das zuletzt von dieser ausgegebene Netzwerkdatenpaket - als keep alive Signal - wiederholt ausgeben soll.	long	10…
maxInternalProcessingTime	Konstante, welche die interne Realzeitanforderung an die SSS festlegt. Es wird definiert wie viel Zeit (in Millisekunden) ein Ausführungsdurchgang (Auswertung von Netzwerkdatenpaketen und Benutzereingaben usw.) der SystemControl-Klasse benötigen darf.	long	1…

Fortsetzung Auflistung sämtlicher Bilderverarbeitungsparameter			
Variable	**Beschreibung**	**Datentyp**	**Wertebereich**
maxInternalProcTimeViolation	Fehlertoleranzkonstante, welche festlegt in welchem Umfang die interne Realzeitanforderung der SSS verletzt werden darf bis dies als Fehler angesehen werden muss. Verletzungen der Realzeitanforderung werden ihrem Ausmaß entsprechend skaliert und aufsummiert.	int	0...
useDebug	Speichert, ob das Debugging aktiv ist oder nicht.	boolean	0 / 1
logNetworkDataPackages	Speichert, ob das Logging von Netzwerkdatenpaketen aktiv ist oder nicht.	boolean	0 / 1

Tabelle 14.4: Auflistung sämtlicher Konfigurationsparameter der Systemsteuerungssoftware.

14.7 Bedienungsanleitung des DummyPackageGenerators

Das Programm `DummyPackageGenerator` (DPG) erzeugt bestimmte Netzwerkdatenpakete des `NetworkMessengers`, um die Funktionalität der Systemsteuerungssoftware oder der Control-Unit unabhängig von der Positionsbestimmung zu testen. Mit Hilfe der verfügbaren Testmethoden können bestimmte Anwendungsfälle und Zustandsübergänge gezielt simuliert werden. Die Auswahl der auszuführenden Testmethoden, der Anzahl der simulierten Fahrzeuge und die Taktung der Ausgabe der Netzwerkdatenpakete wird mit Hilfe folgender Eingabeparameter vorgenommen.

Es Gibt drei Eingabeparameter:

1. Eingabebefehl zur Auswahl der Testmethode
   - `init`
     Simulation einer korrekten Initialisierung des Systems. Die definierte Anzahl an Control-Units und die Positionsbestimmung senden ein Kontrolldatenpaket mit readySignal = 1.
   - `standby`
     Simulation eines fehlerfreien Standby-Zustands. Es werden Fahrzeugdatenpakete mit identischen Fahrzeugposen ausgegeben (keine Bewegung).
   - `race`
     Simulation eines fehlerfreien Rennbetriebs. Es werden Fahrzeugdatenpakete mit unterschiedlichen Fahrzeugposen ausgegeben (Fahrzeuge bewegen sich). Dieser Eingabebefehl kann auch verwendet werden, um einen fehlerhaften Standby-Zustand (mit unerlaubter Fahrzeugbewegung) zu simulieren. Achtung: Die Fahrzeugposen können im nicht befahrbaren Bereich liegen.
   - `fault`
     Simuliert die Ausgabe von Fehlerdatenpaketen. Der Befehl kann z. B. im Anschluss an `init`, `standby` oder `race` verwendet werden, um Fehler im jeweiligen Zustand zu simulieren.
   - `time`
     Simuliert die Ausgabe von Zeitdatenpaketen, welche Timestamps enthalten, die eine Einhaltung der Realzeitanforderung für den gesamten Verarbeitungsprozess (Bilderfassung bis Ausgabe von Steuerungsdaten $<= 50000\,\mu$s) darstellen.

- `timeF`
  Simuliert die Ausgabe von Zeitdatenpaketen, welche Timestamps enthalten, die eine Verletzung der Realzeitanforderung für den gesamten Verarbeitungsprozess (Bilderfassung bis Ausgabe von Steuerungsdaten $> 50000\,\mu$s) darstellen.
- `raceF`
  Simulation eines Rennbetriebs (mit sich verändernden Fahrzeugposen wie in `race`), wobei zwischendurch ein einzelnes Fehlerdatenpaket gesendet wird. Kann z. B. für Performanztests der Systemsteuerungssoftware verwendet werden.
- `delayTest`
  Testmethode zur gezielten Überprüfung einer möglichen Verzögerung in der Control-Unit. Ein Fahrzeug bewegt sich die Start-Zielgerade hinunter und erreicht die Kurve (bei Standardstreckenaufbau gemäße der ersten Iteration der Projektes *RCCARS*). In der Konsole wird eine auffällige Testausgabe ausgegeben, mit deren Hilfe die Verzögerung beim Einlenken des Fahrzeugs erfasst werden kann.

2. Eingabekommando mit dem die Anzahl der zu simulierenden Fahrzeuge [1 ... 16] festlegt wird
3. Verzögerung zwischen den Ausgaben der einzelnen Netzwerkdatenpakete in Mikrosekunden (μs):
   - 0: keine Verzögerung
   - > 0: Verzögerung um die angegebene Zahl an μs
   - < 0: nur eine einzige Ausführung der Testmethode

– Beispiel: ./DummyPackageGenerator delayTest 1 10000

Ende

Glossar

Car2X Kommunikation von Fahrzeugen mit anderen Systemkomponenten. Dazu gehören sowohl andere Fahrzeuge als auch Teile der Systems.

Code-Review Bei einem Code-Review wird eine entwickelte Codekomponente von einem Team oder einem Teammitglied begutachtet und entsprechend genehmigt oder kommentiert.

Control-Unit Steuerwerk, Steuereinheit, Kontrolleinheit.

cyclic redundancy check Zyklische Redundanzprüfung. Verfahren zur Bestimmung eines Prüfwertes von Daten.

DA Digital-Analog.

DA-Wandlerkarte Eigenständig entwickeltes Verbindungsglied zur Kommunikation zwischen Rechner und Fernsteuerung.

DAC Digital-Analog-Konverter.

dNaNo Hersteller der verwendeten RC-Cars.

Dreamspark Microsoft Dreamspark bietet Schülern und Studierenden die Möglichkeit eine große Auswahl an Microsoft Designer und Entwicklungstools kostenfrei zu erwerben. Die Anwendung darf ausschließlich im nicht kommerziellen Bereich stattfinden. [Dre]

Entwicklungsboard Platine für die Evaluierung eines Mikrocontrollers über dessen Schnittstellen.

FPS Bilder pro Sekunde (frames per second).

GPS Global Positioning System.

GUI Grafische Benutzeroberfläche.

HW Hardware.

ID Identität.

IR Infrarot.

Meilensteine Meilensteine sind fixe (Projekt-) Termine, welche nicht verschoben werden dürfen oder können.

Minimum Bounding Box Kleinst mögliches Rechteck mit dem sich ein beliebiges zweidimensionales Objekt beschreiben lässt.

Netzplantechnik Innerhalb der Netzplantechnik können bestimmte Aktionen miteinander verkettet werden.

OpenCV Freie Programmbibliothek für die Bildverarbeitung und für maschinelles Sehen.

OS Betriebssystem.

PID Proportional-Integral-Differential-Regler.

Platooning Ein Steuerungssystem welches ermöglicht, dass mehrere Fahrzeuge in einem geringen Abstand hintereinander fahren können.

Pose Bezeichnet die räumliche Lage eines Objekts aus der Kombination von Position und Orientierung.

Potentiometer Elektrisches Widerstandsbauelement, dessen Widerstandswerte mechanisch veränderbar sind.

RC ferngesteuert (remote controlled).

RC-Car ferngesteuertes Fahrzeug.

RC-Cars ferngesteuerte Fahrzeuge.

SW Software.

Switch Ein Kopplungselement, welches Netzwerkgeräte miteinander verbindet.

SYS System, welches die vier Subsystemene mit jeweiligen Komponenten beinhaltet.

Testautomatisierung Die Testautomatisierung beschreibt die Automatisierung von Testaktivitäten.

Transition Beschreibt die Übergangsbedingung für den Wechsel eines Zustands.

Trimmung Beschreibt die Ausrichtung von Körpern in eine gewünschte Lage. Hier: Ausrichtung der Längs- und Querbeschleunigung des Fahrzeuges auf die Nullstellung.

UDP Datenpaket Netzwerkdatenpaket, welches über das UDP Protokoll versendet wird.

UDP Server C-Modul, welches Funktionen zum Empfang von Netzwerkdatenpaketen bereitstellt.

VM Vorgehensmodell.

Watchdog Ein Watchdog ist eine (Software) Komponente, die andere Komponenten (hinsichtlich des zeitlichen Verhaltens) überwacht.

Literaturverzeichnis

[Aca] Acado Toolkit. http://acado.github.io/. – Letzter Zugriff: 01.03.2016

[CAR14] Universität Uppsala - CARS | Camera-Based Autonomous Racing System. http://cars.it.uu.se/. Version: 2014. – Letzter Zugriff: 22.02.2016

[Coo] CooCox is committed to providing free and open ARM Cortex M development tools to users. www.coocox.org/. – Letzter Zugriff:20.10.2016

[Dox] De facto standard tool for generating documentation from annotated C++ sources. www.doxygen.org/. – Letzter Zugriff:20.10.2016

[Dre] Microsoft Dreamspark. https://www.microsoft.com/germany/techwiese/techstudent/dreamspark/default.aspx. – Letzter Zugriff:27.02.2016

[Ecl] Quelloffenes Programmierwerkzeug zur Softwareentwicklung. https://eclipse.org/. – Letzter Zugriff:20.10.2016

[EGH+15] Ek, Johan ; Gunnarsson, Johannes ; Hellaeus, Viktor ; Insgard, Viktor ; Kuosku, Mattisa ; Norrblom, William: Chalmers University of Technology Goethenburg - Advanced vehicle control systems. (2015)

[Ent] STM32F4/29 Discovery - STM32F4 Discovery. http://stm32f4-discovery.com/stm32f429-discovery/. – Letzter Zugriff: 14.02.2016

[Est] Esterel Technologies. http://www.esterel-technologies.com. – Letzter Zugriff: 26.02.2016

[Fer] Kyosho Sender Perfex KT-18. https://www.kyosho.com/eng/products/rc/detail.html?product_id=103030. – Letzter Zugriff: 29.02.2016

[Fil] Longpass Filter LP780 Near-IR. http://midopt.com/filters/lp780/. – Letzter Zugriff: 12.02.2016

[Fly] FlyCapture SDK Point Grey. https://www.ptgrey.com/flycapture-sdk. – Letzter Zugriff: 17.02.2016

[Fol] 3M Reflexstreifen. http://reflexfolie.de/Reflexfolie/Reflexbaender/3M-Reflexstreifen-8850.html. – Letzter Zugriff: 12.02.2016

[Fon14] FONTANA, Giorgio: Autonomous Driving of Miniature Race Cars, Politecnico di Milano, Diplomarbeit, 2014

[GIT] GIT - Versionierungssoftwaresystem. https://git-scm.com/. – Letzter Zugriff:27.02.2016

[Ham] HAMMER, Tobias: HTerm Terminalprogramm. http://www.der-hammer.info/terminal/. – Zuletzt Aufgerufen: 08.11.2016

[HB13] HENDRIK BERNSMEYER, Christoph Dobiat et a. Tobias Biehl B. Tobias Biehl: Projektgruppe MIPSwarm - Mobile Inverted Pendular Swarm - Zwischenbericht. November 2013

[Hö08] HÖHN, Reinhard: Das V-Modell XT: Grundlagen, Methodik und Anwendungen. Springer, 2008

[Ink] Ink Scape. https://inkscape.org/de/. – Letzter Zugriff:27.02.2016

[Inn16] INNERN, Bundesministerium des: Das V-Modell. (2016). http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/Haeufig-gestellte-Fragen/haeufig_gestellte_fragen_node.html#doc4623024bodyText4. – Letzter Zugriff:20.10.2016

[IR] ILS - OSLON 4 IR Wide PowerStar. http://docs-europe.electrocomponents.com/webdocs/12ba/0900766b812ba142.pdf. – Zuletzt Aufgerufen: 12.02.2016

[JNA] GitHub - Java Native Access. https://github.com/java-native-access/jna. – Letzter Zugriff: 20.10.2016

[JU14] JUnit. http://junit.org/junit4/. Version: 2014. – Letzter Zugriff:20.10.2016

[Kam] Flea 3 FL3-U3-13Y3M-C Kamera. http://www.edmundoptics.de/cameras/usb-cameras/point-grey-flea3-usb-3-0-cameras/86767/. – Letzter Zugriff: 12.02.2016

[KYO] Kyosho Corporation. http://www.kyosho.com/. – Letzter Zugriff: 09.02.2016

[LDM13] LINIGER, Alexander ; DOMAHIDI, Alexander ; MORARI, Manfred: Optimization-based autonomous racing of 1:43 scale RC cars. (2013)

[LF15] LUKAS FELBER, Mirko Stocker et a. Michael Rüegg R. Michael Rüegg: CUTE - C++ Unit Testing Easier. http://cute-test.com/. Version: 2015. – Letzter Zugriff:20.10.2016

[Lyf16] General Motors and Lyft to Shape the Future of Mobility. http://media.gm.com/media/us/en/gm/news.detail.html/content/Pages/news/us/en/2016/Jan/0104-lyft.html. Version: 2016. – Letzter Zugriff: 12.02.2016

[Mat] MathWorks Matlab. http://de.mathworks.com/products/matlab/. – Letzter Zugriff: 01.03.2016

[MB09] MARIO BERNHART, Thomas G.: Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten. Pearson Studium, 2009

[Mip14] MIPSwarm - PG-Projekt "MIPSwarm"- Universität Oldenburg. http://mipswarm.informatik.uni-oldenburg.de. Version: 2014. – Letzter Zugriff: 01.07.2016

[MSP16] Microsoft Project - Projektmanagementsystem. https://www.microsoft.com/de-de/office/project/default.aspx. Version: 2016. – Letzter Zugriff:27.02.2016

[Obj] Lensagon CMFA0420ND Objektiv. http://www.lensation.de/de/shop/detail/detail/51-low-distortion-low-distortion/flypage/92-lensagon-cmfa0420nd-lensagon-cmfa0420ndf003.html?sef=thcfp. – Letzter Zugriff: 12.02.2016

[OE09] OSSWALD, Marc ; ENGELER, Florian: Autonomous real-time RC car racing systems. (2009)

[Ope] OpenCV. http://opencv.org/. – Letzter Zugriff: 14.02.2016

[Oraa] Java 2D Graphics and Imaging. http://docs.oracle.com/javase/8/docs/technotes/guides/2d. – Letzter Zugriff: 20.10.2016

[Orab] java.awt. http://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html. – Letzter Zugriff: 20.10.2016

[Orac] javax.swing. http://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html. – Letzter Zugriff: 20.10.2016

[ORC14] ETH Zurich - Autonomous RC Racing. http://control.ee.ethz.ch/~racing/. Version: 2014. – Letzter Zugriff: 26.02.2016

[rt] Realtime Kernels. https://capocasa.net/realtime-kernels. – Letzter Zugriff: 14.02.2016

[Rut10] RUTSCHMANN, Martin: Infrared based vision system for tracking 1:43 scale race cars. (2010)

[SCA] Esterel Technologies - SCADE Suite. http://www.esterel-technologies.com/products/scade-suite/. – Letzter Zugriff: 12.02.2016

[Scra] Scrum - Agiles Projektmanagement. http://scrum-master.de/Was_ist_Scrum/Scrum_auf_einer_Seite_erklaert. – Letzter Zugriff: 12.02.2016

[Scrb] Scrum - Agiles Vorgehensmodell. http://www.intellias.com/how-we-work/scrum/. – Letzter Zugriff: 12.02.2016

[SG10] SPENGLER, Patrick ; GAMMETER, Christoph: Modeling of 1:43 scale race cars. (2010)

[SSW+14] SCHÖN, Thomas ; SVENSSON, Andreas ; WAGBERG, Johan ; BJÖRSELL, Joachim ; BÄCK, Viktor ; ERIKSSON, Mattias ; NILSSON, Albin ; ÖHLUND, Lukas: CARS - Camera-based Autonomous Racing System. (2014)

[STMa] 14-ADC-DMA-Library (STM32F4) - Mikrocontroller. http://mikrocontroller.bplaced.net/wordpress/?page_id=667. – Letzter Zugriff: 05.11.2016

[STMb] 02-Button-Library (STM32F4) - Mikrocontroller. http://mikrocontroller.bplaced.net/wordpress/?page_id=456. – Letzter Zugriff: 29.02.2016

[STMc] FreeRTOS (STM32F4) - Mikrocontroller. http://http://www.freertos.org/. – Letzter Zugriff: 29.02.2016

[STMd] 01-LED-Library (STM32F4) - Mikrocontroller. http://mikrocontroller.bplaced.net/wordpress/?page_id=113. – Letzter Zugriff: 05.11.2016

[STMe] 12-UART-Library (STM32F4) - Mikrocontroller. http://mikrocontroller.bplaced.net/wordpress/?page_id=744. – Letzter Zugriff: 05.11.2016

[Tec16] Letzter Zugriff:08.11.2016

[Ubu] The leading OS for PC, tablet, phone and cloud | Ubuntu. http://www.ubuntu.com/. – Letzter Zugriff: 14.02.2016

[Ver14] VERSCHUEREN, Robin: Design and implementation of a time-optimal controller for model race cars, KU Leuven Fakultät Ingenieurswissenschaften, Diplomarbeit, 2014

[Vis] Die Komplettlösung für die Diagrammerstellung zur leicht verständlichen Vermittlung komplexer Informationen. https://products.office.com/de-de/visio/flowchart-software. – Letzter Zugriff:20.10.2016