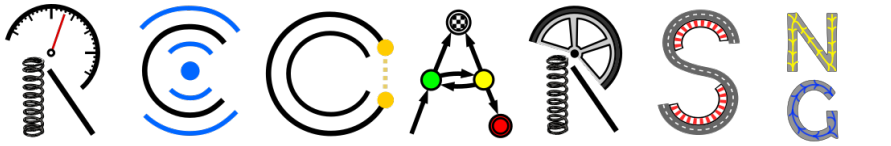


Projektgruppe RCCARSng

Realtime Controlled Cooperative Autonomous Racing System next generation

Endbericht

Philipp Borchers
Joost Brunken
David Fahr
Dario Feiling
Janis Kröger
Tobias Peikenkamp

Carl von Ossietzky Universität Oldenburg
Department für Informatik

Version 1.2
26.04.2018

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

Betreuer:
Dr.- Ing. Willem Hagemann
Dipl.- Inform. Stefan Puch
Dipl.- Inform. Günter Ehmen

Versionshistorie

Version	Datum	Autor(en)	Änderungen
1.0	06.04.2018	Projektgruppe RCCARSng	Finale Version des Berichts zum Ende der Projektlaufzeit
1.1	11.04.2018	Projektgruppe RCCARSng	Tippfehler behoben Layout angepasst
1.2	26.04.2018	Projektgruppe RCCARSng	Finale Version des Berichts zum Abschlussvortrag

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Im Juni 2017 sorgte die deutsche Bundeskanzlerin Angela Merkel auf einer Dienstreise in Argentinien für Aufsehen: „Wir werden in 20 Jahren nur noch mit Sondererlaubnis selbstständig Auto fahren dürfen.“ Es werde das autonome Fahren geben [Vit]. Anhand dieser Aussage wird deutlich, dass die Bedeutung des Themenfeldes autonomes Fahren gestiegen ist und in näherer Zeit weiter an Bedeutung gewinnt. Erste Gesetzesentwürfe für die Zulassung von autonom fahrenden Fahrzeugen für den deutschen Straßenverkehr sind bereits auf den Weg gebracht [Bun] und Industrie und Forschung treiben die Entwicklung voran.

Vor dem Hintergrund dieser Entwicklung haben die Abteilungen *Hybride Systeme* und *Sicherheitskritische Eingebettete Systeme* der Carl von Ossietzky Universität Oldenburg im Wintersemester 2015/16 das Projekt *RCCARS* initiiert. Die Abkürzung *RCCARS* steht dabei für **R**ealtime **C**ontrolled **C**ooperative **A**utonomous **R**acing **S**ystem und soll ein eingebettetes, sicherheitskritisches System zur autonomen Steuerung von kleinen Modellrennfahrzeugen realisieren. Das Projekt startete mit einer für ein Jahr befristeten Projektgruppe, die ihre Arbeit an einer ersten Ausbaustufe bereits im Sommer 2016 mit Erfolg abschließen konnte. Im Sommersemester 2017 wurde eine neue Projektgruppe mit dem Namen *RCCARSng* (**R**ealtime **C**ontrolled **C**ooperative **A**utonomous **R**acing **S**ystem **n**ext **g**eneration) aus 6 Studierenden gegründet, welche die Arbeit nun fortführt.

1.1. Struktur des Dokuments

Mit diesem Bericht werden die Arbeit, Ergebnisse und die Vorgehensweise der Projektgruppe *RCCARSng* dokumentiert. Er wurde im Laufe der Projektzeit sukzessive um Fortschritte und Entwicklungen erweitert und mündet zum Abschluss der Projektgruppe in diesen Endbericht.
Die Ausführungen beginnen in Kapitel 2 mit einer kurzen überblicksartigen Vorstellung des Systems. In diesem Zusammenhang wird auch auf die Arbeit der vorhe-

rigen Projektgruppe *RCCARS* eingegangen. Anschließend wird die Zielsetzung der Weiterentwicklung des bestehenden Settings vorgestellt und ein neues Szenario formuliert.
Im Anschluss wird in Kapitel 3 die Planung des Projektes im Projektmanagement vorgestellt. Neben der Vorgehensweise und Zeitplanung beinhaltet das Kapitel formulierte Zwischenziele und das Testmanagement. In Kapitel 4 folgt eine inhaltliche Einführung in das bestehende System, indem zum Einen auf Referenzprojekte anderer Universitäten und studentische Arbeiten an der Carl von Ossietzky Universität Oldenburg eingegangen wird, zum Anderen werden Planungsalgorithmen, welche zur Berechnung der zu fahrenden Trajektorien benötigt werden, vorgestellt. In Kapitel 5 wird der Ausgangszustand des übernommenen Systems der Projektgruppe *RCCARS* im Detail vorgestellt. Des Weiteren werden in diesem Kapitel die zur Erfüllung des neuen Szenarios notwendigen Anpassungen beschrieben.
Die aus dem neuen Szenario, dem bestehenden System und den nötigen Anpassungen resultierenden Anwendungsfälle und Anforderungen sind in den Kapiteln 6 und 7 formuliert. In Kapitel 8 ist anschließend eine Sicherheitsanalyse zusammen mit einem Sicherheitskonzept beschrieben. Auf die Architektur und Schnittstellen des Systems sowie einzelner Komponenten wird im Kapitel 9 eingegangen. Anschließend sind in Kapitel 10 die Durchführung und Ergebnisse notwendiger Evaluationen beschrieben und in Kapitel 11 ist der Entwicklungsstand bis zum 2. Review am 15.11.2017 dokumentiert. Die Beschreibung des endgültigen Entwicklungsstandes bis zum 06.04.2018 erfolgt in Kapitel 12. Die Vorgehensweise in der Testplanung werden in Kapitel 13 gesondert betrachtet. Abschließend folgt in Kapitel 14 eine Zusammenfassung der erreichten Ergebnisse, mögliche Anpassungen für weitergehende Arbeiten und ein abschließendes Fazit zur Arbeit der Projektgruppe *RCCARSng*.

1.2. Rechtliches

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

2. Szenario

In diesem Kapitel werden die Ziele der Projektgruppe *RCCARSng* bezüglich der Thematik des autonomen Fahrens vorgestellt. Zuerst wird in Abschnitt 2.1 eine kurze Einführung in das System gegeben, welches von der Projektgruppe *RCCARS* entwickelt wurde. Anschließend wird in Abschnitt 2.2 eine Long Term Vision eingeführt, die aufzeigt, welche Ausbaustufen es für zukünftige Projektgruppen oder Abschlussarbeiten geben soll. In Abschnitt 2.3 wird ein neues Szenario vorgestellt, welches von der Projektgruppe in Zusammenarbeit mit den Betreuern erstellt wurde. Dieses Szenario dient als Beschreibung des Ziels, welches am Ende der Projektphase realisiert werden soll und leitet sich aus der Long Term Vision ab.

2.1. Einführung in das bestehende System der vorherigen Projektgruppe *RCCARS*

Wie bereits in der Einleitung erwähnt, realisierte die Projektgruppe *RCCARS* ein System zur autonomen Steuerung eines Modellrennfahrzeugs im Maßstab 1:43. In dieser Größenordnung ist das Fahrzeug zu klein, um es mit einer Kamera für eine Objekterkennung auszustatten, wie es in der Realität der Fall wäre. Daher ist über der Rennstrecke eine Infrarotkamera angebracht (siehe Abbildung 2.1), die das Fahrzeug auf der Rennstrecke erfassen kann. Um die Position des Fahrzeugs genau zu bestimmen, ist es mit Reflexionsmarkern ausgestattet. Ferner sind über der Rennstrecke Infrarotstrahler zur Ausleuchtung angebracht. Die Kamera sendet ihre Bilddaten an einen Rechner weiter, der Koordinaten und Ausrichtung des Fahrzeugs berechnet.
Anhand dieser Positionsdaten berechnet eine Steuerungssoftware, die auf einem Rechner installiert ist, die erforderlichen Steuerdaten für das Fahrzeug. Diese werden anschließend mittels einer zugehörigen Fernsteuerung, die zusätzlich mit dem Rechner verbunden ist, an das Fahrzeug gesendet.
Dieser Regelkreis wurde von der Projektgruppe *RCCARS* entwickelt. Sie hatte sich zu Beginn ein Szenario als Ziel gesetzt, das vorsah, ein Fahrzeug mit einer Geschwindigkeit von 1.5 m/s fünf Runden lang kollisionsfrei fahren zu lassen. Das Szenario

Abbildung 2.1.: Abbildung des Systems mit Rennstrecke, Fahrzeug und angebrachter Kamera

wurde vollständig bis auf die Geschwindigkeit umgesetzt. Die Steuerung des Fahrzeugs erfolgte dabei nach dem oben beschriebenen Regelkreis, die angestrebte Trajektorie des Fahrzeugs war die Mittellinie der Rennstrecke.

2.2. Long Term Vision

Das Projekt *RCCARS* wurde mit dem längerfristigen Ziel initiiert, ein Testfeld zur Entwicklung und Analyse von autonomen Rennstrategien zu entwickeln [RCC16, S. 9]. Es war ein mehrstufiger Entwicklungsprozess geplant, welcher in Abbildung 2.2

dargestellt ist. Die erste Entwicklungsstufe, die den Aufbau einer initialen Rennstrecke, sowie die autonome Steuerung der Längs- und Querführung eines Fahrzeugs beinhaltet, wurde von der Projektgruppe *RCCARS* umgesetzt.

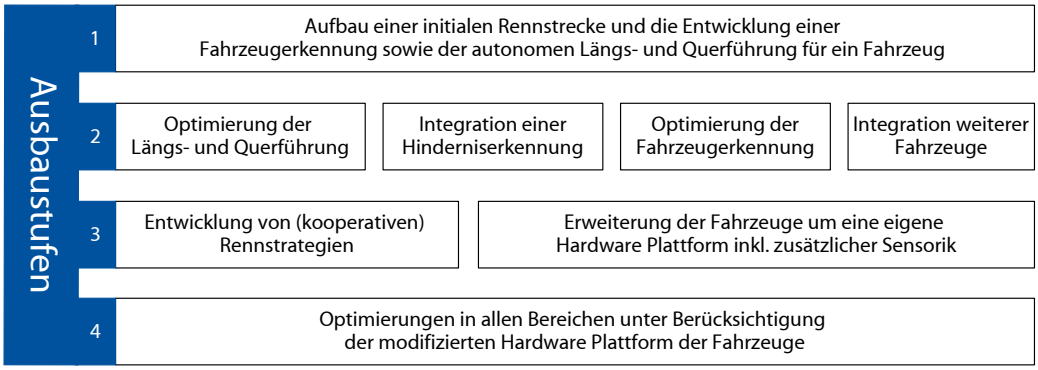


Abbildung 2.2.: Schema der stufenbasierten Entwicklung eines Testfeldes für autonomes Racing. (Quelle: [RCC16, Abb. 2.6, S. 9])

Die Projektgruppe *RCCARSng* versteht sich als Fortsetzung dieses Projekts und hat sich vor dem Hintergrund dieses Entwicklungsprozesses die Umsetzung der zweiten Ausbaustufe zur Aufgabe gemacht. Diese beinhaltet zunächst die Integration einer Hinderniserkennung und eine Rennsituation mit mehreren Fahrzeugen. Diese beiden Ziele ziehen zusätzlich eine Optimierung der Objekt- und Fahrzeugerkennung, sowie eine optimierte Fahrzeugsteuerung nach sich, wodurch diese zu Zielen der zweiten Ausbaustufe gehören.
Da die Projektgruppe *RCCARSng* an die Arbeit der Gruppe *RCCARS* aus dem Wintersemester 2015/2016 und dem Sommersemester 2016 anknüpft und somit als deren Fortsetzung zu verstehen ist, werden Hard- und Softwarekomponenten der vorherigen Projektgruppe übernommen. Die Hardware des bestehenden Systems muss jedoch um zusätzliche Komponenten, wie weitere Fernsteuerungen und mehrere Rechner zur Steuerung der Fahrzeuge, ergänzt werden. Da das Rennsystem neben der Arbeit für Projektgruppen und Abschlussarbeiten auch als Exponat eingesetzt werden und damit transportabel bleiben soll, kommt als weiteres Ziel hinzu, eine verbesserte Halterung für Kamera und Infrarotstrahler zu konstruieren. Damit soll eine flexible und schnelle Montage an unterschiedlichen Trägern, wie Decke und Gestell, ermöglicht werden. Zusätzlich muss auch die Software der vorherigen Projektgruppe erweitert werden, sodass das System in der Lage ist, mehrere Fahrzeuge autonom zu steuern. Weiterhin soll die Erkennung der Ausrichtung der Rennstrecke verbessert werden und eine Anpassung der Software zur Systemsteuerung stattfinden.

Zur Erstellung einzelner Softwarekomponenten wird SCADE [Est] eingesetzt. SCADE eignet sich zur modellbasierten Entwicklung sicherheitskritischer Software und generiert zertifizierten Code, der zur Regelung und Steuerung der Fahrzeuge benutzt werden kann.

Im Folgenden sind die Ziele für die Realisierung der zweiten Ausbaustufe in einem Szenario mit dem Titel „unfallfreier Überholvorgang“ zusammengefasst, welches sich an dem Szenario der Projektgruppe *RCCARS* orientiert (Vgl. [RCC16, Kapitel 2.4, S. 10]). Es soll im Rahmen dieser Projektgruppe realisiert werden.

2.3. Realisierung des Szenarios

Das Szenario „unfallfreier Überholvorgang“ sieht vor, dass zwei Fahrzeuge auf einer geschlossenen Rennstrecke autonom fahren. Vor Beginn des Rennbetriebes wird die Mindestdurchschnittsgeschwindigkeit für jedes Fahrzeug und das Fahrverhalten der Fahrzeuge von einem Administrator festgelegt. Zusätzlich können vor Beginn des Rennbetriebes Hindernisse auf der Fahrbahn platziert werden. Hierbei ist vorgesehen, dass sich mindestens zwei Objekte auf der Rennstrecke befinden, wobei sowohl Fahrzeuge als auch Hindernisse als Objekte zählen. Die Fahrzeuge fahren solange, bis sie eine Mindestrundenanzahl von zehn Runden kollisionsfrei zurückgelegt haben. Dabei ist es für beide Fahrzeuge möglich sich gegenseitig zu überholen und den Hindernissen auf der Rennstrecke auszuweichen.

Die in Abbildung 2.3 illustrierte Rennstrecke befindet sich in einem geschlossenen Raum, der 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 die Fahrbahn und die Fahrzeuge regelmäßig von Staub befreit, um den Fahrzeugen eine optimale Bodenhaftung zu bieten. Vor jedem Rennen wird daher von einem Administrator geprüft, ob die Fahrzeuge auf der Fahrbahn starten können. Ferner wird sichergestellt, dass die Fahrzeuge unbeschädigt sind und die Verbindungen mit den Fernsteuerungen intakt bleiben. Zuschauer müssen einen Mindestabstand zur Rennstrecke einhalten, um weder die Sensorik des Systems, noch den Zustand der Fahrbahn zu beeinträchtigen. Sind alle Sicherheitsvorschriften eingehalten, kann der Rennbetrieb gestartet werden. Über ein externes Steuersystem kann der Administrator den Startbefehl geben, sodass beide Fahrzeuge starten. Hierbei sei angemerkt, dass die Fahrzeuge ausschließlich von der autonomen Steuerungseinheit kontrolliert werden. Ein manuelles Eingreifen durch Personen in

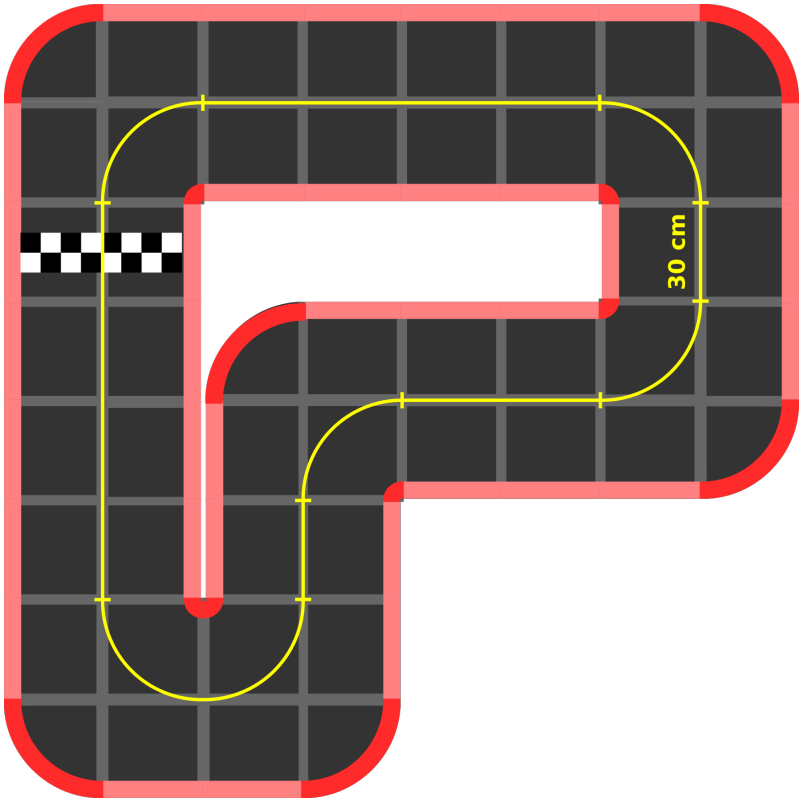


Abbildung 2.3.: Schematischer Aufbau der Rennstrecke. [RCC16, Abb. 2.7, S. 11]

das Fahrgeschehen ist, außerhalb eines sicherheitskritischen Fehlerfalls in dem das Rennen mittels Not-Stop abgebrochen werden muss, nicht vorgesehen.
Das Rennen wird durch eine Kamera oberhalb der Rennstrecke überwacht, welche in Realzeit Bilder von den Fahrzeugen und der Rennstrecke an die Positionsbestimmung sendet. Die Positionsbestimmung errechnet aus den Bilddaten die Pose der Fahrzeuge und leitet die Daten an die Steuerungseinheiten weiter. Diese entscheiden auf Grundlage der vorliegenden Daten welches Fahrverhalten angemessen ist und senden entsprechende Steuerbefehle an die zugehörigen Fahrzeuge. Das Fahrverhalten umfasst das Überholen und Folgen anderer Fahrzeuge. Für beide ist das Umfahren von Hindernissen ebenfalls vorgesehen.
Das Szenario gilt als erfolgreich abgeschlossen, wenn die Fahrzeuge mindestens zehn Runden zurückgelegt haben, ohne mit der Fahrbahnbegrenzung, mit Hindernissen oder miteinander zu kollidieren und das ausgewählte Fahrverhalten eingehalten wurde. Falls Hindernisse und Fahrzeuge auf der Fahrbahn platziert sind, sodass ein Fahrzeug keine Runde vollständig abschließen kann, geht das System in einen sicheren Fehlerzustand über und bringt jedes Fahrzeug zum Stehen. Auch in diesem Falle gilt das Szenario als erfüllt.

Da in der Beschreibung des Szenarios durch die Wahl von Geschwindigkeiten und Fahrverhalten, sowie der Platzierung von Hindernissen Freiheiten gegeben sind, ergeben sich für das Szenario verschiedene Varianten. Im Folgenden werden einige Varianten vorgestellt, die den Zielen: Realisierung von Überholvorgängen, Integration mehrerer Fahrzeuge sowie der Hinderniserkennung entsprechen. Damit soll gezeigt werden, dass die Ziele der Projektgruppe *RCCARSng* durch das Szenario in verschiedenen Varianten abgedeckt ist.

Variante 1 - Folgen anderer Fahrzeuge

Die erste Variante des Szenarios sieht vor, dass sich zwei Fahrzeuge und keine Hindernisse auf der Rennstrecke befinden. Das Fahrverhalten ist für beide Fahrzeuge „Passives Fahren“. So müssen in dieser Variante 10 Runden kollisionsfrei bewältigt werden, indem ein Fahrzeug einem weiteren folgt. Es findet also kein Überholvorgang statt.

Variante 2 - Überholvorgang

Auch in der zweiten Variante befinden sich zwei Fahrzeuge und keine Hindernisse auf der Rennstrecke. Die Mindestdurchschnittsgeschwindigkeiten ist so zu wählen, dass für das Fahrverhalten „Überholen“ in zehn Runden mindestens ein Überholvorgang stattfindet.

Variante 3 - Hindernisintegration

In der dritten Variante ist vorgesehen, dass sich sowohl zwei Fahrzeuge, als auch bis zu 16 Hindernisse auf der Rennstrecke befinden. Es kann sowohl das Fahrverhalten „Überholen“ als auch „Passives Fahren“ gewählt werden. Solange es die Positionierung der Hindernisse ermöglicht, sollen auch in dieser Variante 10 Runden bewältigt werden. Aufgrund einer potentiell hohen Anzahl und einer beliebigen Anordnung von Hindernissen wird nicht garantiert, dass ein Überholvorgang stattfinden kann. Dennoch wird gewährleistet, dass ein Fahrzeug dem anderen folgt. Im Fall, dass die Rennstrecke so blockiert ist, dass kein Fahrzeug eine Runde abschließen kann, werden die Fahrzeuge zum Stehen gebracht.

3. Projektmanagement

Ein gut geführtes Projektmanagement ist für ein erfolgreiches Projekt essentiell. Im folgenden Kapitel wird das Projektmanagement der Gruppe *RCCARSng* beschrieben. Dazu wird das verwendete Vorgehensmodell Scrum, die Vorgehensweise der Projektgruppe in der anfänglichen Einarbeitungsphase und das derzeitige Arbeitsmodell vorgestellt. Folgend wird die Zeitplanung der zu erreichenden Ziele und das Zeitmanagement der Gruppe erläutert. Damit verbunden wird im Risikomanagement aufgezeigt, welche Maßnahmen ergriffen werden können, falls Aufgaben durch gegebene Umstände nicht zu erfüllen sind oder wie anderen Problemen vorgegriffen werden kann. Anschließend werden die im Team für das Projekt verwendeten Werkzeuge und die ggf. damit verbundenen Kosten im Kostenmanagement beschrieben. Außerdem wird auf das Testmanagement und die angewandte Teststrategie eingegangen. Die Testergebnisse sind im Kapitel 13 zusammengefasst. Der letzte Teil des Projektmanagement Kapitels befasst sich mit der Öffentlichkeitsarbeit, die während der Projektdauer stattgefunden hat.

3.1. Vorgehensmodell

Ein Vorgehensmodell ist in der Systementwicklung von besonderer Bedeutung. Es beschreibt Richtlinien, Vorgehensweisen oder Empfehlungen, die für eine erfolgreiche Produktentwicklung erforderlich sind. Zu den bekanntesten Vorgehensmodellen in der Softwareentwicklung gehört das agile Entwicklungsmodell Scrum. Dessen agile Arbeitsweise drückt sich vor allem durch ein selbstorganisiertes Team, eine iterative und inkrementelle Vorgehensweise und wenig Bürokratie aus. Durch das gegebene Produkt, das von der Gruppe weiterentwickelt werden soll, erscheint Scrum in einer für das Team angepassten Form als Vorgehensmodell angemessen. In den folgenden Abschnitten soll das Entwicklungsmodell kurz allgemein vorgestellt werden. Die Anpassungen durch die Gruppe werden in Abschnitt 3.2.2 erläutert.
Scrum beinhaltet drei verschiedene Rollen, die am Prozess beteiligt sind. Diese sind der Product Owner, der Scrum Master und das Entwicklungsteam. Der Product Owner repräsentiert den Kunden und stellt die nötigen fachlichen Anforderungen,

die durch das Entwicklungsteam umgesetzt werden müssen. Der Scrum Master achtet darauf, dass der Scrum Prozess ordnungsgemäß eingehalten wird und beseitigt mögliche außerhalb der Entwicklung liegende Probleme, auf die das Team stößt.
Der Ablauf teilt sich, wie in Abbildung 3.1 dargestellt, in mehrere Meetings sowie Phasen ein, die wiederholt werden, bis ein fertiges Produkt vorliegt. Die zu bearbeitenden Aufgaben werden in Scrum vom Product Owner (PO) definiert und als sogenannte User Stories in einem Product Backlog gesammelt und priorisiert. Diese User Stories beinhalten die Anforderungen aus der fachlichen Sicht eines Nutzers und sollen den Entwicklern Gelegenheit geben, die nötigen Schritte bzw. Aufgaben zu verstehen. Dazu werden die Stories durch das Team geschätzt, um ihnen einen Referenzwert zuzuteilen und ggf. Missverständnisse frühzeitig zu erkennen. Ein beliebtes Schätzverfahren ist das Planning Poker, welches in Abschnitt 3.2.2 näher betrachtet wird.

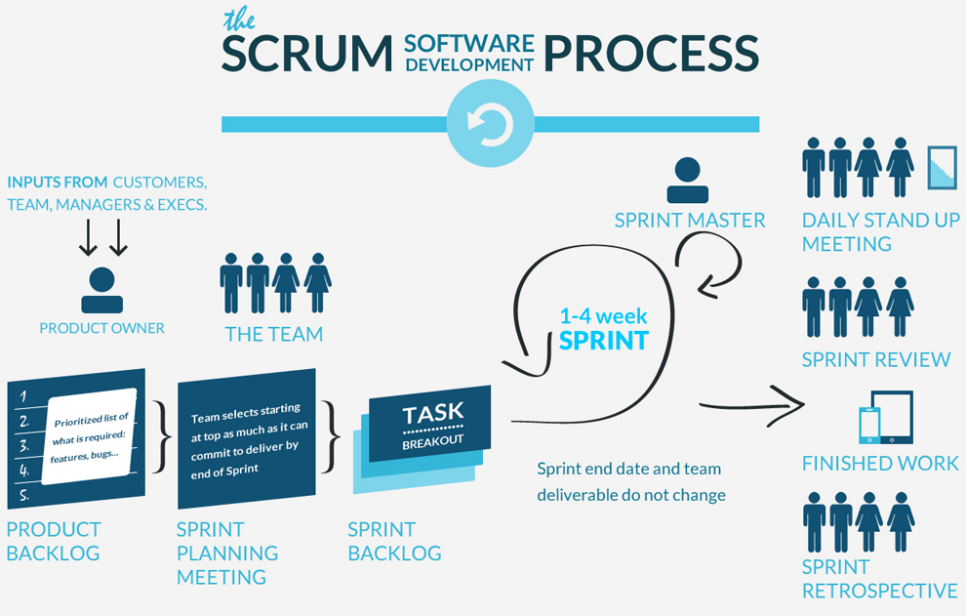


Abbildung 3.1.: Das agile Softwareentwicklungsmodell Scrum in Übersicht [SCR].

Der Scrumprozess startet mit dem ersten gemeinsamen Treffen, dem sogenannten Sprint Planning. Dabei findet eine Planung darüber statt, was in nächster Zeit vom Team bearbeitet werden soll. Die zu bearbeitenden User Stories werden in das Sprint Backlog gezogen, welches als Richtlinie für den nächsten Sprint gilt. Im Anschluss an das Sprint Planning werden User Stories in technisch formulierte Tasks für die Entwickler aufgespalten.
Nach dem Sprint Planning folgt der sogenannte Sprint in Scrum. Dieser bezeichnet die Entwicklungsphase, in der alle sich im Sprint Backlog befindenden Aufgaben

bearbeitet werden. Ein Sprint umfasst je nach Größe des Projekts und der Arbeitsweise einen Zeitraum zwischen einer und vier Wochen. Während dieser Zeit sorgt unter anderem der Scrum Master dafür, dass das Team ungestört arbeiten kann. Während eines Sprints werden keine neuen Aufgaben aus dem Product Backlog in das Sprint Backlog gezogen.
Innerhalb eines Sprints finden täglich kurze teaminterne Treffen von maximal 15 Minuten statt. In diesen sogenannten Dailys wird besprochen, was die Entwickler bisher geschafft haben, welche Probleme ggf. dabei aufgetreten sind und wie sie weiter arbeiten möchten.
Nach einer Sprintperiode folgt ein weiteres gemeinsames Treffen mit dem Product Owner, das Sprint Review. Beim Sprint Review geht es darum, dem Product Owner das durch das Team Erarbeitete zu präsentieren und mit dessen Erwartungen abzugleichen. Weil es sich bei Scrum um ein inkrementelles Modell handelt, muss bei einem Review immer ein funktionales und auslieferbares Produkt vorliegen. Direkt im Anschluss eines Reviews findet wieder ein neues Sprint Planning statt und der Prozess beginnt von Neuem, bis ein fertiges Produkt vorliegt.
Als teaminternes Treffen findet nach dem Review innerhalb des Sprints eine Sprint Retrospektive statt. In dieser werden alle im Team zu behandelnden Probleme und Schwierigkeiten, aber auch positive Aspekte der letzten Sprintphase besprochen.

3.2. Vorgehensweise Projektgruppe

Da die Projektgruppe im Sommersemester 2017 das erste Mal zusammen arbeitet und kein festes Vorgehensmodell durch die Aufgabenstellung vorgegeben wurde, unterscheidet sich die Arbeitsweise vor dem ersten Review, die als Einarbeitungsphase betrachtet wurde von dem Arbeiten nach einem festgelegten Vorgehensmodell in der Entwicklungsphase. Die einzelnen Arbeitsphasen werden im diesem Abschnitt und für den jeweiligen Stand fortführend beschrieben.

3.2.1. Vorgehensweise Einarbeitungsphase

Den Beginn der Projektarbeit bildete die Einarbeitung in das Projekt *RCCARS* der vorherigen Projektgruppe. Um eine bessere Arbeitsaufteilung zu erreichen, wurden einige Rollen im Team festgelegt, die während der Zeit der Projektarbeit eingehalten werden sollen. Zu den festgelegten Rollen gehörten unter anderem Projektleitung, Dokumentenbeauftragter, Entwickler und ein Verantwortlicher für die Pflege der Webseite. Auf dieser wird ein regelmäßiger Blog darüber geführt, was die Projektgruppe neben dem Projektverlauf im Rahmen der Öffentlichkeitsarbeit leistet.

Zudem wurde ein wöchentliches Treffen mit allen Beteiligten inklusive den Betreuern angesetzt, bei dem wichtige Themen, wie das bisherige Vorgehen und der Arbeitsstand besprochen und neue Aufgaben für die folgenden Wochen erarbeitet werden sollten. Als Ergänzung wurden durch die Gruppe zwei weitere wöchentliche Treffen vereinbart, in der zusammen gearbeitet werden kann. Da während der Einarbeitungsphase noch keine User Stories formuliert und Sprints, sowie Sprint Reviews als nicht notwendig erachtet wurden, entschied sich die Gruppe dazu, erst nach dem ersten Review mit der Implementierung neuer Funktionen zu beginnen und gemäß eines festen Vorgehensmodells zu arbeiten.
Zum Überwachen und Planen der Aufgaben wurde durch die Projektleitung ein Zeitplan in Excel angelegt, ein beispielhafter Ausschnitt ist in Abbildung 3.2 zu sehen. Dieser Zeitplan beinhaltete alle bis zum ersten Review wichtigen und zu bearbeitenden Aufgaben. Im Verlauf der Projektarbeit wurde beschlossen, zur Übersichtlichkeit und Pflege des Zeitplans ein Projektmanagement Tool einzusetzen. Nach einiger Recherche, fiel die Wahl auf die Webanwendung *Agantty*. Diese wird im Abschnitt 3.2.2 genauer beschrieben.
Neben dem Zeitplan wurden weitere Dokumente angelegt, um organisatorische oder inhaltliche Beschlüsse festzulegen. Zudem wurden regelmäßig Protokolle über die Gruppentreffen geführt, in denen sich zugleich die Arbeitsaufträge für die folgende Woche befanden. Zusätzlich zu den gemeinsam erarbeiteten Dokumenten führte jede Person einen eigenen monatlichen Tätigkeitsbericht, um selbst abgeschlossene Arbeit zu dokumentieren.

	A	B	C	D	E	F	G
1	**Themenbereich**	**Aufgabenbeschreibung**	**Verantwortliche/**	**Begin**	**Deadline/En(**	**Status**	**Begründung**
2	Öffentlichkeitsarbeit	Webseite aktualisieren	Janis		23. Mai	Erledigt	
3	Dokumentation	Anforderungen definieren	PG		23. Mai	Erledigt	
4	Organisation	Aufgabenbereiche verteilen	PG		24. Mai	Erledigt	
5	Organisation	Einteilung Systemanalyse	Projektleitung		24. Mai	Erledigt	
6	Dokumentation	Anforderungen, Rahmenbedingungen und Szenario zusammentragen	PG		30. Mai	Erledigt	
7	Sonstiges	MS Projects Evaluation	Projektleitung		31. Mai	Verschoben	Problem bei Lizenzbeschaffung
8	Dokumentation	Testplanung	Dario, Janis, Tobias		01. Jun	Erledigt	
9	Dokumentation	Präsentationsfolien (Template)	Tobias		01. Jun	Erledigt	
10	Organisation	Backlog füllen	Projektleitung		04. Jun	Verschoben	Vor dem Review kein Sprint mehr
11	Vortrag	Vortrag Regelungstechnik	PG	06. Jun	06. Jun	Erledigt	
12	IdeenExpo	IdeenExpo Vorbereitungen	Dario + Janis		07. Jun	Erledigt	
13	Dokumentation	Logo erstellen	Philipp		07. Jun	Erledigt	
14	Sonstiges	Email an Dietrich Boles wegen Boule-Turnier	Projektleitung		08. Jun	Erledigt	
15	Öffentlichkeitsarbeit	IdeenExpo Blogeintrag	Janis		08. Jun	Erledigt	
16	Öffentlichkeitsarbeit	Neues Logo auf Webseite einpflegen	Janis		09. Jun	Erledigt	

Abbildung 3.2.: Ausschnitt aus der Zeitplanungstabelle der Gruppe *RCCARSng*.

3.2.2. Vorgehensweise Entwicklungsphase

Die Entwicklungsphase der Projektgruppe teilt sich in die Zeit zwischen dem ersten und zweiten Review, sowie zwischen dem zweiten Review bis zum Abschluss des Projekts auf.

Vorgehensweise 1. Review bis 2. Review

Nach dem Review und der Einarbeitungsphase begann für die Projektgruppe *RC-CARSng* die Entwicklungsphase. Hierbei sollte das in Abschnitt 3.1 beschriebene Vorgehensmodell Scrum zum Einsatz kommen. Dieses wurde durch die PG, wie im späteren Verlauf beschrieben, so angepasst, dass es sich für die Weiterentwicklung des Projekts eignet.
Da für die Projektgruppe kein Product Owner zur Verfügung stand, wurden User Stories von dem gesamten Team definiert und priorisiert. Dabei fand eine Orientierung an den in Abschnitt 3.3 beschriebenen Zwischenzielen statt. Formalitäten wie das Pflegen der Backlogs wurden an die Projektleitung übertragen.
Um die Aufgaben zu schätzen, wurde das Schätzungsverfahren „Planning Poker“ verwendet. Dabei wird jeder Entwickler des Teams eingebunden, indem er den jeweiligen Aufwand mit verdeckten Planning Poker Karten schätzt. Diese beinhalten einen Kartensatz von ca. 10 Karten mit unterschiedlichen Werten von 0 bis 100. Durch das anonyme Schätzen können bei komplexen Aufgaben Unstimmigkeiten ermittelt und sonst eher ruhigen Mitgliedern der Gruppe Gelegenheiten zum aktiven Beitrag geboten werden.
Die Sprintlänge wurde auf drei Wochen festgelegt. Diese Zeit wurde als akzeptabel eingeschätzt, um in ihr genügend Änderungen an dem System vornehmen zu können und in der Präsentation innerhalb des Sprint Reviews einen neuen Arbeitsstand darzustellen.
Da für die Abnahme der Aufgaben innerhalb des Sprint Reviews auch der Product Owner fehlt, wurden hierfür von der Gruppe Dokumente in Form einer „Definition of Ready“ und „Definition of Done“ formuliert. Die Definition of Ready beinhaltet alles Nötige, was ein Entwickler zum Bearbeiten einer Aufgabe braucht. Die von der Gruppe *RCCARSng* festgelegte Definition of Ready beinhaltet folgende Punkte:

- Aufgabenstellung ist definiert.
- Aufgabenstellung ist von jeder bearbeitenden Person verstanden.
- Akzeptanzkriterien sind definiert.
- Akzeptanzkriterien sind von jeder bearbeitenden Person verstanden.

- Abhängigkeiten zur Aufgabenerfüllung sind abgeschlossen.
- Komplexität der Aufgabe ist geschätzt.
- Die Aufgabe ist priorisiert.
- Die für die Aufgabe benötigte Hard- und Software ist vorhanden.

Die Definition of Done beschreibt, unter welchen Kriterien eine Aufgabe innerhalb des Reviews als abgeschlossen gelten darf. Die Projektgruppe hat sich dabei auf die folgenden Anforderungen geeinigt:

- Fertige, lauffähige Software muss im Git eingecheckt sein.
- Akzeptanzkriterien sind erfüllt.
- Code-Review muss durch mindestens eine nicht bearbeitende Person durchgeführt sein.
- Code muss ausreichend kommentiert sein. (Doxygen)
- Kurze, stichpunktartige Dokumentation zum Vorgehen beim Lösen der Aufgabe ist geschrieben.
- Die Aufgabe wird von der absoluten Mehrheit (bzgl. der anwesenden Gruppenmitglieder) als abgeschlossen angesehen.
- Die Aufgabe muss von mindestens einer nicht bearbeitenden Person durch Komponenten, Integrations und Regressionstests erfolgreich getestet sein.

Zum weiteren Einschätzen, ob sich die Projektgruppe auf dem richtigen Weg befindet, werden gegebenenfalls die Betreuer um eine kurze Rückmeldung gebeten.
Um eine Übersicht über den derzeitigen Stand der Arbeit zu behalten, wurde sich für das kostenfreie Webtool Trello entschieden. Dieses soll von der Projektgruppe *RC-CARSng* als Sprintboard genutzt werden. Wie es in der Projektgruppe zum Einsatz gekommen ist, wird ausführlicher in Abschnitt 3.2.2 beschrieben.
Da ein tägliches Treffen aufgrund des nebenbei laufenden Semesters unmöglich ist, wird das Daily zu einem Weekly umfunktioniert, bei dem wie beim Daily nun wöchentlich die Arbeitsfortschritte der einzelnen Mitglieder besprochen werden können. Das wöchentliche Gruppenmeeting, sowie interne zwei Treffen pro Woche inklusive Protokoll und anderen bereits angelegten Dokumente innerhalb des Projektmanagements wurden im weiteren Verlauf durchgehend weitergeführt.
Neben den inhaltlichen Themen, wurden auch außerhalb der Entwicklung Verantwortlichkeiten der Gruppe erweitert, sodass jedes Gruppenmitglied einen besonderen

Aufgabenbereich erhielt. Die vorhandenen Rollen beinhalten Projektleiter, stellvertretender Projektleiter, Testbeauftragter, Verantwortlicher für die Öffentlichkeitsarbeit, Dokumentenbeauftragter und Hardwarebeauftragter.
Um die Retrospektiven abzuhalten, wurde von der Projektleitung festgelegt, dass jedes Projektgruppenmitglied mindestens drei positive und drei negative Eindrücke über den derzeitigen Entwicklungsstand der Projektgruppe, die Arbeitsmoral der Projektgruppenmitglieder oder Probleme innerhalb der Gruppe aufschreibt.
Diese werden anschließend vor der Gruppe einzeln präsentiert. Dabei werden die negativen Eindrücke analysiert und besprochen, welche Maßnahmen ab dem folgenden Sprint ergriffen werden können, um diesen entgegen zu wirken.

Arbeiten mit Agantty

Agantty ist ein Projektmanagement Tool, welches als Webanwendung aufgebaut ist. Die Gründe, sich für *Agantty* zu entscheiden, waren neben einem ansprechenden Design und einer intuitiven Nutzung vor allem, dass es sich um eine vollständig kostenlose Anwendung ohne versteckte Kosten handelte. Durch eine Webanwendung war zudem der Zeitplan und die Aufgabenübersicht für jedes Mitglied zeit- und ortsunabhängig verfügbar. *Agantty* wird von einem deutschen Team entwickelt und befindet sich per 256 Bit SSL Verschlüsselung auf Servern in Deutschland. Dies schaffte zusätzliches Vertrauen in die Anwendung.
Das Tool bietet der Projektgruppe zwei unterschiedliche Übersichten zu dem Projektplan. Die Grundansicht besteht aus einem Gantt-Diagramm (siehe Abbildung 3.3), das von der Projektleitung gepflegt wird. Dazu werden alle Aufgaben mit Zuordnung zu einem Mitglied der Gruppe eingetragen. Zusätzlich bietet die Anwendung eine Dashboard Ansicht, in der eine Übersicht zu allen Aufgaben geboten wird. Als weitere Funktionen werden der Projektleitung ein Rechtemanagement bereitgestellt und die Möglichkeit, Meilensteine in den Zeitplan zu integrieren. Als Erinnerungsstütze kann jedes Mitglied personalisiert einstellen, ob und wann persönliche Berichte mit Aufgaben per E-Mail an sie versendet werden sollen.

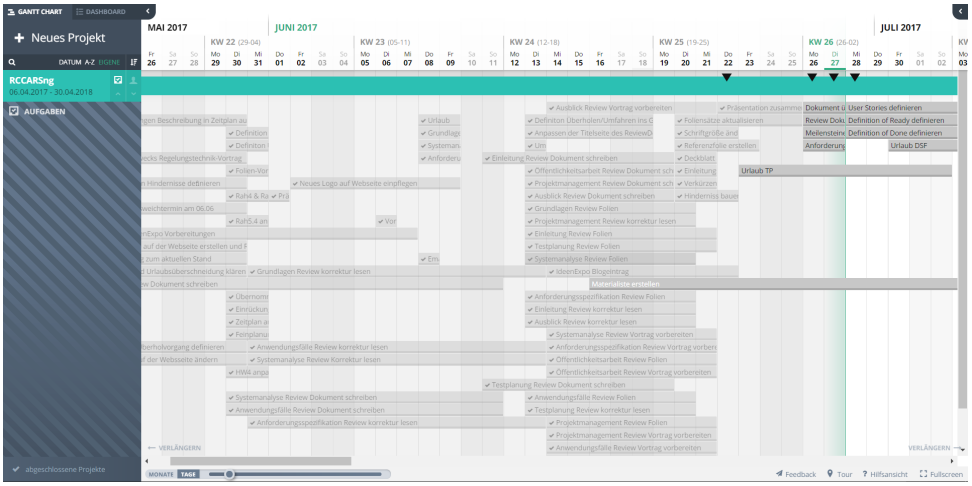


Abbildung 3.3.: Gantt-Diagramm der Projektgruppe *RCCARSng* in *Agantty*.

Arbeiten mit Trello

Trello ist eine kostenfreie, webbasierte Projektmanagementsoftware des Unternehmens Atlassian. Die Hauptfunktion, die Trello bietet, ist unterschiedliche Arten von *Boards* zu erstellen. Da die meisten Tools, die Sprintboards für Scrum abbilden können (wie beispielsweise *Jira*), kostenpflichtig sind, sah die Projektgruppe in Trello eine gute Alternative. Zur Nutzung in Scrum wurde Trello als Sprintboard umfunktioniert wie in Abbildung 3.4 zu sehen ist. Von der Projektgruppe wurde zur Übersicht des aktuellen Arbeitsstandes, mehrere Spalten angelegt.
Dieses beinhaltet die Spalte „Product Backlog“, dass alle noch zu bearbeitenden Aufgaben enthält. Innerhalb eines Sprintplannings werden dann Aufgaben in die Spalte „Sprint Backlog“ gezogen, die für die nächsten 3 Wochen zu bearbeiten sind. Gruppenmitglieder können sich aus der Spalte „Sprint Backlog“ bedienen, und sich selbst Aufgaben zuordnen. Dieses kennzeichnet Trello mit dem Kürzel dieser Person. Zusätzlich verschiebt diese Person dann den Task in die Spalte „In Preparation“. Hier muss anhand der „Definition of Ready“ überprüft werden, ob diese Aufgabe zur Bearbeitung bereit ist.

Sind alle Kriterien erfüllt, kann die Aufgabe in die nächste Spalte, „In Progress“, geschoben werden. Sobald die Bearbeitungen der Aufgabe abgeschlossen sind, kommt diese in die Spalte „Testing“. Jetzt ist die Aufgabe zum Testen bereit und muss von einem nicht bearbeitendem Gruppenmitglied getestet werden. Falls Bugs während des Testens auftreten und noch genügend Zeit während des Sprints übrig bleibt, werden diese sofort korrigiert.

Sollte die Sprintlaufzeit zu Ende sein, wird ein neuer Task für den Bugfix in das „Sprint Backlog“ des folgenden Sprints aufgenommen. Sollten alle Tests abgeschlossen sein und die Aufgabe entspricht nach der „Definition of Done“ einem fertigen Zustand, kann diese in die letzte Spalte, „Done“, gezogen werden.

Abbildung 3.4.: Abbildung des Sprintboards der Projektgruppe *RCCARSng* in *Trello*.

Evaluation

Die ersten Sprints innerhalb der Projektgruppe verliefen etwas unkoordiniert. Dies lag zum Einen daran, dass die meisten Mitglieder noch nie mit dem Vorgehensmodell Scrum gearbeitet hatten und organisatorische Fragen aufkamen, die den Arbeitsprozess verlangsamten. Des Weiteren wurde, wie oft bei unerfahrenen Teams, die Einschätzung der einzelnen Tasks und die zu bearbeitende Zeit unterschätzt. Dies führte in den ersten Wochen zu überfüllten Sprints, bei denen nur ein Bruchteil der eigentlich geplanten Aufgaben wirklich abgeschlossen werden konnte.
Die Problematik, die von der Projektleitung identifiziert wurde, bestand darin, dass Aufgaben, die in der Spalte „Testing“ lagen, nicht eigenständig von anderen Gruppenmitgliedern in den vorgegebenen drei Wochen getestet wurden.
Diese Problematiken hatten zur Folge, dass die Projektgruppe weiter mit der Entwicklung fortgeschritten war als im Zeitplan vorgesehen, aber die Testabdeckung vernachlässigt wurde.
Die Retrospektiven zwischen den Sprints waren sehr erfolgreich und brachten zum Teil interne Probleme frühzeitig zum Vorschein. Dazu gehörten beispielsweise ein zu unregelmäßiges, gemeinsames Entwickeln oder die Atmosphäre innerhalb von Diskussionen, die für einige Gruppenmitgliedern zu unorganisiert war.

Anpassungen des Prozesses

Um den Problemen der ersten Sprints entgegen zu wirken, beschloss die Projektgruppe *RCCARSng* den Sprint in zwei Phasen einzuteilen. Dazu gehören zwei Wochen Entwicklungszeit, sowie eine Woche Testzeit. Nach zwei Wochen intensiven Arbeitens an den Aufgaben erfolgt ein Entwicklungsstopp. Es sollten sich alle Tasks danach in der Spalte „Testing“ befinden. In der folgenden Woche ist vor allem der Testbeauftragte dafür verantwortlich, dass jedes Mitglied Tests durchführt. Dabei ist darauf zu achten, dass niemand seine eigene Arbeit testet. Sollten alle Tests vor Abschluss des Sprintendes abgeschlossen sein, darf die restliche Zeit weiter an Bugs oder der Entwicklung gearbeitet werden.
Zudem wurden interne Probleme gelöst, die während der Retrospektiven erarbeitet wurden.

Vorgehensweise 2. Review bis 3. Review

Nachdem die Projektgruppe im 2. Review den aktuellen Entwicklungsstand und das weitere geplante Vorgehen vorgestellt hatte, wurden Änderungen des Projektmanagements für das weitere Vorgehen vorgenommen. Dabei folgte wie geplant eine weitere Entwicklungsphase, die in Sprints aufgeteilt wurde. Da zum Teil größere Aufgaben während der Entwicklungszeit innerhalb eines Sprints hinzukamen, wurden diese von nun an in der Planung markiert, um die Visualisierung der Entwicklungssprints nicht zu verfälschen. Dadurch, dass bis zum 8. Sprint der Entwicklungsphase keine weiteren Anpassungen im Prozess nötig waren, wurde der erarbeitete angepasste Scrumprozess (siehe Abschnitt 3.1) von der Projektgruppe gelebt. Durch auftretende Abweichungen zum ursprünglichen Zeitplan, beschloss *RCCARSng* die folgenden Sprints zu einer durchgängigen Entwicklungsphase zusammenzufassen (siehe Abschnitt 3.4). Dies war nötig, um die Projektmitglieder nicht in der intensiven Implementierung zu stören und die Entwicklung durch zu diesem Zeitpunkt unnötige Planungen zu unterbrechen.
Zudem wurde nach dem 2. Review auf den Gebrauch von Agantty (siehe Abschnitt 3.2.2) verzichtet. Der Grund war, dass es sehr schwer und schnell unübersichtlich wurde, unterschiedliche Systeme im Projektmanagement zu pflegen und aktuell zu halten. Als Alternative nutzte die Projektleitung eigens online erstellte Zeitstrahle. Während der restlichen Entwicklungszeit wurde weiterhin mit Trello (siehe Abschnitt 3.2.2) als Sprintboard gearbeitet. Zudem wurde das Board genutzt, um eigene Sprints für das Schreiben des Endberichts und die Vorbereitungsphase für das 3. Review angelegt.

3.2.3. Evaluation

Nachdem sich die Projektgruppe *RCCARSng* an den Arbeitsprozess gewöhnt hatte, war ein guter und effektiver Fortschritt in der Entwicklung zu sehen. Durch die regelmäßigen Treffen und das Aktualisieren des Sprintboards in Trello war jederzeit abrufbar, an welcher Aufgabe welches Gruppenmitglied gerade arbeitet. Zudem wurden während der wöchentlichen Treffen alle Projektbeteiligten auf den selben Wissenstand bezüglich der Entwicklungen gebracht. Auch das Testen wurde in den folgenden Sprints regelmäßiger durchgeführt und durch die eingeführte Testwoche in den Prozess integriert.
Dennoch mussten im Laufe der Entwicklungsphase auch wieder einige Änderungen am Prozess beziehungsweise der Zeitplanung vorgenommen werden. Da in den letzten beiden Monate der Projektzeit mehr Zeit in die Implementierung gesteckt werden musste, um die restlichen Zwischenziele erreichen zu können, wurde der vorgesehene Arbeitsablauf in den Hintergrund gestellt. Diese Änderungen sind in Abschnitt 3.4 erläutert.
Eine gesamte Bewertung des Arbeitsprozesses und Projektmanagements findet im Fazit und Ausblick Kapitel 14 statt.

3.3. Zwischenziele

Der Abschnitt Zwischenziele befasst sich mit den Zwischenzielen, die sich die Projektgruppe *RCCARSng* gesetzt hat, um das Ziel „unfallfreier Überholvorgang“ erfolgreich zu realisieren. Diese Zwischenziele gilt es zu erarbeiten, um den Entwicklungsprozess in kleine Abschnitte einzuteilen und zum Ende des Projektes das entwickelte Szenario (siehe Kapitel 2) bestmöglich auf der Rennstrecke umzusetzen.
Das Projekt *RCCARSng* baut auf ein vorheriges Projekt auf. Es verfügt daher schon über eine Basis, die für die weiterführende Entwicklung genutzt wird.

3.3.1. Zweites Fahrzeug

Das Szenario „unfallfreier Überholvorgang“ sieht vor, dass die Anzahl der Fahrzeuge, die gleichzeitig auf der Rennstrecke fahren auf zwei Fahrzeuge erhöht wird. Dies soll in diesem Teilziel realisiert werden.
Da bei diesem Schritt auf das bestehende System aufgebaut wird, ohne das ein anderes Zwischenziel umgesetzt werden muss, geht die Projektgruppe von dem aktuellen System ohne Veränderungen aus. Beide Fahrzeuge fahren hier auf einer festen Tra-

jektorie und mit festen Bremszonen nebeneinander her. Die wahrscheinlich größte Hardwareanpassung während des Projekts geschieht in diesem Zwischenziel. Diese Anpassung beinhaltet die Inbetriebnahme der Regelungsrechner für die einzelnen Fahrzeuge. Außerdem soll die Positionsbestimmung auf einem eigenen Rechner laufen. Des Weiteren muss eine weitere Fernbedienung angeschafft und eine zweite CarControl hergestellt und angepasst werden. Zusätzlich soll die Positionsbestimmung nach dem ersten Zwischenziel fähig sein auf der Rennstrecke platzierte Hindernisse zu erkennen. Sollten sich Hindernisse auf der Rennstrecke befinden, sollen die Fahrzeuge vorerst nicht losfahren, nachdem das System gestartet wurde.

3.3.2. Offline Trajektorienberechnung durch Simulation

Für das zweite Zwischenziel hat sich die Projektgruppe *RCCARSng* vorgenommen innerhalb einer Simulationsumgebung eine neue Trajektorie zu berechnen und diese als CSV auszugeben, sodass diese von der Control-Uniteingelesen werden kann. Die Simulation plant um die vorhandenen Hindernisse eine Trajektorie. Die Hindernisse auf der Rennstrecke müssen dabei an genau die Stellen platziert werden, wie in der Simulation. Die Fahrzeuge fahren mit dem alten Setting die neue Trajektorie ab.
Das Ziel ist, dass ein Fahrzeug den Hindernissen mit der vorher berechneten Trajektorie ausweicht. Die Berechnung der Trajektorie und das Übertragen der CSV geschieht vor Start des Rennbetriebs und kann zu einem späteren Zeitpunkt nicht mehr angepasst werden. Hindernisse, die einmal platziert sind, dürfen während des Rennbetriebes nicht bewegt werden.

3.3.3. Offline Trajektorienberechnung durch Control-Unit

In diesem Zwischenziel soll die Trajektorienberechnung von der Simulationsumgebung in das eigene System überführt werden, sodass keine CSV mehr benötigt wird.

3.3.4. Implementierung von MPC

Innerhalb dieses Zwischenziels soll der vorhandene PID-Regler durch eine modellprädikative Regelung ersetzt werden. Hindernisse werden auch hier vor Beginn des Rennbetriebs auf der Strecke platziert. In diesem Zwischenziel wird die Trajektorie weiterhin vor Start des Rennens berechnet. Die Projektgruppe möchte hier auf eine Masterarbeit eines Mitglieds der ehemaligen Projektgruppe RCCARS eingehen. Diese beschäftigt sich näher mit einer Möglichkeit, das Verhalten eines Fahrzeugs vorauszusagen. Durch die Anpassung der Regelung werden die Weichen zum Zwischenziel 5 gelegt.

3.3.5. Passives Fahren + Überholvorgang

Im Zwischenziel „Überholvorgang“ wird die Projektgruppe versuchen, die Fahrverhalten „Passives Fahren“ und „ Überholvorgang“ zu implementieren. Für das „Passive Fahren“ wird eine vorher berechnete Trajektorie verwendet, der beide Fahrzeuge folgen. Sollte der Abstand zwischen den Fahrzeugen zu gering werden, bremst das auffahrende Fahrzeug automatisch ab. Es findet kein Überholvorgang statt. Ebenfalls sollen die Fahrzeuge anhalten, wenn ein Hindernis so auf der Strecke platziert ist, dass ein Umfahren nicht mehr möglich ist.
Um einen Überholvorgang durchzuführen, müssen zwei Fahrzeuge auf der Rennstrecke fahren. Beide Fahrzeuge fahren auf der Rennstrecke mit zuvor berechneter Trajektorie. Durch eine dynamische Anpassung der Trajektorie soll der Überholvorgang realisiert werden. Ziel ist es, dass das erste Fahrzeug vom Zweiten überholt wird und somit das Szenario (siehe Abschnitt 2.3) als erfolgreich abgeschlossen gilt.

3.4. Zeitmanagement

Die Zeitplanung wurde von der Projektleitung innerhalb *Agantty* (siehe Abschnitt 3.2.2) vorgenommen. Hierzu wurden alle von der Gruppe zu erledigenden Aufgaben mit einem Anfangs- und Enddatum versehen und in die Webanwendung eingepflegt. Es wurde mit einem Arbeitsaufwand von 16 Stunden pro Person pro Woche geplant.
Die definierten Zwischenziele wurden als Meilensteine in den Zeitplan aufgenommen. Für die Bearbeitung pro Zwischenziel werden von der Projektleitung jeweils 6 Wochen (2 Sprints) Arbeitszeit eingeplant. Die Überlegung dazu ist, dass in den Sprintreviews, die alle drei Wochen stattfinden, zur Hälfte der Zeit analysiert werden kann, ob ein Zwischenziel in der restlichen Zeit realisiert werden kann oder eine Umplanung stattfinden muss. Zusätzlich wurden weiteren Termine, wie Einführungen in Softwaresysteme oder Termine der Öffentlichkeitsarbeit, außerhalb der eigentlichen Entwicklung in die Zeitplanung eingepflegt. Ein grober Zeitplan mit den geplanten Meilensteinen ist in Tabelle 3.1 abgebildet. Zusätzlich wurde eine grafische Ausarbeitung eines Zeitstrahls in D.2(a) erarbeitet.

Zeitmanagement Evaluation 1. bis 2. Review

Zum Anfang des Projekts hatte die Projektleitung einen Zeitplan für den Bearbeitungszeitraum erstellt. Vor dem 2. Review wurde evaluiert, ob die derzeitige Planung realisierbar ist, oder Anpassungen durchgeführt werden müssen. Als Ergebnis kam

Zwischenziel/Projektphase	**Datum/Zeitraum**
Einarbeitungsphase	06.04.17 - 11.07.17
1. Review	28.06.17
Begin Entwicklungsphase	12.07.17
Zwischenziel 1: Zweites Fahrzeug	13.09.17
Vorbereitungszeit Zwischenreview	14.09.17 - 21.09.17
Zwischenreview	22.09.17
Zwischenziel 2: Offline Trajektorienberechnung durch Simulation	11.10.17
Vorbereitungszeit 2. Review	19.10.17 - 14.11.17
2. Review	15.11.17
Zwischenziel 3: Offline Trajektorienberechnung auf Control-Unit + Passives Fahren	06.12.17
Weihnachtspause	23.12.17 - 07.01.18
Zwischenziel 4: Implementierung von MPC	31.01.18
Zwischenziel 5: Überholvorgang + Ende Entwicklungsphase	14.03.18
Vorbereitungsphase 3. Review	15.03.18 - 16.04.18
3. Review	17.04.18

Tabelle 3.1.: Übersicht über die geplanten Meilensteine und die Phasen des Projekts.

heraus, dass zu diesem Zeitpunkt im Vergleich zum ursprünglichen Zeitplan genau ein Sprint (3 Wochen) fehlte. Dies war dadurch zu Schulden gekommen, dass ein Zwischenreview zwischen den ersten beiden Reviews eingeplant wurde. Zur Vorbereitung hatte die Projektgruppe eine Woche eingeschoben. Der darauf folgende Sprint wurde um eine Woche verlängert (4 Wochen) um Rückstände im Testen aufzuholen, die in den ersten 3 Sprints durch nicht ausreichende Planung des Prozesses vernachlässigt wurden. Die dritte Woche, die zu Lasten des Entwicklungsstandes kam, wurde zur Vorbereitung auf das 2. Review verwendet. Für diese Zeit wurde ursprünglich nur 3 Wochen eingeplant. Insgesamt können bis Ende des Projekts so nur noch 9 anstatt 10 Entwicklungssprints durchgeführt werden. Deutlich wird dies in Grafik D.2(a).

Zum Zeitpunkt des 2. Reviews stehen somit noch 5 Entwicklungssprint für die Realisierung von den 3 letzten Zwischenzielen zur Verfügung. Ursprünglich waren 2 Sprints pro Zwischenziel eingeplant. Die Projektgruppe hatte entschieden den ursprünglichen Zeitplan inklusive Meilensteine möglichst beizubehalten. Demnach war für die Realisierung des Zwischenziels 3 nach dem 2. Review ein Sprint eingeplant bis zum 06.12.2017. Danach wäre die Projektgruppe wieder in der ursprünglichen Zeitplanung.

Entscheidend für das weitere Einhalten der aktuellen Zeitplanung war, dass einer der folgenden Zwischenziele in nur einem Sprint realisiert werden könnte. Dazu wurde das Zwischenziel 3 angepasst wie in Kapitel 14 beschrieben.

Zeitmanagement Evaluation 2. bis 3. Review

In der Entwicklungsphase nach dem 2. Review hat die Gruppe die aktuelle Zeitplanung am 06.12.2017 evaluiert und ist zum Schluss gekommen, dass die Planung so wie in Zeitstrahl D.2(a) dargestellt und im 2. Review vorgestellt beibehalten wird. Zu diesem Zeitpunkt war die Simulation der offline Trajektorienberechnung funktionsfähig an die Control-Unit angebunden. Während des 6. Sprints vor den Weihnachtsferien hat sich die Projektgruppe hauptsächlich mit der Planung der Architektur und Schnittstellen für die Control-Unit beschäftigt. Dies hat den geplanten Entwicklungsstand etwas zurück geworfen.
Das 4. Zwischenziel, „MPC“, musste dadurch um drei Wochen nach hinten geschoben werden. Bedingt durch das Semesterende und anstehenden Klausuren der Projektmitglieder und durch den im 6. Sprint gelegten Fokus auf Planung und Dokumentation, standen zu wenig Ressourcen zur Verfügung, um das Zwischenziel zeitgerecht zu erreichen. Diese Maßnahme wurde schon vor dem 2. Review als alternative Zeitplanung (Abbildung D.2(b)) betrachtet.
Als weitere Maßnahme, um das 5. Zwischenziel noch zu erreichen, wurde der 8. und 9. Sprint zu einem großen Entwicklungssprint zusammengefasst und um 2 Entwicklungswochen verlängert. Für die anschließende Dokumentation blieben somit wie geplant zwei anstatt vier Wochen. Ab dem 06.04.2018 war die Dokumentation abgeschlossen und die Projektgruppe hat sich auf die Vorbereitung für das finale Review fokussiert.
Die grafische Darstellung der finalen Zeitplanung ist in Abbildung D.1 zu finden.

3.5. Risikomanagement

Die erste Phase, zeitgleich zur Anforderungserhebung des Projekts, wurde genutzt, um ein Risikomanagement auszuarbeiten. In der Entwicklungsphase wurde das Risikomanagement in das Vorgehensmodell Scrum integriert. In diesem Abschnitt wird auf die Risikostrategie der zweiten Phase des Projekts eingegangen.

Risikostrategie

Beim Risikomanagement geht es um das Reagieren auf Abweichungen des Normalprozesses. Diese müssen möglichst frühzeitig erkannt und behandelt werden, um Folgen einschätzen und entsprechend handeln zu können. Scrum konzentriert sich innerhalb des Vorgehensmodells auf die Steuerungsphase eines Projekts und definiert keinen aktiv gelebten Risikomanagementprozess.
Das Risikomanagement wird in vier Teilprozesse eingeteilt. Diese lassen sich ganz ähnlich wie der PDCA-Cycle (Plan, Do, Check, Act) als immer wieder zu durchlaufender Zyklus darstellen. Diese Teilprozesse sind wie folgt aufgeteilt: Identifizierung, Analyse, Planung und Steuerung. Dieser Zyklus wird bis zum Ende des Projekts angewandt und gepflegt.

Zur *Identifizierung* von Risiken werden innerhalb der Gruppentreffen Probleme und Abweichungen vom Prozess besprochen. Diese werden von der Projektleitung im Risikokatalog eingetragen. Risiken können aber auch zu jeder Zeit identifiziert werden und werden dann zur nächsten Gelegenheit gruppenintern besprochen und in den Risikokatalog aufgenommen.

Eine *Analyse* findet dann statt, wenn ein Risiko identifiziert wurde. Hier wird Ursachen- bzw. Auslösersuche betrieben. Außerdem wird die Eintrittswahrscheinlichkeit und das Ausmaß eines eventuellen Risikos bestimmt. Die Analyse der Risiken wird durch die Projektleitung durchgeführt.

Das *Planen* beinhaltet das Festlegen von präventiven und korrektiven Maßnahmen. Diese sind, wie die Einschätzung der Wahrscheinlichkeit und die Schadenshöhe, aus der Risikomatrix 3.2 auch im Risikokatalog für jedes Risiko aufgenommen.

Die *Steuerung* sorgt dafür, dass der Risikokatalog auf dem aktuellen Stand gehalten wird. Es wird geprüft, ob die Einschätzung eines Risikos mit dem aktuellen Projektstand übereinstimmt. Außerdem werden die präventiven und korrektiven Maß-

nahmen auf ihre Wirksamkeit kontrolliert. Risiken, die nicht mehr bestehen, werden archiviert. Im Falle, dass ein zuvor definiertes Risiko eintritt trägt die Projektleitung dafür sorge, dass die entsprechende korrektive Maßnahme durchgeführt wird und somit gegengesteuert wird.

Die identifizierten Risiken können in der Tabelle E.1 im Anhang eingesehen werden. Es folgt eine Risikomatrix mit der die identifizierten Risiken auf ihre Eintrittswahrscheinlichkeit und eventuelle Schadenshöhe eingeschätzt wurden. Der entsprechenede Wert wurde in dem Risikokatalog mit aufgenommen und soll den Risiken Nachdruck verleihen, die einen besonders hohen Wert zugeteilt bekommen haben. Des Weiteren hat die Projektleitung als Teil des Risikomanagements verschiedene Risikozeitpläne entwickelt wie beispielsweise im Anhang D.2(b) zu sehen ist. Diese beinhalten alternative Zeiten und ggf. angepasste Zwischenziele falls der geplante Zeitplan nicht einzuhalten ist.

Eintrittswahrscheinlichkeit / Schadenshöhe	**Sehr geringe Wahrscheinlichkeit**	**Geringe Wahrscheinlichkeit**	**Mittelere Wahrscheinlichkeit**	**Hohe Wahrscheinlichkeit**
Sehr geringer Schaden	1	2	3	4
Geringer Schaden	2	3	4	5
Mittlerer Schade	3	4	5	6
Hoher Schaden	4	5	6	7

Tabelle 3.2.: Risikomatrix.

3.6. Werkzeuge

In diesem Abschnitt wird die Software beschrieben, welche von der Projektgruppe genutzt wird. Dazu wird näher auf die Versionsverwaltung, das Projektmanagement, die Softwareentwicklung und Simulation eingegangen.

Versionsverwaltung Zur Versionsverwaltung wird das Softwaretool *GIT* [Tor] verwendet. *GIT* ermöglicht es, parallel und ortsunabhängig an einem Projekt zu arbeiten und erstellt hierzu sogenannte Branches („Abzweigungen“), an denen gleichzeitig gearbeitet werden kann und die zu einem späteren Zeitpunkt wieder gemerged („zusammengeführt“) werden können. Hierbei gilt, dass der Masterbranch („Hauptzweig“) immer die aktuellste Version bereit stellt.
Zusätzlich ist *GIT* Betriebssystem unabhängig.

Projektmanagement Die Projektleitung hat sich zu Beginn des Projekts entschieden, einen Zeitplan mit *Microsoft Excel* [Mica] zu erstellen. In diesem Zeitplan wurden anfallende Aufgaben aufgenommen. Jeder Aufgabe wurde eine Frist und eine oder mehrere Personen zugewiesen.

Da die Komplexität des Projekts schnell zunahm und die Übersichtlichkeit und Agilität im Zeitplan abnahm, hat sich die Projektgruppe für den Einsatz des Projektmanagement Tools *Agantty* [med] entschieden. In *Agantty* werden sogenannte Gantt-Diagramme verwendet, die es den Benutzern einfach machen, Aufgaben abzuarbeiten, sich selbst zu organisieren und Aufgaben miteinander zu verknüpfen. *Agantty* bietet den Vorteil, dass es sehr schnell einsatzbereit ist und in jedem Webbrowser aufgerufen werden kann. Innerhalb der Projektgruppe *RCCARSng* wird *Agantty* zur übergeordneten Zeitplanung eingesetzt. *Agantty* wird im Abschnitt 3.2.2 genauer beschrieben. Zur visuellen Darstellung hat die Projektleitung außerdem Gebrauch von *time.graphics* [Mus] gemacht. Mit Hilfe dieses Browsertools lassen sich beispielsweise Zeitstrahle zu Veranschaulichung für Präsentationen erstellen.
Ein weiteres Tool, dass die Projektleitung zur Verwaltung des Scrumprozesses eingeführt hat ist *Trello* [Atl]. *Trello* ist auch eine Webanwendung, in der Aufgaben definiert werden. Diese können dann von den Projektmitgliedern bearbeitet werden und durchlaufen so den Scrumprozess. *Trello* wird im Abschnitt 3.2.2 detaillierter dargestellt.

Softwareentwicklung Zur Softwareentwicklung wird das modellgetriebene Tool *SCADE* [Est] verwendet. *SCADE* wurde von der Carl von Ossietzky Universität im Jahre 2016 für die vorhergehende Projektgruppe *RCCARS* erworben. Deren Subsystem Control-Unit besteht zum Teil aus einem *C* Code, der mittels *SCADE* generiert wird. *SCADE* wird hier zur Implementierung der Regelungskomponente eingesetzt. *SCADE* wird vor allem im Entwicklungsbereich von sicherheitskritischer Software eingesetzt, zum Beispiel bei Flugkontrollsystemen und Fahrerassistenzsystemen. Zu *SCADE* gehört ein sogenannter *KCG Code Generator*, welcher unter der ISO 26262:2011 qualifiziert, sowie unter IEC 61508 2010 und EN 50128:2011 zertifiziert ist. Diese stellen sicher, dass die entwickelte Software sicherheitskritischen Standards entspricht.
Die Subsysteme Positionsbestimmung und Control-Unit des Systems *RCCARSng* sind in C++ realisiert, zur Bearbeitung und Erweiterung wird hier mit dem Codeeditor *Eclipse* [Fou] gearbeitet. Die Systemsteuerungssoftware wird in *Java* entwickelt. *Eclipse* unterstützt die Entwicklung von C++ und *Java* Programmen.

Als weiteres Tool wird die C++ Bibliothek *Armadillo* [SC16] genutzt. Mithilfe von *Armadillo* ist es möglich, *Matlab* Funktionen in C++ zu übersetzten.
Zum Testen des entwickelten Codes und der Komponenten wird die Test Datenbank der vorherigen Projektgruppe genutzt. Diese wurde im Rahmen von anderen Projekten im *OFFIS* entwickelt. Zur Abwicklung von Unit-Tests macht die Projektgruppe Gebrauch von *CUTE* [ESR]. Dieses ist ein *Eclipse* Plugin, welches das Durchführen von Komponententests vereinfacht und standardisiert.
Ein weiteres Tool, das zur Dokumentation von Softwarentwicklungen genutzt wird, ist *Doxygen* [Hee]. Mit Hilfe von *Doxygen* können Erklärungen im Quelltext in eine übersichtliche Dokumentation überführt werden, somit dient es einer vereinfachten Erstellung von Softwaredokumentationen.

Dokumentenverwaltung Zur Erstellung von Dokumenten wird von der Projektgruppe LaTeX [Teaa] eingesetzt. Insbesondere für Dokumente, welche für die drei anstehenden Reviews von Bedeutung sind, findet LaTeX Verwendung. Zusätzlich werden die Protokolle von Gruppentreffen mit LaTeX verfasst.
LaTeX ist ein Softwarepaket, das es ermöglicht, komplexe Dokumentationen mithilfe von einfachen Makros zu erstellen. Der große Vorteil von LaTeX ist, dass es rechner- und plattformunabhängig ist. In Kombination mit *GIT* (siehe Abschnitt 3.6) bietet dieses Zusammenspiel eine optimale Umgebung für große und flexible Projektarbeiten.
Zur Präsentationserstellung macht die Gruppe von *Microsoft Power Point* [Micb] Gebrauch.

Simulationsumgebung Da sich die Projektgruppe dazu entschieden hat, einige Verfahren erst zu simulieren, bevor eine Implementierung in das Live System stattfindet, wurde Gebrauch von *Matlab* [Mat] gemacht. Dieses ist ein Tool, welches für mathematische Operationen optimiert ist. Es können mathematische Probleme gelöst und grafisch dargestellt werden. Zusätzlich verwendet die Projektgruppe den Solver *Gurobi* [Gur], mit dem es möglich ist, lineare und quadratische Optimierungsprobleme zu lösen. Zur Anbindung von *Matlab* an das eigene System wurde die *Matlab Engine* Bibliothek verwendet, welche von den *Matlab*-Entwicklern zur Verfügung gestellt wird. Diese lässt es zu, *Matlab*-Code mithilfe von C++ auszuführen.

3.7. Kostenmanagement

Die Abteilungen *Hybride Systeme* und *Sicherheitskritische Eingebettete Systeme* der Carl von Ossietzky Universität Oldenburg finanzieren das Projekt *RCCARSng*. Da das Projekt als Folgeprojekt der Projektgruppe *RCCARS* gilt, ist das aktuelle System zu diesem Zeitpunkt komplett. Es bedarf lediglich kleinerer Investitionen.
Der Projektgruppe steht ein Budget von 3.000,- Euro zur Verfügung. Da der Großteil aller kostenintensiven Hardware schon angeschafft ist, wird davon ausgegangen, dass sich die Kosten zum Ende des Projekts weit unter dem Budget befinden werden. Die Gesamtkosten betragen sich auf 538,44 Euro.
In Tabelle 3.3 werden alle angefallenen Kosten aufgelistet.

Anzahl	**Investition**	**Kosten**
4	Akku á 23,90	95,60
4	Fahrzeugmotor á 13,90	55,60
7	Querlenkerstifthalterung	68,61
4	Hinterradaufhängeplatten	20,44
1	Reflexionsfolie	ca. 24,00
1	Schaumstoffmatte	13,90
1	Fahrzeugevaluation: Holz	38,88
1	Fahrzeugevaluation: Kabel	31,00
1	Fahrzeugevaluation: Schrumpfschlauch	5,00
1	Fahrzeugevaluation: Dioden	12,60
1	Fahrzeugevaluation: Fotowiderstände	6,50
1	Deckenhalterung	50,00
6	Rennstreckenmarker	20,00
4	Kabelkanal á 4,99	19,96
1	Hohlraumschrauben und Dübel	15,99
1	USB 3.0 Kabel inkl. Stromversorgung 10 m	37,95
1	Diverse Versandkosten	22,58
	Totale Kosten in €	**538,44**

Tabelle 3.3.: Anschaffungen und Kosten.

3.8. Testmanagement

Da die Arbeit der Projektgruppe *RCCARSng* unter sicherheitskritischen Aspekten stattfindet, ist es notwendig, dass die Funktionsfähigkeit des Systems durch Tests validiert wird. Tests ermöglichen es, Fehler im System zu erkennen oder sogar zu vermeiden, was bei der Rennsituation, mit der sich die Projektgruppe *RCCARSng* beschäftigt, von großer Bedeutung ist. In diesem Kapitel wird der Testprozess der Projektgruppe und deren Herangehensweise erläutert.
In Abschnitt 3.8.1 wird die Zeitplanung vorgestellt. Im Anschluss wird in Abschnitt 3.8.2 auf die Vorgehensweise beim Testen eingegangen und erklärt, wie aus den Anforderungen die notwendigen Tests abgeleitet werden.
In Abschnitt 3.8.3 wird die Testdatenbank vorgestellt, die zum Verwalten der Testfälle verwendet wird. Des Weiteren wird durch die Testdatenbank ermöglicht, dass Testfälle reproduzierbar durchgeführt werden können.
In Kapitel 13 der aktuelle Teststatus vorgestellt. Der Teststatus gibt ein Überblick über die erfüllten Anforderungen.

3.8.1. Zeitplanung

Das Testen und Validieren von Systemsoft- und Hardware ist eine anspruchsvolle Aufgabe, die viel Zeit benötigt. Nach dem von der Projektgruppe verwendeten Vorgehensmodell (siehe Abschnitt 3.1) sind dreiwöchige Sprints vorgesehen, in denen eine Systemkomponente fertig gestellt werden soll. Allerdings soll die entsprechende Komponente in dieser Zeit auch getestet worden sein.
Um dieser Anforderung gerecht zu werden, wurde die Sprintphase in 2 Phasen unterteilt. Ein Sprint besteht somit aus einer zweiwöchigen Entwicklungsphase, in der an der Komponente gearbeitet wird, und einer einwöchigen Testphase, in der die Testfälle geschrieben und durchgeführt werden (siehe Abbildung 3.8.1).

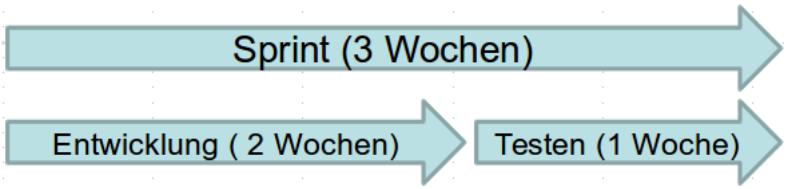


Abbildung 3.5.: Zeitplanung beim Testen der Projektgruppe *RCCARSng*.

Hierbei ist zu beachten, dass in der Testwoche das Testen die höchste Priorität erhält, sodass alle Projektgruppenmitglieder Tests erstellen oder durchführen. Nur wenn alle Testaufgaben verteilt und durchgeführt worden sind, kann weiter an der

Systementwicklung gearbeitet werden. Dies stellt sicher, dass alle Komponenten in einem Sprint auch getestet werden.

Sollten beim Testen Fehler auftreten, sodass eine Systemkomponente angepasst werden muss, so wird diese Aufgabe in den nächsten Sprint mit erhöhter Priorität aufgenommen oder die Aufgabe wird direkt erledigt, falls noch Ressourcen frei sind.

3.8.2. Vorgehensweise

Nachdem nun die zeitliche Planung für die Erstellung von Tests vorgestellt wurde, wird nun die Vorgehensweise beim Testen erklärt.
Die ersten Vorbereitungen für die Tests werden schon in der Entwicklungsphase (siehe Abschnitt 3.8.1) getroffen. Hier erstellt der Entwickler einer Systemkomponente Testfälle, die für den Tester beschreiben, was dieser zu testen hat. Diese Testfälle sind folgendermaßen aufgebaut:

- **ID**: Eine eindeutige Identifikation des Testfalls.
- **Name**: Ein aussagekräftiger Name des Testfalles.
- **Testtyp**: Die Art von Test, nach der getestet werden soll.
- **Anforderung**: Anforderungen, die durch diesen Testfall abgedeckt werden.
- **Anwendungsfall**: Die Anwendungsfälle, die den Testfall betreffen.
- **Bezug**: Bezug zu anderen Testfällen.
- **Testobjekt**: Systemkomponente, die getestet wird.
- **Verantwortlicher**: Entwickler der Komponente.
- **Deadline**: Deadline für die Durchführung des Tests.
- **Vorbedingungen**: Bedingungen, die vor dem eigentlichen Test erfüllt sein müssen.
- **Auslösendes Ereignis**: Das Ereignis, was den Testfall eintreten lässt.
- **Ergebnis**: Das erwartete Resultat bei korrekter Ausführung.

Am Ende der Entwicklungsphase sollten die Komponenten und Testfälle erstellt worden sein und die Testphase kann eingeleitet werden. Dies ist auch der Fall, falls die Komponente noch nicht fertig entwickelt wurde, sodass dann die Tests erst einmal so weit wie möglich erstellt werden.
Nun erfolgt zuerst die Zuordnung von Testern und den zu testenden Komponenten.

Diese Zuordnung wird über das Tool *Trello* ermöglicht (siehe Abschnitt 3.2.2). Hierbei werden Kreuztests durchgeführt. Somit wird sichergestellt das kein Entwickler seine eigenen Komponenten testet und mindestens eine zweite Person herangezogen wird, um die Funktionalität der Komponenten zu überprüfen. Der Testbeauftragte stellt sicher, dass die Zuordnung korrekt verläuft und alle Ressourcen auf Testaufgaben verteilt sind, sodass nun das eigentliche Testen erfolgen kann.

Testarten

In der Testphase wird jede neu entwickelte Komponente auf drei verschieden Arten getestet. Es werden Komponenten-, Integrations- und Regressionstests benutzt, um die Funktion und Kompatibilität der Komponente mit dem Gesamtsystem sicherzustellen (siehe Abbildung 3.6).

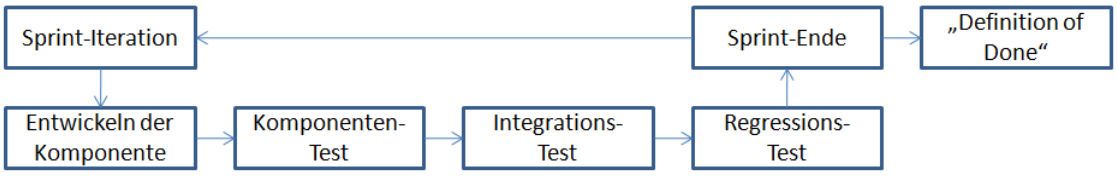


Abbildung 3.6.: Ausschnitt aus dem Ablauf eines Sprints mit Integration der Testabläufe. Hier werden die Komponenten in den Sprints getestet und am Ende der Sprints mit der Definition of Done abgeglichen. Vergleiche mit [HPA14].

Komponententests Die Komponententests überprüfen die entwickelte Komponente auf ihre Funktionsfähigkeit. Hier können Test-Frameworks verwendet werden, um diesen Prozess zu automatisieren.
Die Projektgruppe *RCCARSng* nutzt das Framework *CUTE*, welches das Erstellen von Unit-Tests in sogenannten Test-Suits ermöglicht. Diese können wiederholt ausgeführt werden und bieten so die Möglichkeit bestehende Funktionen zu testen.

Integrationstests Die Integrationstests überprüfen, ob sich die entwickelte Komponente korrekt mit anderen Komponenten zusammenfügen lässt, ohne dass bestehende Funktionalitäten eingeschränkt werden.
Die Projektgruppe *RCCARSng* nutzt hierbei die Tatsache aus, dass immer eine Version des Systems existiert, welche funktionstüchtig ist. In dieser Version wird dann zu Testzwecken die entwickelte Komponente eingebaut und dass System live getestet. Dies bedeutet zum Beispiel, das versucht wird mit der neuen Komponente das Szenario (siehe Abschnitt 2.3) zu erfüllen.

Regressionstests Die Regessionstests dienen der Überprüfung der Funktionsfähigkeit des gesamten Systems. Diese Tests werden nach der Integration einer neuen Komponente wiederholt durchgeführt, um die Resultate ständig zu überprüfen. So wird sichergestellt, dass neue Komponenten nicht die Funktionstüchtigkeit anderer Komponenten oder des kompletten Systems beeinträchtigen. Hierbei erleichtert die Verwendung von Test-Frameworks diese Aufgabe, sodass bestehende Test-Suits erneut gestartet werden können um die Funktionalität des Systems zu überprüfen.

3.8.3. Testverwaltung

Damit die Testfälle rekonstruierbar und nachvollziehbar verwaltet werden können, wird die Testdatenbank der Projektgruppe *RCCARS* weiterverwendet. Anhand dieser Testdatenbank lässt sich der aktuelle Teststatus der Projektgruppe ablesen. So kann erkannt werden, welche Testfälle erfolgreich abgeschlossen wurden, welche Tests noch fehlen und welche Tests fehlgeschlagen sind. Zudem lassen sich die Anforderungen erkennen, die bereits durch Testfälle abgedeckt wurden. In Abbildung 3.7 findet sich ein Ausschnitt der Testdatenbank.

TC Nr.	Bezeichnung	Anforderung	Testtyp	Version	Verantwortlich	Deadline	Ergebnisse
CU-HW-01	Hardware des Subsystems Control-Unit	HW3, HW5.3	Subsystemtest	0.1	Tobias Peikenkamp	2018-03-31	1
CU-HW-02	➥ Rechner der Control-Unit vorbereiten	HW3.1, HW5.1, HW5.3, HW5.4	Komponententest (Sprint)	0.2	Dario Feiling	2017-08-02	1
CU-HW-03	➥ Beschaffung neuer Rechner		Komponententest (Sprint)	0.4	Philipp Borchers	2017-08-02	1
CU-HW-05	➥ Softwarevorbereitung der Rechner		Komponententest (Sprint)	0.2	Dario Feiling	2017-08-02	1
CU-HW-04	➥ Fernsteuerung und CarControl von 2 Control-Units ist funktionstüchtig	HW3.3.1, HW3.3.2, HW3.3.3, HW3.3.4, HW3.3.5, HW3.3.7, HW3.3.8	Abnahmetest (Sprint)	0.3	Philipp Borchers	2017-08-23	1
CU-HW-06	➥ Neue CarControl ist Funktionstüchtig		Komponententest (Sprint)	0.4	Philipp Borchers	2017-08-02	2
CU-HW-07	➥ Leitungen neuer CarControl sind richtig verbunden		Komponententest (Sprint)	0.2	Philipp Borchers	2017-08-02	1
CU-HW-08	➥ FTDI Schnittstellen werden erkannt		Komponententest (Sprint)	0.1	Philipp Borchers	2017-08-02	1
CU-HW-09	➥ Sofwaretest des Entwicklungsboards		Komponententest (Sprint)	0.1	Philipp Borchers	2017-08-02	1
CU-HW-10	➥ LEDs sind funktionstüchtig		Komponententest (Sprint)	0.1	Philipp Borchers	2017-08-02	1
CU-HW-11	➥ Fernsteuerung ist mit dem Board korrekt verbunden	HW 3.3.1	Komponententest (Sprint)	0.2	Philipp Borchers	2017-08-02	1
CU-SW-01	Software des Subsystems Control-Unit	Sys1.3, Sys1.4, Sys4.1 - Sys4.7, Sys3, SW2, SW5	Subsystemtest	0.1	Tobias Peikenkamp	2018-03-31	0
CU-SW-02	➥ Zwei CSV-Dateien mit Trajektorien		Komponententest (Sprint)	0.6	Joost Brunken	2017-08-02	1
CU-SW-04	➥ Control-Unit reagiert und Empfangt Fahrzeugdatenpakete		Komponententest (Sprint)	0.1	Philipp Borchers	2017-09-04	0
CU-SW-05	➥ Reaktion auf Fehlercode: Hindernisse auf Rennstrecke		Komponententest (Sprint)	0.3	Philipp Borchers	2018-03-31	1
NW-SW-01	Netzwerk des Systems		Systemabnahmetest	0.2	Tobias Peikenkamp	2017-08-23	0
NW-SW-02	➥ Senden der Hindernisinformationen ins Netzwerk	SW1.5	Integrationstest	0.5	Dario Feiling	2017-08-02	1
NW-SW-04	➥ Versenden und Empfangen von Hindernispaketen via UDP		Komponententest (Sprint)	0.1	Tobias Peikenkamp	2018-09-13	1
NW-SW-03	➥ Senden von Konfigurationsdaten ins Netzwerk	SW3.1.8, SW3.1.9, SW3.1.10	Integrationstest	0.3	Tobias Peikenkamp	2017-08-02	0
NW-SW-05	➥ Versenden und Empfangen von Konfigurationspaketen via UDP		Komponententest (Sprint)	0.1	Tobias Peikenkamp	2018-09-13	1
NW-SW-06	➥ Senden von Fahrzeugdaten ins Netzwerk		Integrationstest	0.2	Dario Feiling	2017-10-18	1
PB-HW-01	Hardware des Subsystems Positionsbestimmung	HW2, HW5.2	Subsystemtest	0.3	Tobias Peikenkamp	2018-03-31	0
PB-SW-01	Software des Subsystems Positionsbestimmung	Sys1.2, SW1, SW4, SW5	Subsystemtest	0.4	Tobias Peikenkamp	2018-03-31	0
PB-SW-03	➥ Erkennung von Hindernisposen	SW1.3	Komponententest (Sprint)	0.4	Dario Feiling	2017-08-02	1
PB-SW-06	➥ Erkennen von Fehlern- Hindernis wurde Bewegt	SW1.8	Komponententest (Sprint)	0.3	Dario Feiling	2017-09-13	1
PB-SW-07	➥ Erkennen der Korrekten Anzahl der Hindernisse		Komponententest (Sprint)	0.1	Dario Feiling	2017-09-13	2
PB-SW-11	➥ Erkennen von Fehlern- zu viele Hindernisse auf der Strecke	SW1.8	Integrationstest	0.2	Dario Feiling	2017-09-13	1
PB-SW-04	➥ Laufzeit	SW1.6	Komponententest (Sprint)	0.1	Dario Feiling	2017-09-13	1
PB-SW-05	➥ Erkennung von mehreren Fahrzeugen	SW1.2	Komponententest (Sprint)	0.2	Dario Feiling	2017-09-13	1

Abbildung 3.7.: Ausschnitt aus der Testdatenbank der Projektgruppe *RCCARSng*. Hier zu sehen sind die eindeutigen IDs der Testfälle, der Name, die Anforderungen, die Testart, die Version, der Verantwortliche der Komponente, eine Deadline sowie der Teststatus.

3.9. Öffentlichkeitsarbeit

In den folgenden Abschnitten werden die einzelnen Öffentlichkeitsarbeiten näher beschrieben. Bestandteile sind die IdeenExpo, die Mathefahrt, die Erstsemester-Begrüßung, die Aktualisierung der Website und die Erstellung des Logos.

3.9.1. IdeenExpo

Die IdeenExpo ist eine Messe für Kinder und Jugendliche mit dem Ziel, diese für technische Berufe und MINT-Studiengänge (**M**athematik, **I**nformatik, **N**aturwissenschaften und **T**echnik) zu begeistern. Die Kinder, welche die Expo besuchen, sind im Alter von 11 bis 16 Jahren. Unter der Woche wird die Messe oft von Schulklassen besucht (Klasse 6 - 11). An den Wochenenden kommen noch viele Familien hinzu. Die IdeenExpo findet alle zwei Jahre auf dem Messegelände in Hannover statt. Auf der IdeenExpo 2017 hat die Projektgruppe *RCCARSng* den aktuellen Stand, mit dem von der Projektgruppe *RCCARS* entwickelten Demonstrator im Zeitraum vom 09.06. bis 18.06.2017 vorgestellt. Der Demonstrator wurde von Mitarbeitern des *OFFIS* und einigen Projektgruppenmitgliedern beaufsichtigt und betreut.
Das Ziel der Ausstellung des Demonstrators war, Besuchern der IdeenExpo die Probleme und Herausforderungen des autonomen Fahrens nahezubringen. Neben dem autonomen Fahrbetrieb wurde den Kindern und Jugendlichen auch die Möglichkeit geboten, selber die *dNaNo*-Fahrzeugen von *Kyosho* zu steuern. Um dies zu ermöglichen, wurde von Michael Bukowski aus der PG *RCCARS* eine kleine Software geschrieben, welche es ermöglicht, die Daten eines an den Rechner angeschlossenen *Xbox 360* - Controller an die CarControl weiterzuleiten. Die Besucher konnten sich somit an den Rundenzeiten der Computersteuerung messen. So wurden an fast jedem Messetag neue Rekorde aufgestellt. Manche der Kinder hatten, nachdem sie den Controller in die Hand bekommen hatten, kleine Probleme mit der Steuerung. Andere wiederum hatten keine Probleme die Steuerung zu erlernen. Die Verbesserung des Streckenrekordes von Tag zu Tag ist in Tabelle 3.4 einzusehen.

Datum	Zeit	Controller
13.06.2017	3.40 s	Kyosho
18.06.2017	3.64 s	Xbox
17.06.2017	3.70 s	Rennmodus
13.06.2017	3.76 s	Kyosho
15.06.2017	3.76 s	Xbox
17.06.2017	3.77 s	Rennmodus
18.06.2017	3.77 s	Xbox
17.06.2017	3.79 s	Rennmodus
13.06.2017	3.85 s	Xbox

Tabelle 3.4.: Top 10 Liste der Rekorde mit Datum und Steuerungsmethode.

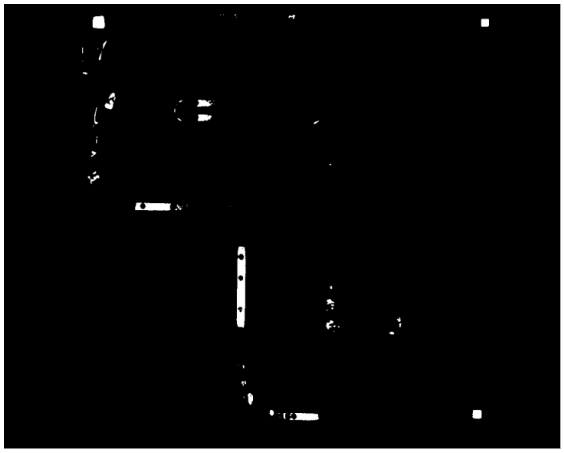


Abbildung 3.8.: Gefilterte Bildmatrix aufgenommen um 14:30 Uhr.

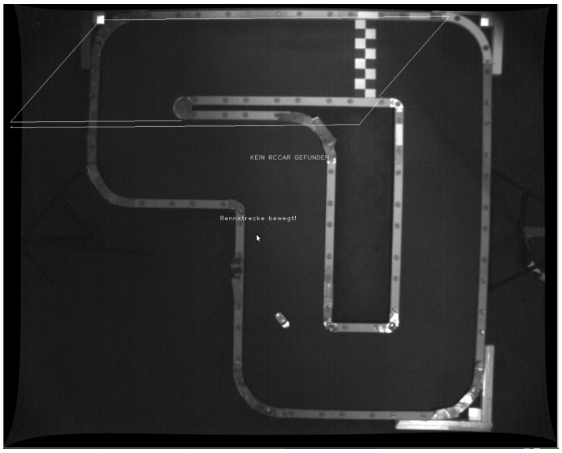
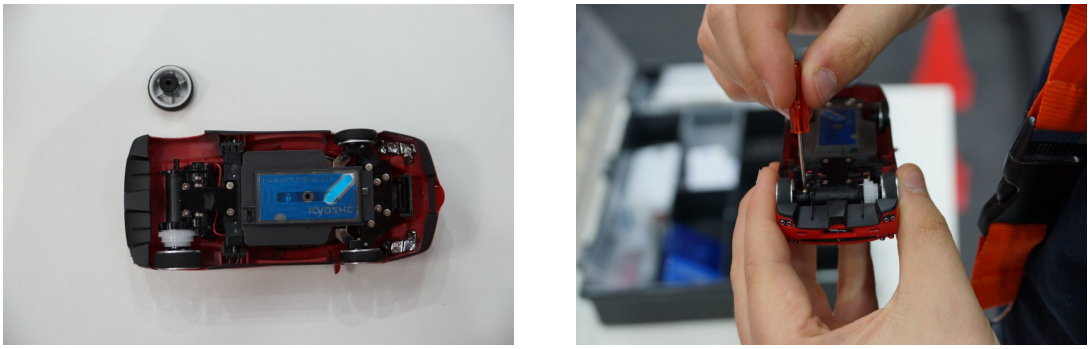


Abbildung 3.9.: Bildausgabe aufgenommen um 15:20 Uhr.

Während der Ausstellung sind mehrere Probleme aufgetreten. Das größte Problem war die Sonneneinstrahlung (siehe Abbildung 3.8 und Abbildung 3.9) durch die Fenster direkt über dem Exponat. Die Sonneneinstrahlung hat die Rennstrecke teilweise zu hell erleuchtet. Der verwendete Infrarotfilter konnte, zusammen mit den Softwarefiltern, diese abschnittsweise starke Erleuchtung nicht kompensieren. Somit wurde für bestimmte Tageszeiten der autonome Rennbetrieb eingestellt. Daraufhin haben sich die Standbetreuer überlegt, wie sie diese Sonnenstrahlen aus dem Bild filtern oder reflektieren können. Dies ist in Abbildung 3.10 zu sehen.
Des Weiteren sind kleine Probleme mit der Mechanik (siehe Abbildung 3.11(a)) der Fahrzeuge aufgetreten. Diese wurden dann von einem Projektgruppenmitglied repariert (siehe Abbildung 3.11(b)) und somit wieder in einen gebrauchsfähigen Zustand gebracht.

Abbildung 3.10.: Abdeckung der Rennstreckenbegrenzung mit grauem Gewebeklebeband.



(a) Fahrzeug mit abgefallenen Reifen

(b) Fahrzeug bei der Reperatur.

Abbildung 3.11.: Wartung von defekten Fahrzeugen.

Da die Ausstellung vom *OFFIS* zusammen mit der Projektgruppe *RCCARSng* betreut wurde, kamen zu dem normalen Messebetrieb noch besondere Termine, bezüglich der Öffentlichkeitsarbeit des *OFFIS*', hinzu.
Am Mittwoch, den 14. Juni 2017, kamen überraschend Vertreter der *SalzgitterAG*. Diese haben sich für den aktuellen Stand der Technik bezüglich des autonomen Fahrens interessiert.
Am Donnerstag, den 15. Juni 2017, kam das Innovationsnetzwerk (siehe Abbildung 3.12).
Am Samstag, den 17. Juni 2017, kamen Vertreter aus der Politik. Diese waren unter anderem Bundeswirtschaftsministerin Brigitte Zypries und der niedersächsische Ministerpräsident Stephan Weil (siehe Abbildung 3.13). Zudem war Dr. Volker Schmidt als Vertreter der IdeenExpo Teil der Delegation. Um die Delegation zu empfangen,

Abbildung 3.12.: Delegation des Innovationsnetzwerks im Gespräch mit dem PG- Betreuer Willem Hagemann.

kam der 1. Vorsitzende des *OFFIS* - Vorstandes, Prof. Dr. Wolfgang Nebel mit zum Ausstellungsstand.

Abbildung 3.13.: Personen vl.: Dr. Volker Schmidt, Stephan Weil, Brigitte Zypries und Prof. Dr. Wolfgang Nebel, während Dario Feiling den Aufbau erklärt.

Im Laufe der IdeenExpo wurden von der Projektgruppe *RCCARSng* Erkenntnisse über das Exponat gewonnen. Durch die einfache Steuerung mit Debugausgaben der Fahrzeuge über den *Xbox 360* - Controller wurde erkannt, dass jedes einzelne Fahrzeug anders auf die gesendeten Steuersignale reagiert. Zusätzlich konnten Erfahrungen über die Zuverlässigkeit des Systems gewonnen werden.

Am Freitag, den 16. Juni 2017, wurden von den Fahrzeugen insgesamt ca. 20 km über den gesamten Tag zurückgelegt. Diese Zahl errechnet sich aus der in Abbildung 3.14 dargestellten Rundenanzahl und der Länge der Rennstrecke von 6.42 m.

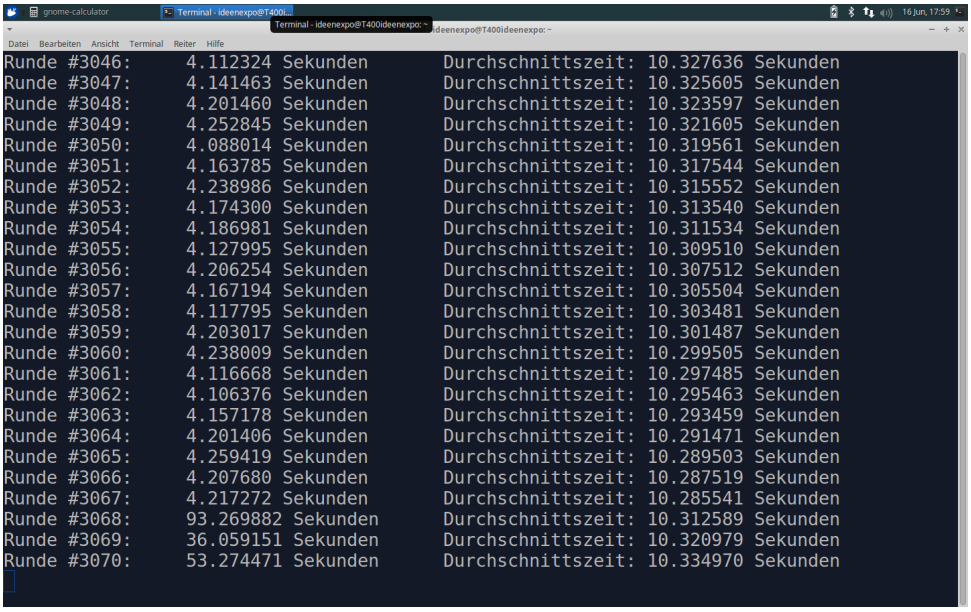


Abbildung 3.14.: Maximale Rundenzahl der Zeitmessung am Freitag, den 16.06.2017.

3.9.2. Mathefahrt

Am 05.10.2017 wurde das *OFFIS* von einer Gruppe Schülern besucht, die an einer sogenannten Mathefahrt teilnahmen. Dabei handelte es sich um Mathe begeisterte Schüler aus den Klassen 10-12. Die Projektgruppe stellte in diesem Zuge das Projekt *RCCARSng* den Schülern in mehreren Kleingruppen vor. Die Schüler wirkten offen und interessiert an dem Projekt. Ein Highlight für alle war, als jeder Schüler einmal selbst das Fahrzeug über einen *Xbox*-Controller steuern durfte. So konnte jeder einmal selbst ein Gefühl für das Auto bekommen und schnell feststellen, dass es nicht das Einfachste ist, das Fahrzeug selbst kollisionsfrei über die Strecke zu bringen. Einzelne Fragen hinsichtlich des Projektes konnten zufriedenstellend beantwortet werden. Im Gesamten war die Veranstaltung ein Erfolg und alle waren zufrieden und begeistert von der bisherigen Arbeit der Projektgruppe *RCCARSng*.

3.9.3. Ersti-Begrüßung

Am 12.10.2017 fand im *OFFIS* eine Erstsemsterbegrüßung der Bachelor und Masterstudiengänge Informatik, Wirtschaftsinformatik und Engineering of Socio-Technical Systems statt. Zu dieser Erstsemesterbegrüßung wurden den neuen Studierenden

Abbildung 3.15.: Begrüßung der Erstsemester und Vorstellung des Projektes *RCCARSng*

mehrere Projekte im *OFFIS* vorgestellt. Auch die Projektgruppe *RCCARSng* stellte dort ihr Projekt vor. In mehreren kleineren Gruppen wurde das Projekt aber auch die Möglichkeiten, die sich im Masterstudiengang bieten, den Erstsemesterstudenten nahe gebracht. Alle waren sehr interessiert und begeistert von dem Projekt. Zu den Fragen zählten neben spezifischen Fragen zum Projekt jedoch auch allgemeine Fragen zum Studium selbst. Alle aufkommenden Fragen konnten ausführlich und zufriedenstellend beantwortet werden. Die Veranstaltung war gelungen und allen Erstsemestern konnte ein guter Start in ihr Studium gewünscht werden.

3.9.4. Besuch der GIZ

Am 24.01.2018 bekam die Projektgruppe kurzfristig Besuch von einer Delegation der GIZ (Deutsche Gesellschaft für Internationale Zusammenarbeit), die im Rahmen eines Projektes mit der Abteilung *VLBA* der Carl von Ossietzky Universität Oldenburg zusammen arbeiten. Diese Delegation hat in Begleitung von Prof. Dr. Martin Fränzle und Prof. Dr. Jorge Marx Gómez auch das *OFFIS* besucht. Auch hier wurde der aktuelle Entwicklungsstand vorgeführt und das weitere Vorgehen erläutert.

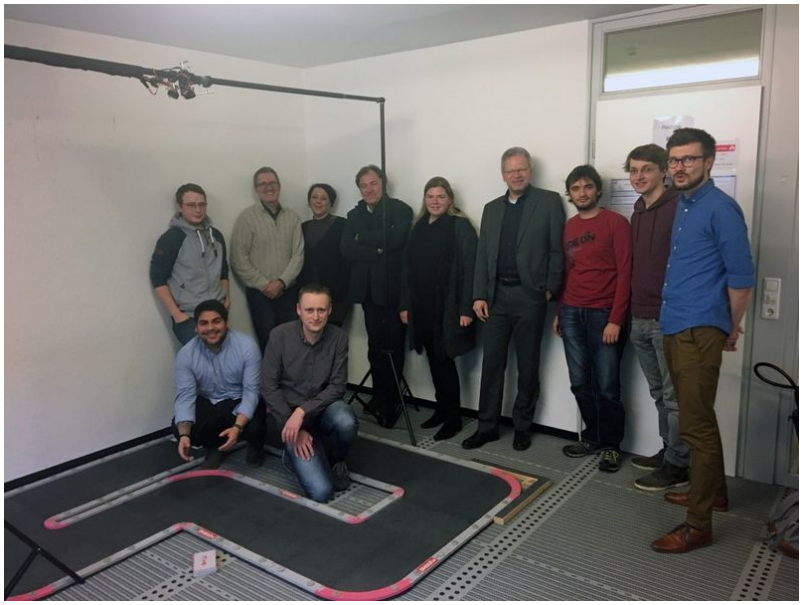


Abbildung 3.16.: Delegation des GIZ.

3.9.5. Hochschulinformationstag

Am 26.01.2018 fand ein Schülerinformationstag an der Carl von Ossietzky Universität in Oldenburg statt. An diesem Tag besuchten Schüler mehrerer unterschiedlicher Schulen die Universität und konnten einen Einblick in die Studiengänge Informatik und Wirtschaftsinformatik erhalten. Auch die Projektgruppe *RCCARSng* stellte im Laufe des Schülerinformationstages ihr Projekt vor. Die Schüler waren sehr begeistert von dem autonomen Fahrzeug und der Demonstration durch die Projektgruppenmitgliedern. Während der Präsentation konnten die Schüler fragen stellen zu dem Projekt, aber auch zum allgemeinen Studium. So konnte vielleicht auch der ein oder andere zu einem späteren Studium in einer dieser Studiengänge überzeugt werden.

Abbildung 3.17.: Vorstellung des Projektes *RCCARSng* auf dem Hochschulinformationstag

3.9.6. Wirtschaftsjunioren

Am 19.04.2018 fand ein Schülertag am *OFFIS* in Oldenburg statt. An diesem Tag besuchten Schüler der 9. Klasse des Neuen Gymnasiums Oldenburg im Rahmen der Wirtschaftsjunioren das *OFFIS* und bekamen einen Einblick in die verschiedenen Projekte. Auch die Projektgruppe *RCCARSng* stellte im Laufe des Tages ihr Projekt vor. Dazu wurden die Schüler in Kleingruppen durch das *OFFIS* geführt und konnten die ausgestellten Projekte betrachten und konnten Fragen rund um das jeweilige Projekt stellen. Die Schüler waren sehr begeistert von der Arbeit der Projektgruppe und stellten auch die ein oder andere spezifischere Frage. Im Gesamten war auch diese Veranstaltung der Projektgruppe ein Erfolg.

3.9.7. Website

Die Projektgruppe richtete eine Website für das Projekt *RCCARSng* ein. Da es bereits eine bestehende Website des Projekts *RCCARS* gab, konnte auf dieser Vorlage aufgebaut werden. Da die Website des Vorgängerprojektes jedoch mit ihrem gesamten Inhalt bestehen bleiben sollte, wurde sie mit allen Unterpunkten und Dateien in die neue Website als Vorgängerversion integriert. Dazu musste die Website zunächst umstrukturiert werden. Der grundlegende Aufbau der vorherigen Website wurde jedoch übernommen, um eine einheitliche Darstellung zu gewährleisten. Um von der jetzigen Website auf die ehemalige Website und ihre Daten zugreifen zu können, kann der Menüpunkt „Projektgruppe RCCARS 2015/16“ ausgewählt werden. Um

Abbildung 3.18.: Besuch der Wirtschaftsjunioren (Klasse 9d des NGO)

zur aktuellen Website zu gelangen, genügt ein Klick auf den Menüpunkt „Zurück zu RCCARS-ng“. Die aktuellen Menüpunkte entsprechen der Auswahl der vorherigen Website. So werden zunächst alle Teammitglieder kurz mit ihrer Funktion aufgeführt. Es wird ein Einblick in die Aufgabenstellung und das zu bearbeitende Szenario geliefert. Auch einzelne Veranstaltungen, an denen die Projektgruppe teilnahm, wie beispielsweise die IdeenExpo 2017 in Hannover, sind auf der Homepage dargestellt. Einzelne Dokumente sind ebenfalls als Download hinterlegt. Um Interessierte dauerhaft auf dem Laufenden zu halten, wurde ein Blog in die Website integriert, der wöchentlich aktualisiert wird und so einen kleinen Einblick in die Arbeit der Projektgruppe geben soll. Die vorherige Projektgruppe wurde in die Referenzen eingepflegt.
Um sich als Projektgruppe *RCCARSng* gut präsentieren zu können, wurde das alte Logo der Projektgruppe *RCCARS* erweitert und in leicht verändertem Stil ein „NG“ eingepflegt. Dadurch wird ein Aufbau auf dem vorgehenden Projekt, aber auch eine Erweiterung dessen, verdeutlicht. Das Logo ist in Abbildung 3.19 dargestellt.

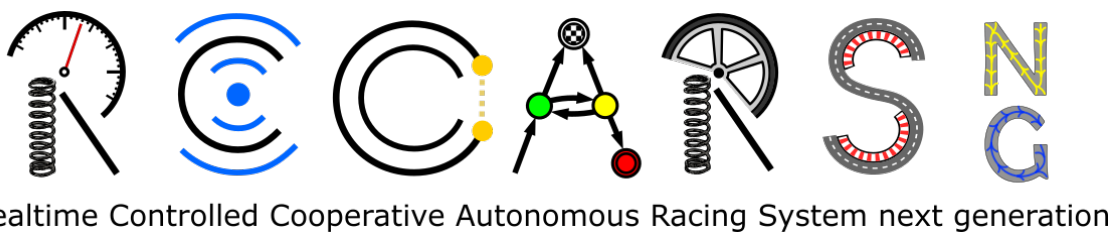


Abbildung 3.19.: Aktuelles Logo der Projektgruppe *RCCARSng*.

Die Website der Projektgruppe kann über die Adresse: https://www.uni-oldenburg.de/rccars/ aufgerufen werden. Eine weitere einfache Möglichkeit bietet die Verwendung des in Abbildung 3.20 dargestellten QR-Codes. Dieser kann eingescannt werden, um die Homepage aufzurufen.

Abbildung 3.20.: QR-Code zur Website der Projektgruppe *RCCARSng*.

4. Grundlagen

Dieses Kapitel dient einer thematischen Einführung in das Themenfeld „Autonomous Racing“. Dazu werden Projekte anderer Universitäten und Abschlussarbeiten vorgestellt, die sich bereits autonomen Rennsystemen gewidmet haben. Diese Projekte sollen hinsichtlich der von *RCCARSng* gesetzten Ziele vorgestellt werden. Zum Abschluss dieses Kapitels soll auf den $A*$-Algorithmus eingegangen werden, den die Projektgruppe ebenfalls verwendet hat.

4.1. Ausblick zu anderen Universitäten

Zur Realisierung des Szenarios der Projektgruppe *RCCARSng* wurden die Arbeiten von anderen Universitäten studiert und deren Ergebnisse als Ansatzpunkt für die weitere Projektarbeit verwendet. Im Folgenden werden diese Ergebnisse vorgestellt.

4.1.1. Eidgenössische Technische Hochschule Zürich

An der ETH Zürich finden schon seit langem Arbeiten im Bereich des „Autonomous Racing“ statt und so sind dort bereits zahlreiche Informationen bezüglich Überholmanövern oder dem Umfahren von Hindernissen zu finden.
In [LDM15] von Alexander Liniger, Alexander Domahidi und Manfred Morari werden zwei verschiedene Controller vorgestellt, welche die autonome Steuerung eines Fahrzeugs und das Umfahren von Hindernissen ermöglichen. Hierfür wird das Problem in ein Planungsproblem und ein Regelungsproblem zerlegt. Der erste Controller wird als „Hierarchical Receding Horizon Controller“ (HRHC) bezeichnet. Bei seiner Verwendung werden für das Fahrzeug verschiedene Trajektorien generiert und der Controller wählt diejenige aus, die den größten Fortschritt bezogen auf die Rennstrecke bietet (Abbildung 4.1). Die ausgewählte Trajektorie wird zur Zieltrajektorie, auf der das Fahrzeug fahren soll.

Der zweite Controller wird als „Model Predictive Contouring Control“ (MPCC) bezeichnet. Hierbei wird die Bewegung des Fahrzeugs anhand einer Referenztrajektorie geregelt. In [LDM15] ist diese Referenztrajektorie die Mittellinie der Rennstrecke.

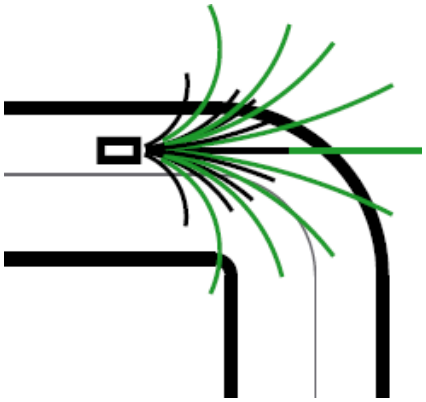


Abbildung 4.1.: Beispielhafte Generierung von Trajektorien für ein Fahrzeug [LDM15].

Anhand dieser Mittellinie wird der Fortschritt des Fahrzeugs im Bezug auf die Rennstrecke gemessen, wobei der Controller die Freiheit besitzt, von dieser geplanten Trajektorie abzuweichen, um Trajektorien zu finden, die kürzer oder schneller zu befahren sind. Der Vorteil dieses Ansatzes ist, dass sowohl das Planen der zu fahrende Route als auch die Regelung des Fahrzeugs in ein einzelnes Optimierungsproblem gefasst werden können.
Beide Ansätze ermöglichen das Umfahren von Hindernissen, indem bei der Generierung von Trajektorien die Hindernisse als Constraints hinzugefügt werden.

4.1.2. Universität Mailand

Auch an der Universität Mailand finden Arbeiten im Bereich des „Autonomous Racing“ statt. Insbesondere werden dabei Algorithmen zur Hindernisumfahrung in den Blick genommen. Ein solcher Algorithmus wird in [Fon14] vorgestellt.
In einem ersten Schritt wird hier bei erkannten Hindernissen überprüft, ob diese als eine Erweiterung der Rennstreckenbegrenzung betrachtet werden können. Falls dies der Fall ist, werden diese Hindernisse mit einer Seite der Rennstreckenbegrenzung verbunden, sodass auch der potentiell freie Platz zwischen Hindernis und der Seitenbegrenzung als Hindernis angesehen wird. Dieser Schritt vereinfacht die Routenberechnung für die Fahrzeuge, da die Hindernisse nur an der Seite umfahren werden können, die nicht mit der Wand verbunden ist.
In einem zweiten Schritt werden sogenannte „Possible Threats“ betrachtet, also solche Hindernisse, die sich in einem festgelegten Höchstabstand vom Fahrzeug entfernt befinden. Zur Erfassung der Hindernisse wird vor das Fahrzeug ein Raster gelegt, auf dem die Hindernisse markiert werden. Über die Minimierung einer Kostenfunktion, zur deren Berechnung der Lenkwinkel, die Länge der Strecke und die Abweichung zur ursprünglichen Trajektorie herangezogen werden, wird ein neuer optimaler Pfad

ermittelt. Dieser Pfad wird zur Zieltrajektorie des Fahrzeugs und anschließend vom Fahrzeug verfolgt. Ein Ablaufdiagramm dieses Algorithmus zur Hindernisumfahrung kann Abbildung 4.2 entnommen werden.

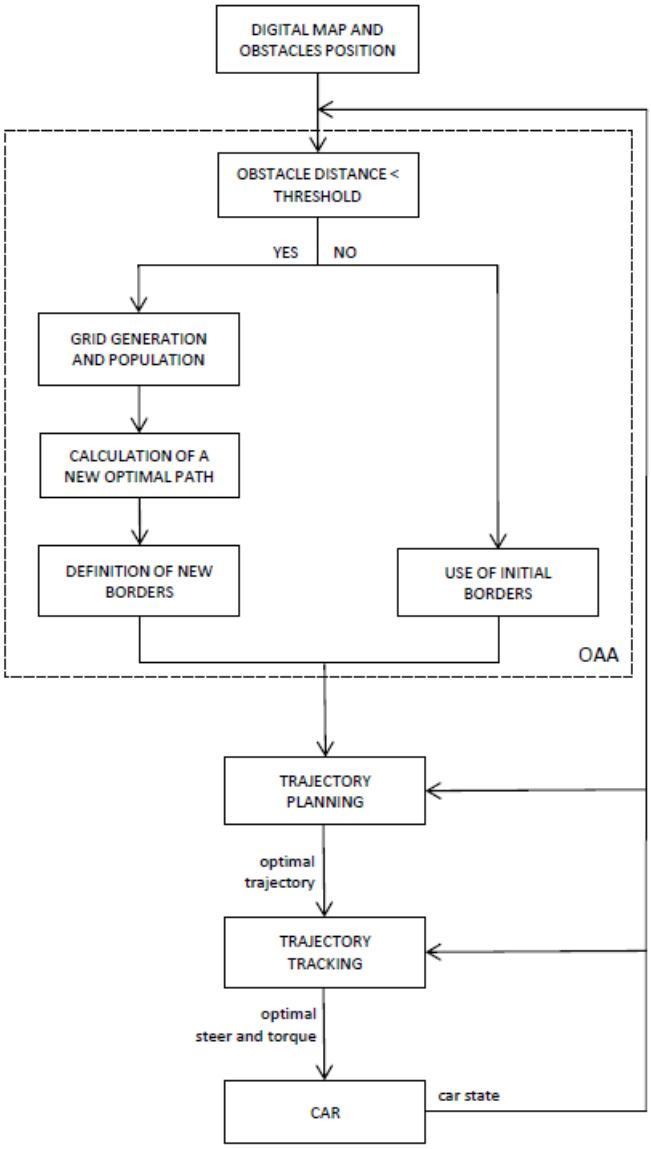


Abbildung 4.2.: Algorithmus zur Hindernisumfahrung nach [Fon14].

Berechnung einer krümmungsminimierten Trajektorie

In Mailand wurde von den Wissenschaftlern F. Braghin, F. Cheli, S. Melzi und E. Sabbioni eine Methode vorgestellt, mit der eine Trajektorie mit minimierter Krümmung berechnet werden kann. Diese ist im Artikel „Race driver Model“ [BCMS08] vorgestellt und sieht vor, die Rennstrecken entlang der Mittellinie in n Stützstellen (p_{x_i}, p_{y_i}) einzuteilen. Diese sollen zwar senkrecht zum Verlauf der Mittellinie ver-

schoben werden, können jedoch durch die Fahrbahnbreite eingeschränkt sein. Zur Trajektorienberechnung werden die Stützstellen nach dem Verschieben durch kubische Splines interpoliert. In Abbildung 4.3 sind die Stützstellen samt Trajektorie dargestellt.

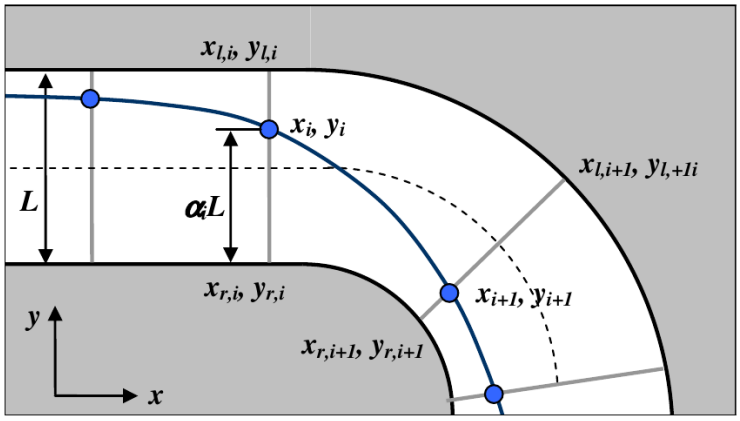


Abbildung 4.3.: Darstellung der Stützstellen für die Trajektorienberechnung. Die Stützstellen können senkrecht zur Mittellinie auf der grauen Linie bewegt werden. Die Trajektorie ergibt sich aus der Spline-Interpolation der Stützstellen [BCMS08, S. 1504, Fig. 2].

Die Trajektorie ist mit einem Parameter $s \in [0, L]$ parametrisiert, der von 0 bis zur Länge L der Trajektorie verläuft. Die für eine Interpolation genutzten Splines sind für jede Stützstelle (p_{x_i}, p_{y_i}) durch die Gleichungen

$$\begin{cases} x_i(t) & = a_{i,x} + b_{i,x} \cdot t + c_{i,x} \cdot t^2 + d_{i,x} \cdot t^3 \\ y_i(t) & = a_{i,y} + b_{i,y} \cdot t + c_{i,y} \cdot t^2 + d_{i,y} \cdot t^3 \\ t(s) & = \frac{s - s_{i0}}{ds_i} \end{cases} \tag{4.1}$$

beschrieben. Hierbei ist ds_i die Länge zwischen den zu interpolierenden Stützstellen P_i und P_{i+1} auf der Trajektorie, s_{i0} der Parameterwert für die erste Stützstelle. Somit gibt t für jedes s einen Parameterwert zwischen 0 und 1 an. $t = 0$ entspricht dabei der ersten Stützstelle P_i, $t = 1$ der zweiten P_{i+1}.

Für die Optimierung wird der in Formel 4.2 beschriebene Krümmungsbegriff verwendet.

$$\hat{\Gamma}^2 = \left(\frac{\mathrm{d}^2 x(s)}{\mathrm{d}s^2}\right)^2 + \left(\frac{\mathrm{d}^2 y(s)}{\mathrm{d}s^2}\right)^2 \tag{4.2}$$

Dieser Krümmungsbegriff kann zum in Formel 4.3 aufgeführten Ausdruck zusammengefasst werden.

$$\hat{\Gamma}^2 = \left(\frac{\mathrm{d}t(s)}{\mathrm{d}t}\right)^4 \left[\left(\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}\right)^2 + \left(\frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2}\right)^2\right]. \tag{4.3}$$

Um den Ausdruck in Gleichung 4.3 vereinfachen zu können, gehen die Autoren nun davon aus, dass die Länge der Splines zwischen den Stützstellen mit l äquidistant ist. Dann ist $\frac{\mathrm{d}t(s)}{\mathrm{d}t} = \frac{1}{l}$ und Gleichung 4.3 ergibt sich zu:

$$\hat{\Gamma}^2 = \left(\frac{1}{l}\right)^4 \left[\left(\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}\right)^2 + \left(\frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2}\right)^2\right] \tag{4.4}$$

Nach Gleichung 4.4 ist die Krümmung nur noch von Ableitungen der Splines nach t abhängig. Um die Krümmung zu minimieren, werden die Ableitungen aller Splines an der Stelle $t = 0$ zwischen allen Stützstellen aufaddiert und nach dieser Summe minimiert. Da der Faktor $\frac{1}{l}$ für jedes Segment konstant ist, wird er bei der Minimierung nicht berücksichtigt. Daher wird über die folgende Summe minimiert:

$$\sum_{i=1}^{n} \left[(\ddot{x}_i(0))^2 + (\ddot{y}_i(0))^2\right]. \tag{4.5}$$

Diesen Algorithmus hat die Projektgruppe *RCCARSng* in der Programmiersprache Matlab umgesetzt. Die Implementierung ist in Abschnitt 11.3.5 beschrieben.

4.1.3. Universität Uppsala

Auch an der Universität Uppsala wurde ein Projekt im Bereich „Autonomous Racing“ aufgebaut, in dem Fahrzeuge autonom eine Rennstrecke bewältigen. Das in [NES+16] beschriebene Projekt beinhaltet die Möglichkeit, sowohl statische Hindernisse, als auch ein sich langsam bewegendes Hindernis zu umfahren. Besonders interessant ist hierbei die Entscheidung der Projektbeteiligten einen PID-Regler zur Regelung der Fahrzeuge zu benutzen. Zusätzlich werden Splines verwendet, um geeignete Trajektorien zur Hindernisumfahrung zu finden. Auf die Nutzung von „Model Predictive Control“ (MPC) wurde verzichtet, da dieser Ansatz ein exaktes Fahrzeugmodell voraussetzt, was auch Faktoren wie das Rutschen der Fahrzeuge, Staub auf der Fahrbahn oder Ähnlichem einschließt. Deswegen wurde ein Ansatz gewählt, der nicht durch ein eventuell zu ungenaues Fahrzeugmodell beeinflusst werden kann.
Die Berechnung der zu fahrenden Trajektorie findet sowohl statisch vor Beginn des

Rennbetriebs, als auch dynamisch während des Rennbetrieb s statt. Für die statische Berechnung werden zeitlich abhängige Variablen zur Position und Geschwindigkeit des Fahrzeugs aufgestellt, die sowohl durch eine maximale Beschleunigung als auch der Geometrie der Rennstrecke eingeschränkt sind. Mithilfe einer elaborierten Trajektorie und eines JModelicaSolver [jmo] entsteht aus den Positionsvariablen x, y eine initiale zu verfolgende Trajektorie.
Für die online stattfindende Trajektorienberechnung nutzen die Entwickler der Universität Uppala ein Punktraster, auf dem die Rennstreckenbegrenzung, sowie statische und dynamische Hindernisse markiert sind. Da sich die dynamischen Hindernisse bewegen können, wird die Position der Hindernisse auf dem Raster laufend aktualisiert. Auf dem Rastermaß ist auch die initiale Trajektorie eingetragen. Kreuzt diese Trajektorie ein Hindernis, wird der nächste Punkt P, der sich sowohl auf der Trajektorie als auch auf dem Hindernis befindet, senkrecht zur Trajektorie verschoben, bis dieser außerhalb des Hindernisses liegt. Mithilfe von Splines wird eine neue Trajektorie berechnet, die durch diesen Punkt P verläuft. Falls die neu berechnete Trajektorie immer noch ein Hindernis kreuzt, wird P weiter verschoben und eine neue Trajektorie berechnet, bis keine Kollision mehr auftritt (siehe Abbildung 4.4).

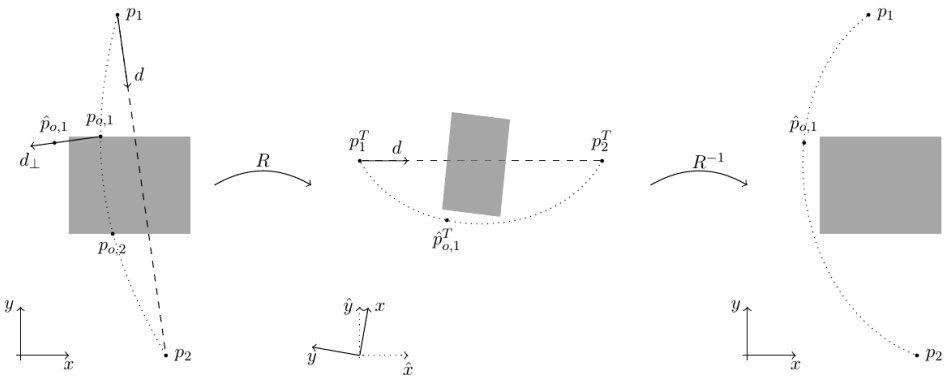


Abbildung 4.4.: Neuberechnung der Trajektorie: Punkt $p_{0,1}$ wird zunächst senkrecht verschoben, anschließend wird der Abschnitt gedreht, eine neue Trajektorie berechnet und wieder in die ursprüngliche Ausrichtung gesetzt [NES+16, S. 56, Fig. 5.7].

Die neue Trajektorie bildet die Referenztrajektorie, anhand der sich das Fahrzeug bewegen soll. Ein PID-Regler sorgt dafür, dass das Fahrzeug auf dieser Trajektorie fährt. Abbildung 4.5 stellt den Unterschied zwischen der initialen und der neu berechneten Trajektorie bildlich dar.

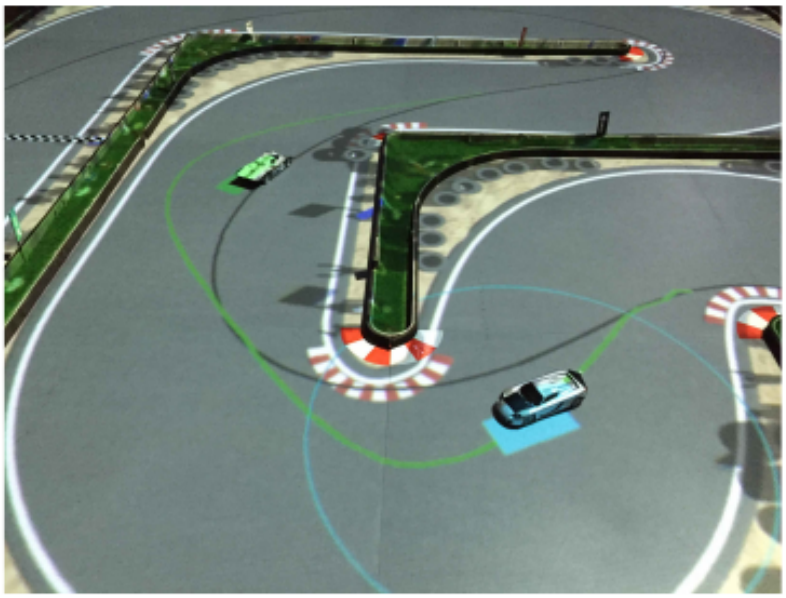


Abbildung 4.5.: Zu sehen ist das silberne Fahrzeug über dem blauen Rechteck, welches der schwarz markierten Trajektorie folgt. Da das grüne Fahrzeug sich jedoch auf eben dieser Trajektorie befindet, wurde die grüne Trajektorie als neue Referenztrajektorie berechnet [NES+16, S. 62, Fig. 5.11].

4.2. A*- Algorithmus

$A*$ (gesprochen A-Stern) ist ein Wegfindungsalgorithmus für gewichtete Graphen, der auf dem Dijkstraalgorithmus basiert. Dieser Algorithmus findet immer die optimale Lösung bezüglich der ausgewählten Kostenfunktion. Durch diese Kostenfunktion gehört der Algorithmus zur Klasse der informierten Suchalgorithmen. Die Kostenfunktion hat die Form:

$$min(g(n) + h(n)) \tag{4.6}$$

In $g(n)$ sind die Kosten des bisherigen Weges beschrieben und in $h(n)$ ist eine Heuristik definiert, die den zukünftigen Weg approximieren soll.

Funktionsweise $A*$ findet immer einen Weg durch den Graphen, sofern einer existiert. Alle Knoten im Graphen sind in einem von drei Zuständen.

Unbekannte Knoten sind Knoten, die noch nicht gefunden wurden. Es existiert also noch kein Weg zu diesen Knoten.

Bekannte Knoten sind Knoten, zu denen ein Weg gefunden wurde.

Untersuchte Knoten sind Knoten zu denen ein optimaler Weg gefunden wurde.

Bekannte Knoten werden in der sogenannten *OpenList* gespeichert. Untersuchte Knoten werden in der *ClosedList* gespeichert. Zu Beginn des Algorithmus ist nur der Startpunkt in der *OpenList* hinterlegt. Die *ClosedList* ist zu diesem Zeitpunkt leer. Während des Algorithmus wird der Knoten mit den minimalen aktuellen Kosten ($g(h)$) betrachtet. Zu diesem Knoten werden alle anliegenden Knoten betrachtet und deren Kosten aktualisiert. Ein Knoten wird dann zur *ClosedList*, wenn alle Nachbarknoten betrachtet wurden. Sobald ein Weg zum Zielknoten gefunden wurde, wird dieser Weg als optimaler Weg festgelegt, da immer der Knoten mit den aktuell minimalen Kosten betrachte wurde.
Im Worst-Case muss der gesamte Graph durchlaufen werden.

4.3. Mehrdimensionale Wahrscheinlichkeitsverteilung

Mehrdimensionale oder auch multivariate Wahrscheinlichkeitsverteilung ist die Generalisierung der eindimensionalen Wahrscheinlichkeitsverteilung. Ein Beispiel einer eindimensionalen Wahrscheinlichkeitsverteilung ist die Normalverteilung. Diese lässt sich über den Erwartungswert μ und die Varianz σ^2 beschreiben. μ ist in diesem Fall der Mittelwert aller aufgetretenen Ereignisse (siehe Formel 4.7), während σ^2 die Streuung (siehe Formel 4.8) um diesen Wert angibt.

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.7}$$

$$\sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_n)^2 \tag{4.8}$$

Die Berechnung der entsprechenden Wahrscheinlichkeiten geschieht über eine Dichtefunktion, welche im eindimensionalen Fall wie folgt aussieht:

$$p = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{(x-\mu)^2}{2\sigma^2})} \tag{4.9}$$

Diese Konstrukte lassen sich auf mehrere Dimensionen erweitern. Der Erwartungswert wird durch die Erweiterung zum Vektor der Erwartungswerte ($\vec{\mu}$), wobei die Einträge dieses Vektors die Erwartungswerte in den einzelnen Dimensionen sind. Die Varianz σ^2 wird zur Kovarianzmatrix Σ, welche die Varianz der Erwartungswerte untereinander beschreibt. Für eine n-dimensionale Wahrscheinlichkeitsverteilung

hat der Erwartungswertvektor die Dimension $n \times 1$ und die Kovarianzmatrix die Dimension $n \times n$. Die Dichtefunktion wird nun durch Formel 4.10 beschrieben.

$$p = \frac{1}{\sqrt{(2\pi^n * det(\Sigma))}} e^{(-\frac{1}{2}((\vec{x}-\vec{\mu})^T \Sigma^{-1}(\vec{x}-\vec{\mu})))} \tag{4.10}$$

Wenn in einem laufenden System neue Messungen gemacht werden, müssen diese dem Erwartungswert hinzugefügt und die Abweichung muss angepasst werden. Für das Hinzufügen einer Messung im eindimensionalen Fall gibt es folgende Berechnungsvorschrift:

$$\mu_{n+1} = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i = \frac{n}{n+1}(\frac{1}{n} \sum_{i=1}^{n} x_i + \frac{1}{n} x_{n+1}) \tag{4.11}$$

Mit

$$K = \frac{1}{n+1}$$

kann dies zu 4.12 vereinfacht werden.

$$\mu_{n+1} = \mu_n + K(x_{n+1} - \mu_n) \tag{4.12}$$

Das Aktualisieren der Varianz geschieht über Formel 4.13 [Roj03].

$$\sigma^2_{n+1} = (1-K)\sigma^2_n + K(x_{n+1} - \mu_n)^2 \tag{4.13}$$

Diese Berechnungsvorschriften lassen sich ebenfalls in den mehrdimensionalen Fall erweitern. Formel 4.12 wird um die entsprechenden Dimensionen erweitert und wird somit zu Formel 4.14. Die mehrdimensionale Erweiterung der Formel 4.13 ist vergleichbar einfach. Hier muss auf die Quadrierung der Vektoren geachtet werden (siehe Formel 4.15).

$$\vec{\mu}_{n+1} = \vec{\mu}_n + K(\vec{x}_{n+1} - \vec{\mu}_n) \tag{4.14}$$

$$\Sigma_{n+1} = (1-K)\Sigma_n + K(\vec{x}_{n+1} - \vec{\mu}_n)(\vec{x}_{n+1} - \vec{\mu}_n)^T \tag{4.15}$$

4.4. Masterarbeiten am System *RCCARS*

Im Sommersemester 2017 entwickelten zwei Mitglieder der Projektgruppe *RCCARS* das Rennsystem in Form einer Masterarbeit weiter. Diese Arbeiten konnten die Studierenden im August 2017 abschließen und sollen hier kurz vorgestellt werden.

4.4.1. Modellprädikativer Regelungsansatz

Die bisher verwendete Steuerungsvariante im vorhandenen System nutzt für die Regelung des Fahrzeugs ausschließlich seine aktuelle Position. Der Studierende *Tom Reske* aus der Projektgruppe *RCCARS* schrieb im Sommersemester 2017 eine Masterarbeit mit dem Titel „Entwicklung eines modellprädikativen Regelungsansatzes (Model Predictive Control - MPC) für ein echtzeitkontrolliertes autonomes Rennsystem“, um damit einen neuen Regelungsansatz umzusetzen. Hierfür verwendete er ein lineares Einspurmodell, welches die Dynamik des Fahrzeugs abbildet. Diese wird anhand der Ableitungen der Fahrzeugkoordinaten, des Ausrichtungswinkels sowie der Geschwindigkeit beschrieben.
Darauf aufbauend entwickelte der Student an einer prädikativen Regelung, die auf den Fahrzeugmodellformeln basiert und somit in der Lage ist, das zukünftige Verhalten des Fahrzeugs vorherzusagen. Auf Basis dieser Prädiktion soll die MPC die zum aktuellen Zeitpunkt optimale Steuergrößen berechnen. Diese setzen sich aus der Beschleunigung und dem Lenkwinkel zusammen. Durch die Verwendung einer modellprädikativen Regelung wurde eine Erhöhung der Regelgüte im Vergleich zum aktuell verwendeten PID-Regler erzielt. Somit sollte eine Verwendung von unterschiedlichen Rennstreckenverläufen möglich sein.
Nach Abschluss der Arbeit konnte eine Simulation der modellprädikative Regelung in Matlab [Mat] fertig gestellt werden.
Besonders für das Umfahren von Hindernissen ist eine Veränderung der Zieltrajektorien und eine stabile Verfolgung dieser Trajektorien unumgänglich. Daher ist die Simulation von der Projektgruppe *RCCARSng* als Basis für die Fahrzeugregelung verwendet und weiter entwickelt worden.

4.4.2. Modulare Systemarchitektur für das Fahrzeug

Im bestehenden *RCCARS* System erfolgt die Übermittlung der Regelungswerte an das Fahrzeug über ein Entwicklungsboard, das über DA-Wandler mit der Fernsteuerung für das verwendete Fahrzeug verbunden ist. Hier tritt das Problem auf, dass die Reaktionszeit des Fahrzeugs auf Regelungswerte zwischen 4 ms und 12 ms

schwankt. Der Student *Michael Bukowski* befasste sich daher in seiner Masterarbeit mit der Aufgabe, diese Zeit zu stabilisieren und versuchte dies durch die Inbetriebnahme einer eigenen Fahrzeugplatine mit programmierbarem Mikrocontroller und einer Bluetooth Schnittstelle realisieren. Sowohl in den Arbeitsgruppen der ETH-Zürich als auch der Universität Uppsala hat sich die Verwendung einer selbst entwickelten Fahrzeugplatine bewährt, weshalb sie auch in die Long-Term Vision (siehe Abbildung 2.2) des *RCCARS*-Projekts mit aufgenommen wurde. Mit seiner Masterarbeit realisierte Michael Bukowski daher Teile der dritten Ausbaustufe der Long-Term Vision (siehe Abschnitt 2.2).
Die Verwendung einer eigenen Fahrzeugplatine bietet den Vorteil, dass das Fahrzeug nicht mehr als BlackBox betrachtet werden muss. Es wäre möglich speziell entwickelte Software zu betreiben und somit das interne Verhalten des Fahrzeugs, wie beispielsweise die verzögerte Reaktion auf Regelungswerte, genauer zu analysieren und zu beeinflussen. Zusätzlich kann die Kommunikation mit dem Fahrzeug erweitert werden. Bisher ist es nur möglich, Daten an das Fahrzeug zu senden, einen Rückkanal gibt es allerdings nicht. Für die neue Platine kann zusätzlich ein Kommunikationsprotokoll entwickelt werden, das eine bidirektionale Kommunikation mit dem Fahrzeugs ermöglicht. Dadurch kann zum Einen die Sensorik, die auf der Fahrzeugplatine verbaut ist, ausgelesen werden, zum Anderen ist eine Auslagerung von Funktionen zum Fahrzeug hin möglich. Weitere Aspekte sind die Entwicklung der Systemarchitektur und des Kommunikationsprotokolls, sowie eine alternative händische Steuerung des Fahrzeugs (beispielsweise mit einem Bluetooth-Controller).
Für das Projekt *RCCARSng* ist die Entwicklung eines Fahrzeuges, dessen Verhalten genauer vorhersehbar ist und schneller auf Regelungswerte reagieren kann, von besonderem Interesse. Des Weiteren kann die verbaute Hardware zu Evaluationszwecken verwendet werden. Zum Einen soll ein sicherheitskritisches System entwickelt werden, zum Anderen wurde sich das Ziel der Implementierung von Überholvorgängen gesetzt, welches eine verlässliche Reaktionszeit des Fahrzeugs auf Regelungswerte voraussetzt. Eine Realisierung der gesetzten Ziele wäre für die Projektgruppe *RCCARSng* eine Unterstützung hinsichtlich Stabilität.

5. Systemanalyse RCCARS

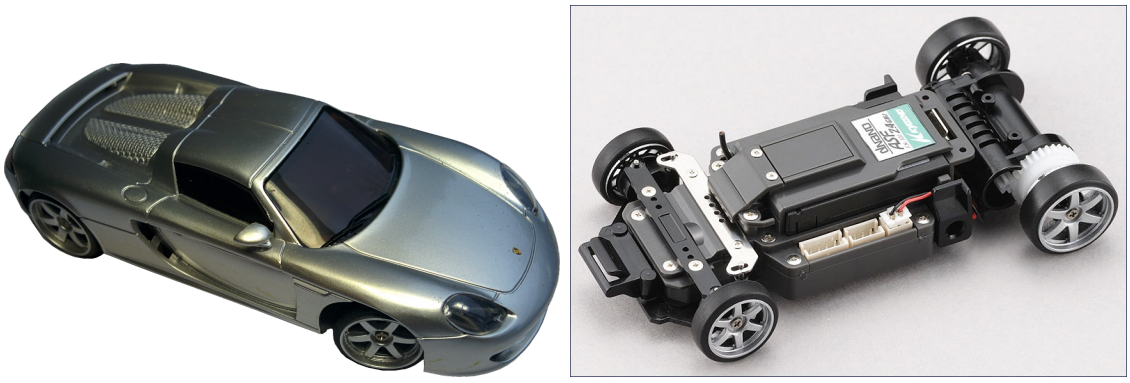
Die Projektgruppe *RCCARS* hat das gesamte System in verschiedene Subsysteme unterteilt. Diese Aufteilung wird auch in diesem Kapitel beibehalten. Das System ist modular aufgebaut, damit eine Änderung in einem Subsystem möglichst wenig Einfluss auf die anderen Bestandteile des Systems hat. Die Fahrzeuge sind Bestandteile des Subsystem 1 (Abschnitt 5.1), die Positionsbestimmung mit all ihren Komponenten bildet das Subsystem 2 (Abschnitt 5.2). Die Steuereinheit des Fahrzeugs, die Control-Unit, wird in Abschnitt 5.3 und die Rennstrecke mit ihren Bestandteilen abschließend in Abschnitt 5.4 beschrieben.

5.1. Subsystem 1: Fahrzeug

Das Subsystem 1 besteht aus mehreren ferngesteuerten Fahrzeugen im Maßstab 1:43. Diese werden von der Positionsbestimmung erfasst und mit Hilfe einer Regelungssoftware auf der Rennstrecke gesteuert.

5.1.1. Fahrzeug

Die Fahrzeuge sind vom Typ *dNaNo FX-101* der Marke *Kyosho*. Ein Fahrzeug ist in Abbildung 5.1(a) zu sehen. Die Karosserie der Fahrzeuge ist ca. 10.5 cm lang und 4.6 cm breit. Diese Fahrzeuge bestehen aus verschiedenen austauschbaren Einzelteilen. Zu diesen Einzelteilen gehören unter anderem die Reifen, der Akku, die Aufhängung und der Motor (siehe Abbildung 5.1(b)). Die Reifen und der Motor sind Verschleißteile und müssen des Öfteren gewartet und gewechselt werden. Die Karosserien der Fahrzeuge sind mit eindeutigen Reflexionsmarkern ausgestattet (siehe Abbildung 5.2). Die Positionsbestimmung erkennt anhand dieser Reflexionsmarker die Position und Ausrichtung des Fahrzeugs (Pose).



(a) Fahrzeug mit Karosserie (b) Fahrzeug ohne Karosserie

Abbildung 5.1.: Fahrzeug des *dNaNo FX-101* Modells

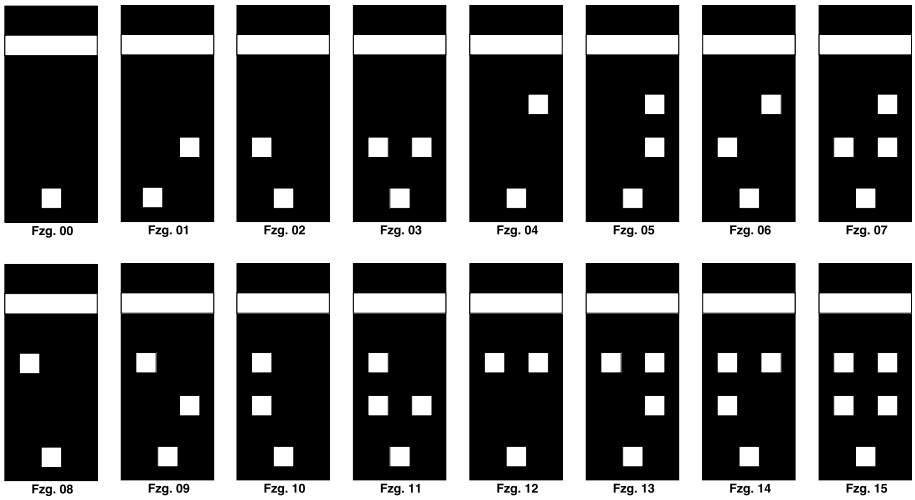


Abbildung 5.2.: Fahrzeugmarker

Reflexionsfolie Die verwendete Reflexionsfolie ist genau auf die in Abschnitt 5.2.1 beschriebene Hardware abgestimmt. Diese Folie reflektiert das ausgestrahlte IR-Licht mit genau dem gleichen Winkel, mit dem es auf die Folie trifft. Durch diese Reflexion ist gewährleistet, dass die Kamera das Licht in der richtigen Intensität erfassen kann.

5.1.2. Sicherheit

Das Fahrzeug ist ein sicherheitskritisches Element des gesamten Systems. Daher muss der Zustand des Fahrzeugs vor jedem Starten auf Fehler überprüft werden. Insbesondere muss ein Administrator die Verschleißteile kontrollieren. Hierzu gehört das Reinigen der Aufhängung von Staub und Reifenabrieb und das Tauschen des

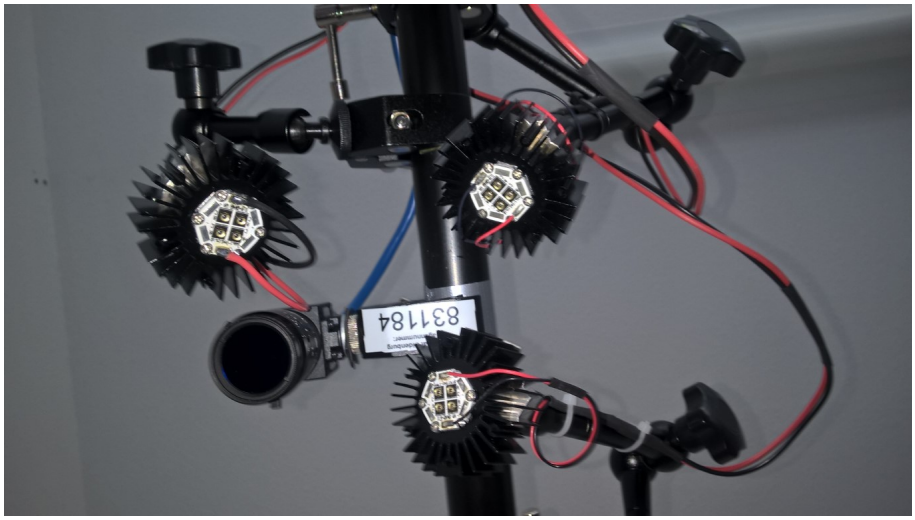


Abbildung 5.3.: Kamera mit IR-Filter und Weitwinkelobjektiv mit den IR Strahlern.

Akkus. Die Reflexionsmarker müssen fest auf dem Dach der Karosserie angebracht und groß genug sein, dass sie auf dem Bild der Kamera noch gut zu erkennen sind.

5.2. Subsystem 2: Positionsbestimmung

Das Subsystem 2 befasst sich mit der Bilderfassung und -verarbeitung. In diese beiden Bereiche fallen das Einlesen der Bilder und die Ermittlung der Pose der Fahrzeuge. Die Positionsbestimmung unterteilt sich dabei in Software- und Hardwarekomponenten. Zur Hardware gehören ein Beleuchtungssystem, eine Kamera und ein Rechner, auf dem die Software ausgeführt wird. Zu dem Beleuchtungssystem gehören Infrarot-Strahler mit dem dazugehörigen Kühlkörper und Netzteil. Zur Kamera gehören ein Objektiv und ein Infrarotfilter. Das Beleuchtungssystem und die Kamera sind an einem Gestell befestigt. Die Hardwarebestandteile werden in Abschnitt 5.2.1 detaillierter beschrieben und sind in Abbildung 5.3 dargestellt. Zur Software gehören die Bibliotheken und Treiber der Kamera zum Einlesen der Bilder. Zudem wird eine für die Bildverarbeitung und Posenbestimmung selbst entwickelte Software verwendet (siehe hierzu Abschnitt 5.2.2).

5.2.1. Hardware

Infrarotstrahler Die Pose des Fahrzeugs auf der Rennstrecke wird über die Reflexionsmarker auf der Karosserie von der Kamera berechnet. Die Projektgruppe *RCCARS* hat verschiedene Referenzprojekte studiert und sich daraufhin für ein Beleuchtungssystem aus Infrarotstrahlern entschieden. Hier wurde der Strahler *OL-*

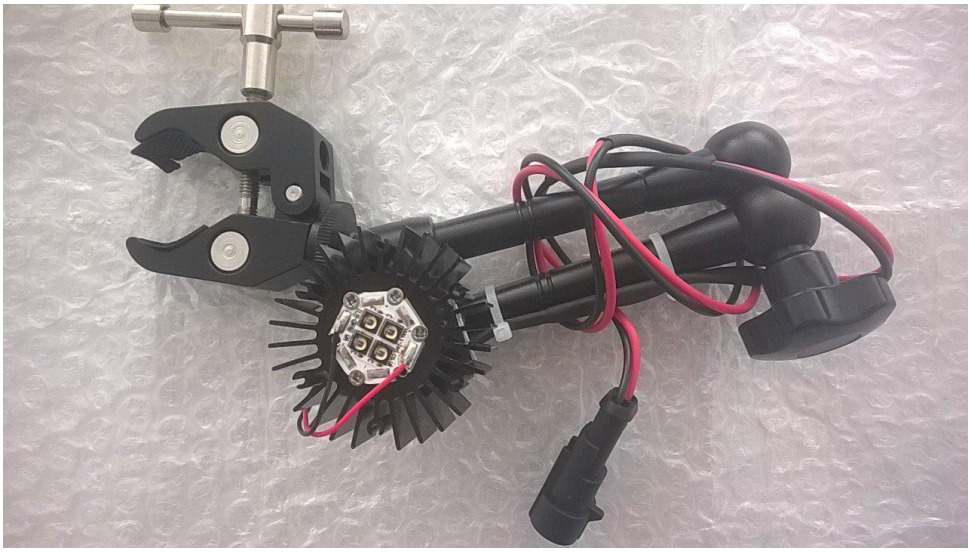


Abbildung 5.4.: Infrarotstrahler mit Kühlkörper und Befestigungsarm.

SON 4 IR Wide PowerStar (siehe Abbildung 5.3), welcher auch von der Universität Göteborg, für ein ähnliches Setting, verwendet wird, ausgewählt. Dieser Strahler besteht aus vier Infrarot-LEDs. Diese werden über eine Leiterplatine angesteuert. Die LEDs strahlen Licht mit einer Wellenlänge von 850 nm ab. Die Infrarotstrahler sind auf einem Kühlkörper angebracht (siehe Abbildung 5.4). Die Spannungsversorgung der Strahler wird über ein Netzteil geregelt. Des Weiteren muss das Beleuchtungssystem so nah wie möglich an der Kamera angebracht werden, da die Reflexionsfolie das Licht in dieselbe Richtung reflektiert. Um die Ausleuchtungsprobleme mit der Bildverarbeitung zu reduzieren, wurden mehrere Strahler verwendet. Diese wurden, wie die Kamera und die anderen IR-Strahler, am selben Gestell befestigt.

Infrarotfilter Um das von der Reflexionsfolie auf den Fahrzeugen reflektierte Licht vom Tageslicht zu trennen wurde der Infrarotfilter *MIDOPT FIL LP780/27* verwendet. Dieser Filter lässt Licht mit einer Wellenlänge größer als 780 nm passieren. Somit ist der Filter ideal für dieses Anwendungsgebiet.

Objektiv Um die geringe Raumhöhe zu kompensieren, wurde ein Weitwinkelobjektiv verwendet. Hierzu wurde das Objektiv *Lensagon CMFA0420ND* angeschafft. Dieses hat einen horizontalen Erfassungswinkel von 75.14 ° und einen vertikalen Erfassungswinkel von 59.96 °. Diese beiden Werte sind für die Berechnung der Mindesthöhe des Gestells ausschlaggebend.

Abbildung 5.5.: Kamerasystem bestehend aus Kamera, Weitwinkelobjektiv und Infrarotfilter.

Kamera Zur Erfassung des Bildes wurde die Kamera *Point Grey Flea FL3-U3-13y3m-C* (siehe Abbildung 5.5) verwendet. Diese Kamera benutzt eine USB 3.0 Schnittstelle zum Computer. Alle Referenzprojekte verwenden die gleiche Kamera. Die Kamera weißt eine hohe Bildrate von maximal 150 Hz und eine Auflösung von 1280×1024 Pixel auf. Die hohe Bildrate ermöglicht zusammen mit der hohen Auflösung eine präzise Erkennung der Fahrzeugposen, wodurch die Arbeit mit schnellen Fahrzeugen möglich ist. An der Kamera wird der Infrarotfilter und das Objektiv befestigt.

Gestell Das Kamerasystem muss zusammen mit den Infrarotstrahlern zentral über der Rennstrecke befestigt werden. Hierzu wird ein Gestell verwendet. Die erforderliche Anbauhöhe der Kamera lässt sich aus der Breite der Rennstrecke und dem vertikalen Erfassungswinkel des Weitwinkelobjektivs errechnen. Die Projektgruppe *RCCARS* hat berechnet, dass die Kamera in $\approx$2.05 m Höhe angebracht werden muss, um die komplette Rennstrecke erkennen zu können [RCC16].

Rechner Da die Bildverarbeitung ein rechenintensiver Prozess ist, wird ein Rechner benötigt, der die gewünschten Aufgaben in einer möglichst minimalen Zeit berechnet. Der verwendete Rechner ist das Modell *OptiPlex 7020MT* von *Dell*. Dieser ist mit einem *i7-4790* Prozessor von *Intel*, mehreren USB 3.0 Schnittstellen und 16 GB Arbeitsspeicher ausgestattet. Der Prozessor unterstützt Hyper-Threading, was für die erste Ausbaustufe der Long-Term-Vison wichtig ist, da auf dem gleichen Rechner noch die Regelung der Fahrzeuge ausgeführt wird (siehe Abschnitt 5.3.1).

5.2.2. Software

Die Positionsbestimmungssoftware ist in der Lage ein aufgenommenes Kamerabild zu empfangen, zu verarbeiten, auszuwerten und weiterzuleiten, bevor ein neues Bild

von der Kamera eintrifft. Diese Berechnungen finden, ausgehend von der maximalen Bildrate von 150 Hz, in weniger als 6.7 ms statt. Für das Einlesen der Bilddaten wird die *Point Grey FlyCapture* Library [FIIS] in Verbindung mit *OpenCV* [teab] verwendet. In beiden Bibliotheken sind bereits viele Algorithmen zur Bildverarbeitung implementiert. Beide Bibliotheken verfügen über C++Interfaces. Demnach ist die Positionsbestimmung in derselben Sprache implementiert.

Das Weitwinkelobjektiv, welches auf der Kamera montiert ist, verzerrt das Bild minimal. Daher muss die Positionsbestimmung diese Verzerrung zuerst herausrechnen um die Objekterkennung zu vereinfachen. Mit der Objekterkennung wird dann die Fahrzeugpose ermittelt, zudem werden weitere Störeinflüsse heraus gefiltert. Die ermittelte Fahrzeugpose beinhaltet die zweidimensionalen Fahrzeugkoordinaten und die Ausrichtung des Fahrzeuges. Diese werden über die Anordnung der Reflexionsfolie (siehe Abschnitt 5.1.1) erkannt. Zudem ist die Software fähig, die eindeutige Identität der Fahrzeuge zu erkennen.

Neben den Fahrzeugen erfasst die Positionsbestimmung die Lage der Rennstrecke. Die Koordinaten der Fahrzeugpose werden in relative Rennstreckenkoordinaten transformiert. Diese werden an die Control-Unit übermittelt. Durch die ständige Ermittlung der Position der Rennstrecke erkennt das System, ob die Rennstrecke während des Betriebes verschoben wurde.

5.2.3. Sicherheit

Das Fahrzeug fährt mit einer hohen Durchschnittsgeschwindigkeit von 1.2 m/s. Daher muss die Position mit einer entsprechend hohen Frequenz erfasst werden. Um die Fahrzeugpose zuverlässig und präzise zu erkennen, wurde eine Mindestbildwiederholrate von 100 Hz realisiert. Dies ist ein Kompromiss zwischen Rechenaufwand und der Präzision der Erkennung. Aus dieser Einschränkung leitet sich ab, dass zwischen dem Empfang und der Verarbeitung eines Bildes nicht mehr als 10 ms vergehen dürfen. Um sicherzustellen, dass diese Anforderungen eingehalten werden, ist die Software laufzeiteffizient implementiert und wird auf einem leistungsstarken Rechner ausgeführt. Das auf dem in Abschnitt 5.2.1 beschriebenen Rechner ausgeführte Betriebssystem ist ein *Ubuntu 14.04 LTS* System mit dem Realzeitkernel *3.12.31-rt45l*. Durch den Einsatz einer Grauwert-Kamera mit einem Infrarotfilter wurde die Laufzeit der Algorithmen deutlich verringert, da nur der Unterschied zwischen hellen und dunklen Bereichen überprüft werden muss. Die Positionsbestimmungssoftware erkennt die Lage der Rennstrecke. Sobald entweder die Kamera oder die Rennstrecke bewegt worden sind, sendet die Positionsbestimmungssoftware einen Fehler. Dieser

Fehler führt dazu, dass das System in einen kontrollierten Fehlerzustand überführt wird und das Fahrzeug anhält.

5.3. Subsystem 3: Control-Unit

Die Control-Unit unterteilt sich in Soft- und Hardwarekomponenten. Zur Software gehört die Regelungssoftware für die autonome Steuerung der Fahrzeuge. Die Hardware unterteilt sich in die CarControl, die für die Signalverarbeitung und die Weiterleitung an das Fahrzeug zuständig ist, und einen leistungstarken Rechner, auf dem die Berechnung der Regelungswerte stattfindet. Die Berechnung erfolgt aufgrund der ermittelten Pose des Fahrzeugs von der Positionsbestimmung. Die berechneten Regelungswerte werden über die CarControl an das Fahrzeug übermittelt. Durch die Regelungswerte werden Geschwindigkeit und Lenkung des Fahrzeugs bestimmt. Die CarControl empfängt Fahrzeugdatenpakete, sowie Fehlercodes der Positionsbestimmung und der Systemsteuerungssoftware. Die CarControl sendet Fernsteuerungssignale, Kontrollbefehle, Fehlercodes und Zeitdaten.

5.3.1. Hardware

Rechner Für die Berechnung der Regelungswerte wird ein leistungsstarker Rechner (siehe Abschnitt 5.2.1) benötigt, um die festgelegten Realzeitanforderungen einzuhalten. In der bisherigen Projektausführung werden Positionsbestimmung und die Berechnung der Steuerungswerte parallel auf einem Rechner ausgeführt. Die Regelungswerte werden über eine USB-Schnittstelle an die CarControl übermittelt.

CarControl Die CarControl besteht aus dem Entwicklerboard *STM32F4 Discovery* und der *Perfex KT-18* Fernbedienung der *dNaNo*-Fahrzeuge von *Kyosho*. Die Fernsteuerung ist für die Übermittlung der Steuerungsbefehle an das Fahrzeug notwendig. Die Fernbedienung besitzt zwei Potentiometer. Die Stellung der Potentiometer bestimmen die Geschwindigkeit und die Steuerung der Fahrzeuge. Beide Potentiometer arbeiten in einem Bereich von 0.00 V bis 3.00 V. Durch die anliegende Spannung ermittelt die Fernbedienung die gewünschten Werte für Beschleunigung und Lenkung und übermittelt die Regelungswerte über ein 2.4 GHz Frequenzband an das Fahrzeug. Das Fahrzeug kann sowohl manuell, über die Fernbedienung, als auch autonom, über die Regelungssoftware, gesteuert werden. Um das Fahrzeug über die Regelungssoftware steuern zu können, ist das Entwicklungsboard an den Potentiometern der Fernsteuerung angeschlossen (siehe Abbildung 5.6). Die Potentiometer sind von der Fernbedienung getrennt. Das *STM32F4 Discovery* besitzt einen integrierten

Abbildung 5.6.: CarControl: Entwicklungsboard mit befestigter Fernbedienung.

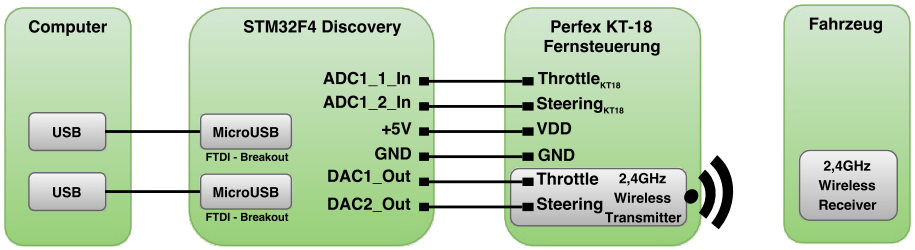


Abbildung 5.7.: Schematische Darstellung der Kommunikation zwischen CarControl, Rechner und Fahrzeug.

DA-Wandler. Das Board versorgt die Fernbedienung ebenfalls mit Spannung. Neben einer integrierten Micro-USB-Schnittstelle auf dem Entwicklungsboard, die für die Stromzufuhr notwendig ist, besitzt die CarControl zwei FTDI Basic-Breakout Schnittstellen. Diese dienen zur stabilen Kommunikation mit dem Rechner. Eine Schnittstelle ist für die Übertragung der Steuerungssignale zuständig, die andere für Debug-Ausgaben. Die Kommunikation der CarControl mit dem Fahrzeug und dem Rechner ist in Abbildung 5.7 schematisch dargestellt.

Die CarControl besitzt insgesamt vier verschiedene Betriebsmodi. Bei dem „vollständig manuellen“ Modus werden die Regelungswerte der Potentiometer aus der Fernbedienung verwendet. Die Regelungswerte des Wrapper Codes (siehe Abschnitt 5.3.2) werden dabei vernachlässigt. Im hybriden Modus „Längsführung manuell“ werden nur die Regelungsdaten für die Querführung vom Wrapper Code berücksichtigt. Die Längsführung erfolgt manuell über die Fernbedienung. Im hybriden Modus „Querführung manuell“ werden nur die Regelungsdaten der Längsführung vom Wrapper Code verwendet. Die Querführung erfolgt über die manuelle Steuerung mithilfe der Fernbedienung. Im „autonomen“ Modus werden die Werte der Fernbedienung voll-

ständig ignoriert und nur die Regelungswerte des Wrapper Codes verwendet. Um den Modus zu wechseln wird ein entsprechender Taster auf der CarControl betätigt. Dort befinden sich ebenfalls Taster für ein Not-Aus, einen Softreset und einen Hardreset. Durch verschiedene LED's, die sich auf dem Entwicklungsboard befinden, und deren Kombination wird signalisiert, in welchem Betriebsmodus sich die CarControl befindet, oder ob ein Fehlerzustand vorliegt.

5.3.2. Software

Die Software der Control-Unit erhält die aktuelle Position des Fahrzeugs, welche von der Positionsbestimmung erfasst wird. Wenn von der Positionsbestimmung in zwei aufeinander folgenden Bildern kein Fahrzeug erkannt wird, löst die Control-Unit nach spätestens 30 ms einen Not-Halt aus. Es wird zunächst eine Plausibilitätsüberprüfung durchgeführt, da es bei der Erfassung der Kamerabilder zu Fehlern kommen kann. Durch die Plausibilitätsüberprüfung wird verhindert, dass Steuersignale, die auf falschen Grunddaten basieren, übermittelt werden. Sind die Daten plausibel, wird mit der Berechnung der Regelungsdaten aufgrund von Fahrzeugposition, Ausrichtung, Geschwindigkeit und Beschleunigung des Fahrzeugs fortgefahren.
Die Trajektorie, welche das Fahrzeug fährt, ist bekannt und an den Rennstreckenverlauf angepasst. Die Software erkennt, ob das Fahrzeug den befahrbaren Bereich der Rennstrecke verlässt, mit der Fahrbahnbegrenzung kollidiert oder in die falsche Richtung fährt und löst, falls notwendig, einen Not-Halt des Fahrzeugs aus.
Die Software übermittelt die digital berechneten Regelungswerte an die CarControl. Die Regelung wird mit einer Wiederholungsrate von 100 Hz durchgeführt. Die Laufzeit wird durch eine laufzeiteffiziente Implementierung der Regelungssoftware realisiert. Durch die Verwendung von SCADE wird sichergestellt, dass die entwickelte Software sicherheitskritischen Anforderungen genügt.
Die Software unterteilt sich nochmals in den Wrapper Code und das SCADE Modell. Der Wrapper Code stellt dabei die Schnittstelle zwischen dem SCADE Modell und der CarControl dar. Diese Schnittstelle ist bidirektional. Über diese Schnittstelle werden Rennstreckendaten, Fahrzeugpose und Konfigurationsparameter an das SCADE Modell weitergegeben. Dieses berechnet daraus die notwendigen Regelungswerte und eventuelle Fehlercodes. Die aus dem SCADE Modell resultierenden Werte werden über die gleiche Schnittstelle an die CarControl gesendet.

Wrapper Code Der Wrapper Code bildet das Grundgerüst der Control-Unit. Er dient zur Kommunikation zwischen den unterschiedlichen Systemkomponenten. Dazu zählt die CarControl und das SCADE Modell. Durch den Wrapper Code wird

die Realzeitbedingung und die Plausibilität der empfangenen Pose überprüft. Er liest die Rennstreckendatei und die Konfigurationsdatei ein. Durch die Konfigurationsdatei, wird eine Initialisierung der Control-Unit durchgeführt. Um die passende Trajektorie zu finden, wird um die Koordinaten des Fahrzeuges ein Kreis gelegt, der sich mit der Zieltrajektorie in einem, zwei oder keinem Punkt schneidet. Durch eine in Längsrichtung gelegte Achse, eine Achse durch den Mittelpunkt des Fahrzeuges und dem Schnittpunkt mit der Zieltrajektorie wird im SCADE Modell der notwendige Lenkeinschlag für das Fahrzeug und die Geschwindigkeit berechnet. Der Radius des Kreises hängt von der Geschwindigkeit des Fahrzeugs ab. Er berechnet sich durch die Formel 5.1, wobei $R_{default}$ im Konfigurationsfile hinterlegt ist. Um einem zu großen oder zu kleinen Kreis vorzubeugen, sind dort ebenfalls Ober- und Untergrenze des Radius hinterlegt.

$$R = v \cdot R_{default} \tag{5.1}$$

Für die Schnittpunktfindung wird für jeden Punkt der Trajektorie überprüft ob sie im Kreis, der durch den Radius und dem Fahrzeugmittelpunkt aufgespannt wurde, liegt. Anschließend wird mit Formel 5.2 bestimmt, welcher der gefundenen Punkte die kleinste Distanz zu den Fahrzeugkoordianten aufweist. Daraufhin wird der Punkt an das SCADE Modell übergeben. Liegt kein Schnittpunkt mit der Zieltrajektorie vor, wechselt das System in einen sicheren Fehlerzustand.

$$R \leq \sqrt{(X_{Trajectory} - X)^2 + (Y_{Trajectory} - Y)^2} \tag{5.2}$$

Die Regelungswerte werden an das SCADE Modell nur übergeben, wenn sie sich von dem vorherigen Regelungswert unterscheiden. Über einen in dem Wrapper Code eingebauten Softreset, kann die CarControl zurückgesetzt werden. Hierfür wird bei Betätigung ein Flag an die CarControl gesendet.
Für die Überprüfung der Plausibilität und der Realzeitanforderung wurde ein Watchdog eingebaut, der die Fahrzeug-, Kontroll- und Fehlerdaten überwacht. Die einzelnen Funktionen im Wrapper Code werden über einzelne Threads und demzufolge mit Multithreading realisiert.

SCADE Modell Das SCADE Modell wird durch den Wrapper Code aufgerufen. Durch das SCADE Modell wird die Fahrzeugposition auf der Rennstrecke überprüft. Es wird festgestellt, ob eine Kollision mit der Fahrbahnbegrenzung vorliegt oder ob das Fahrzeug den befahrbaren Bereich verlassen hat. Die Fahrbahnbegrenzung ist

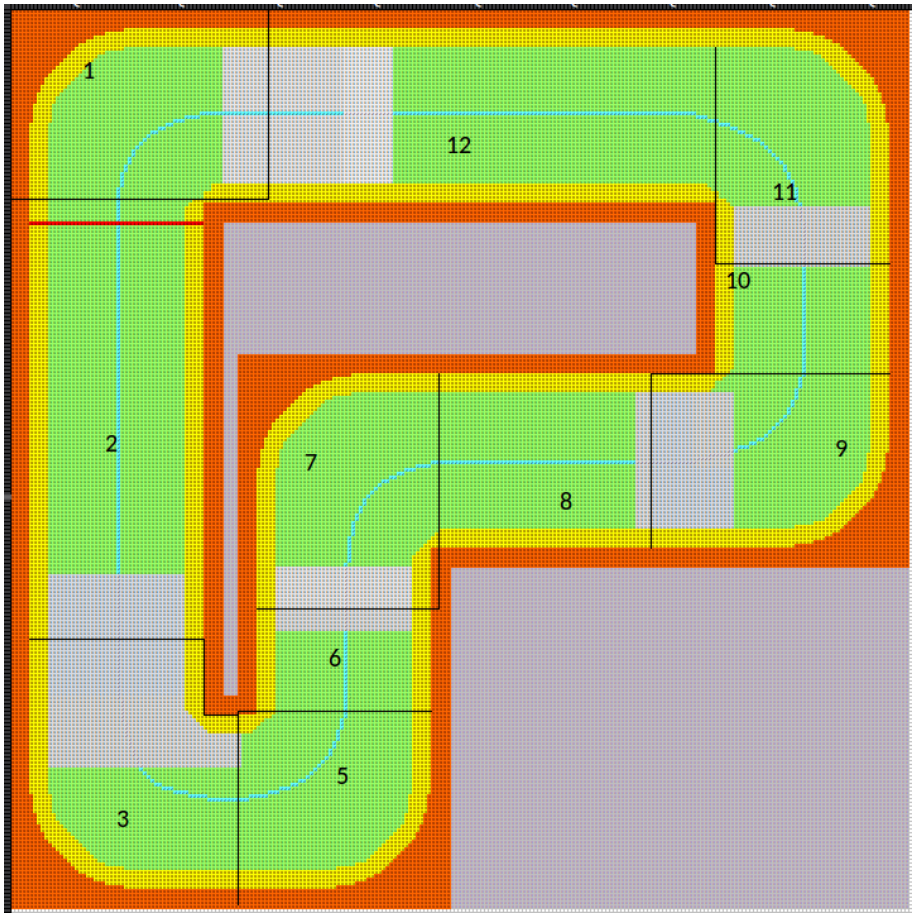


Abbildung 5.8.: Darstellung der Rennstrecke als Tabellendokument.

virtuell um 5 cm vergrößert worden, da das Fahrzeug als Punkt angesehen wird, in der Realität jedoch eine gewisse Länge besitzt. Die äußersten Kanten des Autos liegen ca. 4 cm von dem von der Control-Unit angenommenen Punkt entfernt. Zusätzlich wurde 1 cm Toleranz hinzugefügt.

Um das Fahrzeug sicher zu steuern, wird eine Rennstreckendatei (siehe Abbildung 5.8) benötigt. In ihr ist die Zieltrajektorie, der befahrbare und der nicht befahrbare Bereich dargestellt. Um festzustellen, in welchem Segment der Rennstrecke sich das Fahrzeug befindet, sind die Segmente durch Zahlen gekennzeichnet. Gerade Zahlen beschreiben dabei ein gerades Segment, ungerade beschreiben eine Kurve. Je nach Streckenabschnitt werden die aktuellen Koordinaten mit den letzten Koordinaten verglichen. So wird überprüft, ob das Fahrzeug in die richtige Richtung fährt. Diese Überprüfung findet nur auf geraden Segmenten statt.

Durch das SCADE Modell wird mithilfe von den in den Fahrzeugdatenpaketen hinterlegten Zeitstempeln, die jeweilige Geschwindigkeit berechnet. Um ein stabiles Durchfahren der Kurve zu ermöglichen, sind auf der Rennstrecke virtuelle Bremszonen eingefügt. In diesen vollzieht das Fahrzeug durch ein sofortiges Einstellen des

entsprechenden Steuerungswerts der Längsführung eine Vollbremsung. Die Bremszonen sind in der bisherigen Ausbaustufe statisch und auf die festgelegte Rennstrecke angepasst. Da das Fahrzeug eine ausreichende Geschwindigkeit benötigt, um mit Hilfe der Bremszonen um die Kurve zu gelangen, wird zunächst immer mindestens eine Einführungsrunde gefahren. So wird das Fahrzeug auf Geschwindigkeit gebracht und ein fliegender Start ermöglicht. In diesen Einführungsrunden fährt das Auto mit einer verminderten Geschwindigkeit. Sobald die in der Konfigurationsdatei eingestellten Einführungsrunden absolviert sind, wird in den Rennmodus gewechselt.

Die Längsregelung ist über eine Kennfeldsteuerung realisiert. Das Kennfeld beinhaltet Stellwerte zwischen 0.1 m/s bis 3.5 m/s in einer Abstufung von 0.1 m/s. Jeder Geschwindigkeitswert besitzt einen passenden Throttle-Stellwert, der den Gasstellwert beschreibt. In bisheriger Ausbaustufe ist das Kennfeld nur teilweise gefüllt. Die Querregelung ist ebenfalls über ein Kennfeld realisiert. Jede Kurve besitzt dabei einen evaluierten Lenkstellwert. Mithilfe eines PID-Reglers wird, sobald die Zielgeschwindigkeit durch das Kennfeld bestimmt ist, dauerhaft überprüft, ob die Zielgeschwindigkeit der aktuellen Geschwindigkeit des Fahrzeuges entspricht. Durch eine Sättigungsfunktion und eine Einflussbegrenzungsfunktion ist gewährleistet, dass der Regler den Stellwert nicht zu hoch oder zu niedrig korrigiert.

5.3.3. Sicherheit

Um eine ausreichend präzise autonome Fahrzeugsteuerung zu erreichen, wird die Regelung mit einer Wiederholrate von mindestens 100 Hz ausgeführt. Durch die Verwendung von SCADE wird sichergestellt, dass der generierte Programmcode für die Regelungsalgorithmen korrekt ist. Wird das Fahrzeug in zwei aufeinander folgenden Kamerabildern nicht erkannt, was einer Zeit von 20 ms entspricht, versetzt die Control-Unit das System in einen sicheren Fehlerzustand und bringt das Fahrzeug zum Stehen. Dies erfolgt auch, wenn das Fahrzeug danach wieder erkannt wird. Das ist notwendig, damit das Fahrzeug sich nicht unkontrolliert über die Rennstrecke bewegt. Die Control-Unit versetzt das System auch in einen Fehlerzustand, wenn die Rennstreckenmarker verdeckt werden oder wenn durch die Control-Unit festgestellt wird, dass sich das Fahrzeug rückwärts auf der Strecke bewegt.

Dies ist im Allgemeinen nicht vorgesehen, sodass sich daraus eine Kollision und somit ein Abprallen von der Fahrbahnbegrenzung schließen lässt.

5.4. Subsystem 4: Rennstrecke

Die Rennstrecke wird als eigenes Subsystem betrachtet, da sie eine essentielle Rolle bei der Projektumsetzung spielt. Bei der verwendeten Rennstrecke handelt es sich um die Rennstrecke *Mini-Z Grand Prix Circuit 30* von der Firma *Kyosho*. Die Rennstrecke besteht aus Schaumstoff und ist für die *dNaNo*-Fahrzeuge von *Kyosho* geeignet. Die Strecke besteht aus insgesamt 48 einzelnen Streckensegmenten, die beliebig zusammengebaut werden können. So kann die Rennstrecke mit nur wenigen Eingriffen komplett verändert werden. In dem Projekt *RCCARSng* wird die Rennstrecke jedoch in keinem Anwendungsfall verändert. Die Rennstrecke besitzt neben den einzelnen Streckensegmenten eine Fahrbahnbegrenzung, die den befahrbaren Bereich von dem nicht befahrbaren Bereich trennt. Dadurch wird verhindert, dass Fahrzeuge die Strecke selbständig verlassen können.

5.4.1. Streckenaufbau

Streckenführung Die Abbildung 5.9 zeigt die Rennstrecke aus der Vogelperspektive, wie sie auch von der Kamera aufgenommen wird. Die Strecke setzt sich aus insgesamt sechs 90 °-Kurven und zwei 1.2 m langen Geraden zusammen. Zwei der 90 °-Kurven sind durch ein direktes Aufeinanderfolgen zu einer 180 °-Kurve zusammengebaut. Zwischen den anderen Kurven befinden sich gerade Verbindungsstücke der Länge 30 cm, 60 cm und 60 cm. Bei der Berechnung der Mittellinie ergibt sich eine gesamte Streckenlänge von ca. 6.427 m. Die Berechnung ist in Rechnung 5.3 zu finden. Die Rennstrecke wird von den Fahrzeugen gegen den Uhrzeigersinn befahren.

$$\begin{aligned} Length_{Strecke} &= 12 \cdot 0.3\,\mathrm{m} + 6 \cdot \frac{1}{4} \cdot 2\pi \cdot 0.3\,\mathrm{m} \\ &= 3.6\,\mathrm{m} + \frac{9 \cdot \pi}{10} m \\ &\approx 6.427\,\mathrm{m} \end{aligned} \tag{5.3}$$

Segmente Für die Rennstrecke gibt es drei verschiedene Rennstreckensegmente, ein gerades Segment und zwei verschiedene für die Kurven (für das Kurveninnere und -äußere). Auf allen äußeren Streckensegmenten ist eine 1.5 cm hohe und 4.8 cm breite Schaumstoffleiste montiert, die als Fahrbahnbegrenzung dient. Durch das Material ist gewährleistet, dass die Fahrzeuge auch bei einer Kollision mit der Fahrbahnbegrenzung nicht zu Schaden kommen. Weiterhin dient die Fahrbahnbegrenzung dazu,

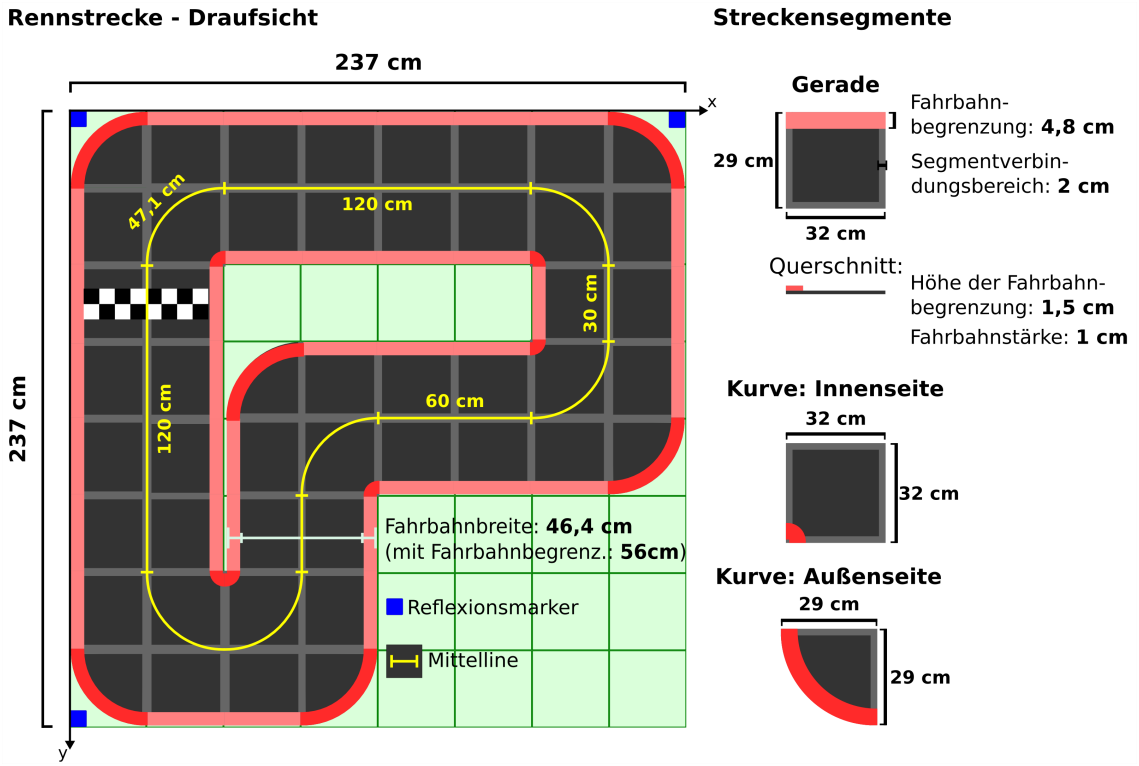


Abbildung 5.9.: Darstellung der Rennstrecke aus der Vogelperspektive mit Darstellung der Segmente und der einzelnen Maße.

dass kein Fahrzeug durch Kollision oder Softwarefehler die Rennstrecke verlassen kann. Alle geraden Segmente haben eine Länge von 32 cm und Breite von 29 cm. Die Segmente der Kurveninnenseite sind quadratisch und besitzen eine Länge und Breite von 32 cm. Die Segmente der Kurvenaußenseite haben eine Länge und Breite von 29 cm. Bei allen geraden Segmenten entfallen aufgrund der Verbindungen 2 cm. Durch die einzelnen Segmente kann die Rennstrecke in weiteren Aufbauprojekten auch in veränderter Form Verwendung finden. In dieser Ausbaustufe wird die Rennstrecke nicht anders moduliert und, wie in Abbildung 5.9 dargestellt, Verwendung finden.

Erfassung Um eine korrekte und gesamte Erfassung der Rennstrecke zu gewährleisten, muss um die gesamte Rennstrecke eine Minimum Bounding Box in einer Größe von 5.617 m^2 gelegt werden. Diese Größe kommt zustande, da die Rennstrecke mit einer Seitenlänge von 2.36 m quadratisch ist. Aufgrund der Verformbarkeit von Schaumstoff, kann es durch Deformation in Form von Stauchung oder Streckung zu einer Veränderung der Länge und Breite der Rennstrecke kommen. Diese Deformation kann eine Abweichung von bis zu 1 cm hervorrufen. Diese mögliche Abweichung wurde bei der Implementierung berücksichtigt, sodass die Seitenlänge der Minimum

Bounding Box auf 2.37 m vergrößert wurde. Dadurch ist sichergestellt, dass die gesamte Rennstrecke von der Kamera und somit der Positionsbestimmung erfasst wird.
Die Lage der Rennstrecke muss dem System ebenfalls bekannt sein, damit die Software das Fahrzeug präzise steuern kann. Zum Systemstart wird dabei eine Initialisierung durchgeführt, wobei die Lage der Rennstrecke genau erfasst wird. Um dies zu automatisieren, befinden sich drei Reflexionsmarker an den äußeren drei Kurven. Über die Rennstrecke wird ein Koordinatensystem gelegt. Dabei befindet sich der Koordinatenursprung links oben in Abbildung 5.9. Es wird in Zentimeter gemessen.

5.4.2. Sicherheit

Vor dem Rennbetrieb muss sichergestellt sein, dass sich keine Hindernisse oder störende Objekte auf der Rennstrecke oder im Erfassungsbereich der Kamera befinden. Zuschauer müssen einen Sicherheitsabstand von 1 m einhalten. Dies verhindert, dass Zuschauer mit ihrem Körper oder Teilen ihres Körpers die Positionsbestimmung beeinflussen, falls sie in den Aufnahmebereich der Kamera gelangen. Da die Rennstrecke nach der Initialisierung nicht mehr bewegt werden darf, dient dieser Sicherheitsabstand auch dazu, die Zuschauer daran zu hindern die Rennstrecke zu berühren. Der Rennbetrieb wird gestoppt, sobald eine Person einen im Kamerabild befindlichen Marker verdeckt oder die Rennstrecke bewegt. Die Fahrzeuge selbst müssen ebenfalls einen Sicherheitsabstand zur Fahrbahnbegrenzung einhalten, um keine Deformationen der Begrenzung zu verursachen. In der bisherigen Ausbaustufe des Projekts *RCCARS* musste dieses nicht berücksichtigt werden, da das Fahrzeug in der Mitte der Rennstrecke fährt und so eine Kollision mit der Fahrbahnbegrenzung bestmöglich ausgeschlossen wird. Die Fahrbahnbegrenzung umfasst die gesamte Rennstrecke. So wird sichergestellt, dass das Fahrzeug die Rennstrecke zu keiner Zeit selbstständig verlassen kann.

5.5. Netzwerk

Da in der momentanen Ausbaustufe alle Komponenten auf eigenen Rechnern laufen sollen, müssen diese miteinander kommunizieren. Dies wird durch die Verwendung des IEEE 802.3 Ethernet realisiert, da alle Rechner bereits eine eingebaute Netzwerkkarte besitzen. Alle verwendeten Rechner werden über einen Switch miteinander im Netzwerk verbunden. Durch diese Verbindung können alle Rechner und somit auch alle Komponenten (wie Control-Unit, Positionsbestimmung etc.) miteinander kom-

munizieren. Zum bisherigen Zeitpunkt wird lediglich eine Control-Unit verwendet, da nur ein Fahrzeug auf der Strecke gesteuert werden muss. Die Control-Unit und die Positionsbestimmung werden auf demselben Rechner ausgeführt. Die Kommunikation findet über UDP statt. Die Datenpakete werden unter der Benutzung von Broadcast verschickt. Dadurch müssen keine IP-Adressen bekannt sein. Durch die Übertragung als Broadcast, werden die Datenpakete an das gesamte Netzwerk gesendet. Durch die geringen Zeitabstände, in denen ein Datenpaket gesendet wird, wird das gesamte Netzwerk stark belastet.

5.6. Systemsteuerungssoftware

Das gesamte System besitzt eine Systemsteuerungssoftware, um dem Administrator ein Interface zur Überwachung und Steuerung der Softwarekomponenten der Control-Unit und der Positionsbestimmung zu bieten. Durch die Systemsteuerungssoftware werden alle Zustandswechsel, die im System auftreten, dem Administrator dargestellt. Zudem kann ein Administrator die Zustände manuell wechseln. Lediglich der automatische Not-Aus bildet hierbei eine Ausnahme. Durch das Interface werden einzelne Systemparameter, wie Fehlertoleranz oder Anzahl der Fahrzeuge, die am Rennen teilnehmen sollen, festgelegt. Die Systemsteuerungssoftware empfängt alle Netzwerkdatenpakete der Positionsbestimmung und der Control-Unit. Wird ein sicherheitskritischer Fehler erkannt, wird das System durch die Software in einen sicheren Fehlerzustand versetzt. Durch sie wird ebenfalls überprüft, ob die Realzeitanforderungen eingehalten werden. Es können alle Systeminformationen und Netzwerkdaten protokolliert werden. Alle eingehenden Fehlermeldungen werden automatisch protokolliert und gespeichert. Andere Aufzeichnungen und Protokollierungen sind optional.

5.7. Signalfluss zwischen den Subsystemen

Der Signalfluss der einzelnen Subsysteme ist in Abbildung 5.10 schematisch aufgezeigt. Diese Abbildung zeigt die einzelnen Hardwarekomponenten der jeweiligen Subsysteme. Aus dieser Darstellung wird ersichtlich, dass der Rechner zum bisherigen Zeitpunkt sowohl für die Positionsbestimmung, als auch für die Berechnung der Regelungswerte durch die Control-Unit verwendet wird. Die einzelnen Subsysteme sind durch Eingrenzungen dargestellt.
Die gesamte Rennstrecke wird durch Infrarotstrahler mit Infrarotlicht ausgeleuchtet. Hierdurch werden auch die einzelnen Fahrzeuge angestrahlt. Durch die vorhande-

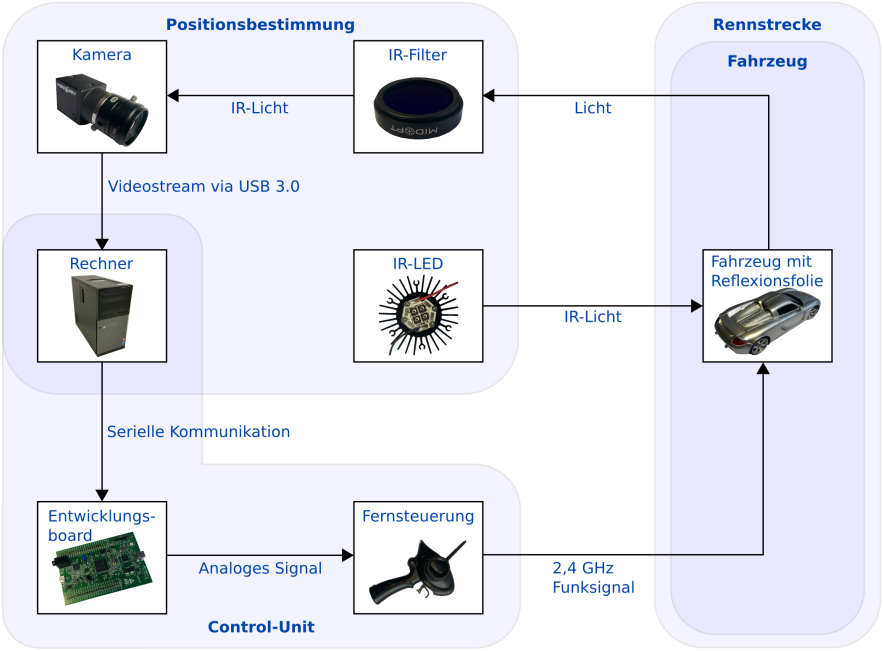


Abbildung 5.10.: Schematische Darstellung des Signalflusses des Projekts *RCCARS*.

nen Reflexionsmarker auf den Fahrzeugen und die Eckmarker wird das Infrarotlicht reflektiert. Durch einen IR-Filter wird das Infrarotlicht von anderen Reflexionen gefiltert und gelangt in die Kamera. Diese sendet die aufgenommenen Bilder über eine USB 3.0 Schnittstelle an den Rechner. Auf diesem findet zunächst die Positionsbestimmung statt. Die ermittelte Pose des Fahrzeugs wird an die Control-Unit übergeben, welche die notwendigen Regelungswerte für das Fahrzeug berechnet. Über eine serielle Schnittstelle übermittelt die Software der Control-Unit die Steuersignale an die CarControl. Dieses wandelt die digitalen Signale in analoge Signale um und übergibt sie an die Fernbedienung. Diese sendet die Steuerungssignale über ein 2.4 GHz Frequenzband an das Fahrzeug.

5.8. Notwendige Anpassungen

Aus der vorangegangenen Analyse des bestehenden *RCCARS* Aufbaus ergeben sich für die Erfüllung des in Abschnitt 2.3 beschriebenen Szenarios inklusive der beschriebenen Varianten folgende notwendige Anpassungen. Diese Anpassungen werden im Folgenden subysystemweise dargestellt.

5.8.1. Subsysteme

Subsystem 1: Fahrzeug Die Fahrzeuge müssen für das Erfüllen des Szenarios und der Varianten nicht angepasst werden. Es muss lediglich sichergestellt sein, dass sich Fahrzeuge mit unterschiedlichen IDs auf der Rennstrecke befinden. Möglicherweise ist die Verwendung der Fahrzeugarchitektur aus der Masterarbeit von Michael Bukowski (siehe Abschnitt 4.4.2) eine Option, um das Fahrzeug nicht mehr als Black-Box betrachten zu müssen. Dies ist jedoch optional.

Subsystem 2: Positionsbestimmung Um die Wartbarkeit und den Aufbau der Positionsbestimmung zu verbessern, ist es notwendig eine Halterung für die Beleuchtungseinheit zu entwerfen, welche sowohl an der Decke wie auch an dem transportablen Gestell montiert werden kann. Die Halterung muss die Möglichkeit bieten die Ausrichtung der Kamera sowie der IR Strahler einfach und zuverlässig vorzunehmen. Die Implementierung der Positionsbestimmung muss in soweit erweitert werden, dass mindestens zwei Fahrzeuge erkannt und deren Pose berechnet werden kann. Des Weiteren muss die Erkennung und Behandlung von sich auf der Rennstrecke befindlichen Hindernissen integriert werden. Durch die Erweiterungen darf die Berechnungszeit der einzelnen Datenpakete jedoch nicht verlängert werden, da ansonsten die bestehenden Realzeitanforderungen des Systems verletzt werden würden.

Subsystem 3: Control-Unit Um zwei Fahrzeuge ansteuern zu können, ist es notwendig eine zweite CarControl aufzubauen, dabei kann nach aktuellem Stand die Software von der ersten CarControl 1:1 übernommen werden. Die Software der Control-Unit wird in großen Teilen überarbeitet werden, wenn nicht sogar gänzlich ausgetauscht werden müssen, da durch das zweite Fahrzeug und die zusätzlichen Hindernisse auf der Rennstrecke die Regelung wesentlich komplexer wird. Die statische Trajektorie sowie die statischen Bremszonen erlauben keine Überholvorgänge und auch das Ausweichen ist nicht realisierbar. Aus diesem Grund wird auch die Architektur der Control-Unit überarbeitet werden müssen. Bei den Änderungen ist jedoch zu berücksichtigen, dass die Realzeitanforderungen weiterhin erfüllt und entsprechende Sicherheitsmechanismen zum Erreichen des sicheren Fehlerzustands implementiert werden. Des Weiteren gilt es zu analysieren, ob der aktuelle sichere Fehlerzustand im System mit mehreren Fahrzeugen und Hindernissen noch seine Gültigkeit besitzt.

Subsystem 4: Rennstrecke Die Rennstrecke an sich muss nicht angepasst werden, jedoch wird sie um mehrere Hindernisse erweitert. Für die Hindernisse gelten identische Anforderungen wie für die Rennstrecke, so dürfen beispielsweise die Fahrzeuge bei einer Kollision nicht beschädigt werden. Des Weiteren müssen die Hindernisse eine feste Größe besitzen und mit eindeutigen Markern versehen sein, damit die Positionsbestimmung die Positon der Hindernisse auf der Rennstrecke bestimmen kann.

Subsystem 5: Systemsteuerung Die Systemsteuerung ermöglicht in der bisherigen Version das Einstellen von Parametern und das manuelle Wechseln von Systemzuständen. Um das Szenario vollständig zu implementieren, muss die Systemsteuerung auf neue Parameter und gegebenenfalls neue Systemzustände ebenfalls angepasst werden. Über sie muss zusätzlich das Fahrverhalten, die Anzahl der Fahrzeuge und die Mindestdurchschnittsgeschwindigkeit festgelegt werden können. Um dies zu realisieren, müssen die Schnittstellen zu den Config-Files und den Subsystemen Positionsbestimmung und Control-Unit angepasst werden. Eine Anpassung der vorhandenen GUI ist ebenfalls notwendig. Des Weiteren wird die Systemsteuerungssoftware zukünftig als eigenes Subsystem aufgefasst.

5.8.2. Netzwerk

Die Netzwerkschnittstelle wird voraussichtlich von einer Broadcast Kommunikation auf eine Multicast Kommunikation umgestellt werden müssen, da durch die zweite Control-Unit und die zusätzlichen Hindernisdaten die Kommunikationslast steigen wird und diese bereits jetzt zu Leistungsspitzen bei einigen Netzwerkswitches führt. Alternativ wäre es denkbar das gesamte Setting in einem eigenen Netzwerk zu betreiben. Des Weiteren werden die zu versendenden Netzwerkpakete aufgrund der zusätzlichen Daten strukturell angepasst werden müssen.

5.8.3. Signalfluss

Durch die zusätzlichen Hindernisse sowie die weiteren Fahrzeuge im System *RCCAR-Sng* ergibt sich im Vergleich zu dem in Abschnitt 5.7 erläuterte, der in Abbildung 5.11 dargestellte erweiterte Signalfluss. Die Kommunikation zwischen den Subsystemen erfolgt explizit über die Netzwerkschnittstelle. Zudem werden die Positionsbestimmung und jede Control-Unit auf je einem dedizierten Rechner ausgeführt.

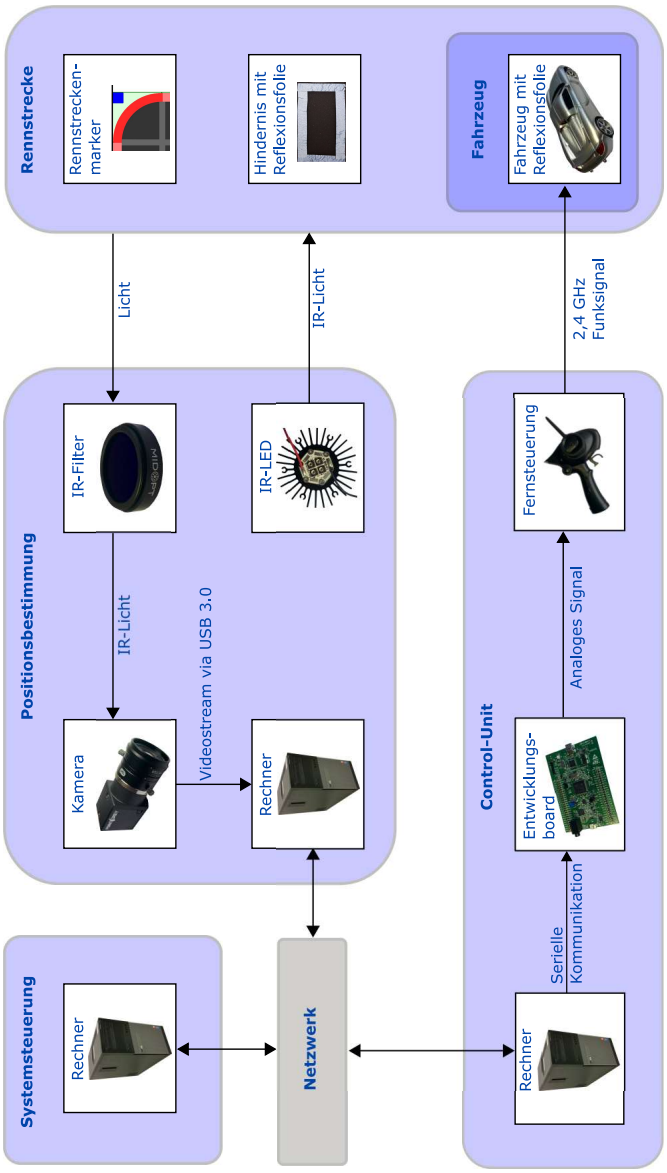


Abbildung 5.11.: Schematische Darstellung des Signalflusses zwischen den Subsystemen

6. Anwendungsfalldiagramme

In diesem Kapitel werden Anwendungsfälle definiert, welche sich direkt aus dem Szenario „unfallfreier Überholvorgang“ (siehe Abschnitt 2.3) ableiten. Im Folgenden werden diese Anwendungsfälle durch UML-Use-Case-Diagramme und Anwendungsfalltabellen erläutert.
Das Kapitel ist in zwei Abschnitte untergliedert, in denen die Anwendungsfälle des Szenarios aus der Sicht zweier unterschiedlicher Typen von Akteuren definiert werden. Unter Akteuren vom Typ A werden Personen verstanden, welche mit dem System interagieren können. Die Akteure des Typs B repräsentieren die fünf Subsysteme des Systems. Dadurch soll die Lesbarkeit verbessert werden.
Die Anwendungsfalltabellen sind alle nach dem folgenden Schema aufgebaut: Die eindeutige ID erleichtert die Referenzierung. Die Beschreibung verschafft dem Leser eine Übersicht über den jeweiligen Anwendungsfall. Akteure sind die am Anwendungsfall beteiligten Personen bzw. Systemkomponenten und der Auslöser beschreibt die Aktionen, welche nötig sind, um den Anwendungsfalls auftreten zu lassen. Die Vorbedingung gibt die Voraussetzungen an, unter denen der Anwendungsfall eintreten kann, während das Ergebnis die Situation beschreibt, welche nach einer erfolgreichen Durchführung des Anwendungsfalls vorliegt. Der Tabellenabschnitt Standardablauf verdeutlicht die Abfolge von Ereignissen, die während der Durchführung des Use-Cases auftreten. Zum Schluss werden die Anforderungen aus den Rahmenbedingungen (Abschnitt 7.1) und den generellen Systemanforderungen (Abschnitt 7.2.1) referenziert, welche für die Realisierung des jeweiligen Anwendungsfalls relevant sind. Falls die angegebenen Anforderungen bzw. Rahmenbedingungen in Unteranforderungen zerfallen, sind auch diese zu berücksichtigen, da die Hauptanforderung erst erfüllt ist, wenn alle unterliegenden Anforderungen erfüllt sind.
Die Projektgruppe *RCCARS* hat innerhalb ihres Anforderungdefinitiondokuments bereits grundlegende Anwendungsfälle beschrieben, die für die Ausführung des Systems essentiell sind. Zur Übersichtlichkeit wurden folgende Anwendungsfälle in diesem Dokument nicht beschrieben: System starten, Kamerabild einlesen, Fahrzeugpose an Control-Unit übermitteln, Automatischer Notstopp und Fahrzeugpose empfangen.

Bestehende Anwendungsfälle, die auf das neue Szenario angepasst wurden, sind in diesem Dokument aufgeführt. Zusätzlich wurden neue Anwendungsfälle anhand des neuen Szenarios und der Meilensteine erarbeitet.

6.1. Personeninteraktionen mit dem System

In diesem Kapitelabschnitt werden die Anwendungsfälle des Szenarios spezifiziert, in denen Personen mit dem System interagieren (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.

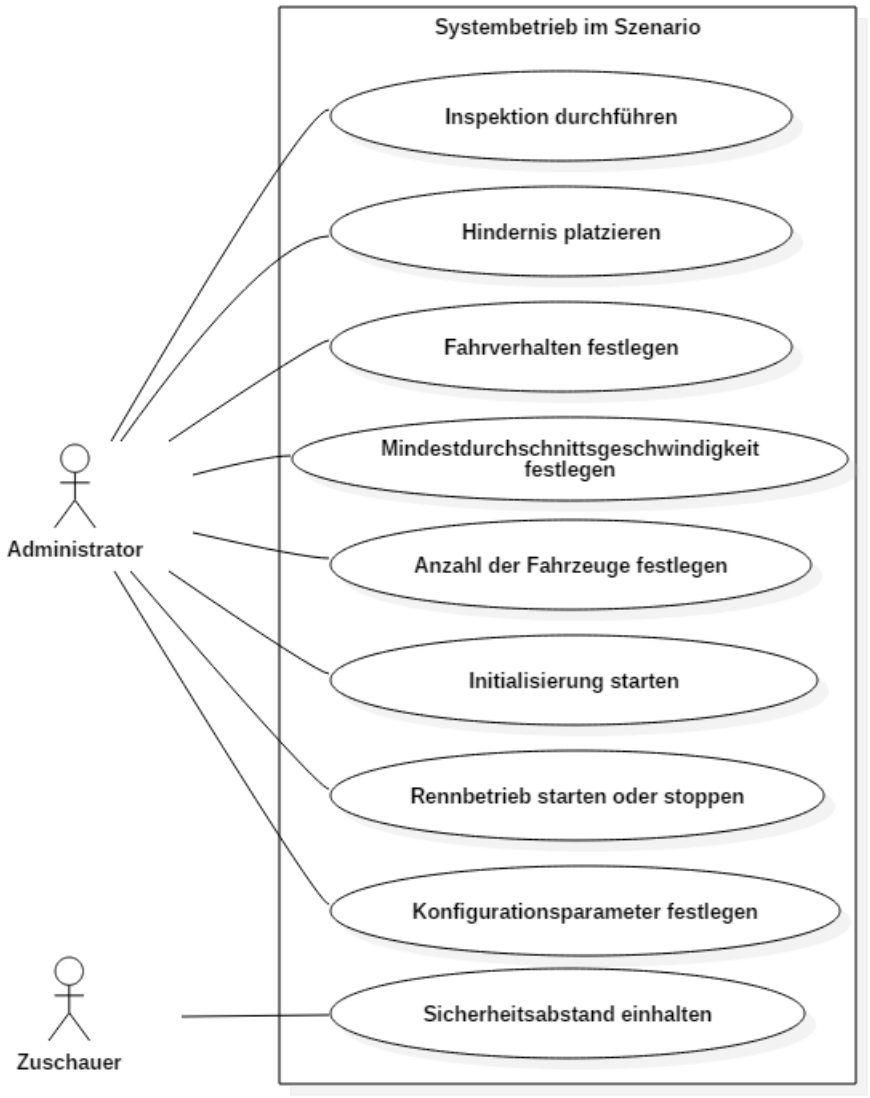


Abbildung 6.1.: Erster Teil des Anwendungsfalldiagramms für das Szenario „unfallfreier Überholvorgang“ mit Administrator und Zuschauern. Es wird dargestellt, inwieweit der Administrator und die Zuschauer in das Szenario eingreifen.

ID	A01
Beschreibung	Der Administrator führt eine Inspektion der Hardwarekomponenten durch.
Akteure	Administrator
Auslöser	Das System soll in Betrieb genommen werden.
Vorbedingungen	-
Ergebnis	Alle Hardwarekomponenten sind für den Rennbetrieb in einem geeigneten Zustand.
Standardablauf	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 der Fahrzeuge und säubert diese ggf. oder ersetzt Fahrzeugteile. Hierbei muss auch nach dem Akkustand geschaut werden. 4. Der Administrator überprüft die Hindernisse anhand der Anforderungen auf ihre Tauglichkeit. 5. Der Administrator prüft, ob alle ggf. platzierten Hindernisse auf der Strecke ordnungsgemäß aufgestellt wurden und einen ausreichenden Sicherheitsabstand besitzen.
Anforderung	Sys2.1

Tabelle 6.1.: Anwendungsfalltabelle Inspektion der Hardwarekomponenten.

ID	A02
Beschreibung	Der Administrator platziert ein statisches Hindernis auf der Rennstrecke.
Akteure	Administrator
Auslöser	Es soll eine Fahrt mit einem oder mehreren Hindernissen stattfinden.
Vorbedingungen	Das System muss sich im Zustand „Inspektion“ befinden.
Ergebnis	Das statische Hindernis liegt den Anforderungen entsprechend auf der Rennstrecke, sodass dieses von der Positionsbestimmung erkannt werden kann.
Standardablauf	1. Der Administrator platziert das Hindernis den Anforderungen entsprechend auf der Rennstrecke.
Anforderung	Rah4, HW4.4

Tabelle 6.2.: Anwendungsfalltabelle Platzierung eines Hindernisses.

ID	A03
Beschreibung	Der Administrator legt ein Fahrverhalten fest, nach dem die Fahrzeuge im Rennbetrieb fahren sollen.
Akteure	Administrator
Auslöser	Es soll festgelegt werden, wie die Fahrzeuge sich auf der Rennstrecke verhalten.
Vorbedingungen	Die Inspektion ist erfolgreich abgeschlossen.
Ergebnis	Die Fahrzeuge fahren gemäß dem vom Administrator festgelegten Fahrverhalten.
Standardablauf	1. Der Administrator legt über das dafür vorgesehene User-Interface das gewünschte Fahrverhalten fest.
Anforderung	Rah5, SW3.1.11

Tabelle 6.3.: Anwendungsfalltabelle Fahrverhalten festlegen.

ID	A04
Beschreibung	Der Adminstrator legt die Mindestdurchschnittsgeschwindigkeit der Fahrzeuge fest.
Akteure	Adminstrator
Auslöser	Es soll eine Fahrt mit einem oder mehreren Fahrzeugen stattfinden.
Vorbedingungen	Die Inspektion ist erfolgreich abgeschlossen.
Ergebnis	Die Mindestdurchschnittsgeschwindigkeit der Fahrzeuge wurde korrekt festgelegt.
Standardablauf	1. Der Adminstrator gibt die gewünschte Mindestdurchschnittsgeschwindigkeit ein.
Anforderung	SW3.1.10

Tabelle 6.4.: Anwendungsfalltabelle Mindestdurchschnittsgeschwindigkeit festlegen.

ID	A05
Beschreibung	Der Adminstrator legt die Anzahl der Fahrzeuge fest, die sich auf der Rennstrecke befinden.
Akteure	Adminstrator
Auslöser	Es soll eine Fahrt mit einem oder mehreren Fahrzeugen stattfinden.
Vorbedingungen	Die Inspektion ist erfolgreich abgeschlossen.
Ergebnis	Die Anzahl der Fahrzeuge auf der Rennstrecke wurde korrekt festgelegt.
Standardablauf	1. Der Adminstrator platziert die gewünschte Anzahl der Fahrzeuge auf der Rennstrecke. 2. Der Adminstrator gibt die gewünschte Anzahl der Fahrzeuge über das User-Interface ein.
Anforderung	SW3.1.9

Tabelle 6.5.: Anwendungsfalltabelle Fahrzeuganzahl wird festgelegt.

ID	A06
Beschreibung	Der Administrator startet die Initialisierung des Systems.
Akteure	Adminstrator
Auslöser	Das System soll auf den Rennmodus vorbereitet werden.
Vorbedingungen	Die Inspektion ist erfolgreich abgeschlossen.
Ergebnis	Alle Komponenten im System sind initialisiert.
Standardablauf	1. Der Administrator startet die Initialisierung über die Systemsteuerungssoftware. 2. Alle Komponenten melden nach Durchlauf eine erfolgreiche Initialisierung zurück.
Anforderung	SW3.1.2

Tabelle 6.6.: Anwendungsfall Initialisierung starten.

ID	A07
Beschreibung	Der Administrator startet den Rennbetrieb des Systems und kann diesen stoppen.
Akteure	Adminstrator
Auslöser	Das System soll in den Rennmodus versetzt werden oder wieder gestoppt werden.
Vorbedingungen	Initialisierung aller Komponenten erfolgreich abgeschlossen. Zum Stoppen muss das System sich im Rennbetrieb befinden.
Ergebnis	Das System befindet sich im Rennbetrieb oder ist gestoppt.
Standardablauf	1. Der Administrator startet Rennmodus über die Systemsteuerungssoftware 2. Fahrzeuge bewegen sich gemäß des vorgegebenen Fahrverhaltens. 3. Die Fahrzeuge werden angehalten.
Anforderung	SW3.1.5, SW3.1.3, Sys2.6

Tabelle 6.7.: Anwendungsfall Rennbetrieb starten.

ID	A08
Beschreibung	Der Administrator legt unterschiedliche Konfigurationsparameter für das System fest.
Akteure	Adminstrator
Auslöser	System erwartet Eingaben von Konfigurationsparametern.
Vorbedingungen	System wurde gestartet.
Ergebnis	System läuft mit vorgegeben Konfigurationsparametern.
Standardablauf	1. Der Administrator trägt in der Systemsteuerungssoftware nötige Konfigurationsparameter ein. 2. Administrator speichert die Konfigurationsparameter.
Anforderung	SW3.1.4, Sys2.5

Tabelle 6.8.: Anwendungsfall Konfigurationsparameter festlegen.

ID	A09
Beschreibung	Der Zuschauer muss den Sicherheitsabstand einhalten, um das System nicht zu stören.
Akteure	Zuschauer
Auslöser	Der Zuschauer will das System im Rennbetrieb beobachten.
Vorbedingungen	-
Ergebnis	Der Zuschauer kann das System im Rennbetrieb beobachten.
Standardablauf	1. Der Zuschauer behält einen Sicherheitsabstand ein.
Anforderung	Rah2.1, Rah2.2

Tabelle 6.9.: Anwendungsfalltabelle Sicherheitsabstand der Zuschauer.

6.2. Anwendungsfälle der Subsysteme

In diesem Kapitelabschnitt werden die Anwendungsfälle des Szenarios aus Sicht der Subsysteme spezifiziert. Der Akteur Positionsbestimmung umfasst die dazugehörigen Hard- und Softwarekomponenten, die zur Erfassung der Fahrzeug und Hindernisposen bis hin zur Weiterleitung dieser notwendig sind. Der Akteur Control-Unit erhält die von der Positionsbestimmung ermittelte Fahrzeugpose und nutzt diese, um das Fahrzeug autonom über die Rennstrecke zu steuern. Der Akteur Fahrzeug beschreibt die Interaktionen zwischen den Fahrzeugen und dem System. Zum Akteur Rennstrecke gehören zusätzlich zur Rennstrecke Hindernisse, die ggf. auf der Bahn platziert werden. Die Systemsteuerungssoftware ist der letzte Akteur, der die Kommunikationsschnittstelle zwischen System und Administrator darstellt.

ID	B01
Beschreibung	Die Positionsbestimmung erkennt die Position und Ausrichtung der Rennstrecke.
Akteure	Positionsbestimmung
Auslöser	Ein Kamerabild wurde eingelesen und verarbeitet.
Vorbedingungen	Das System befindet sich im Zustand „Initialisierung“, „Standby“ oder „Race“.
Ergebnis	Die Pose der Rennstrecke wurde ermittelt.
Standardablauf	1. Die Positionsbestimmung verarbeitet ein Bild. 2. Die Positionsbestimmung stellt die Pose der Rennstrecke fest.
Anforderung	SW1.7

Tabelle 6.10.: Anwendungsfalltabelle Erkennung der Position und Ausrichtung der Rennstrecke.

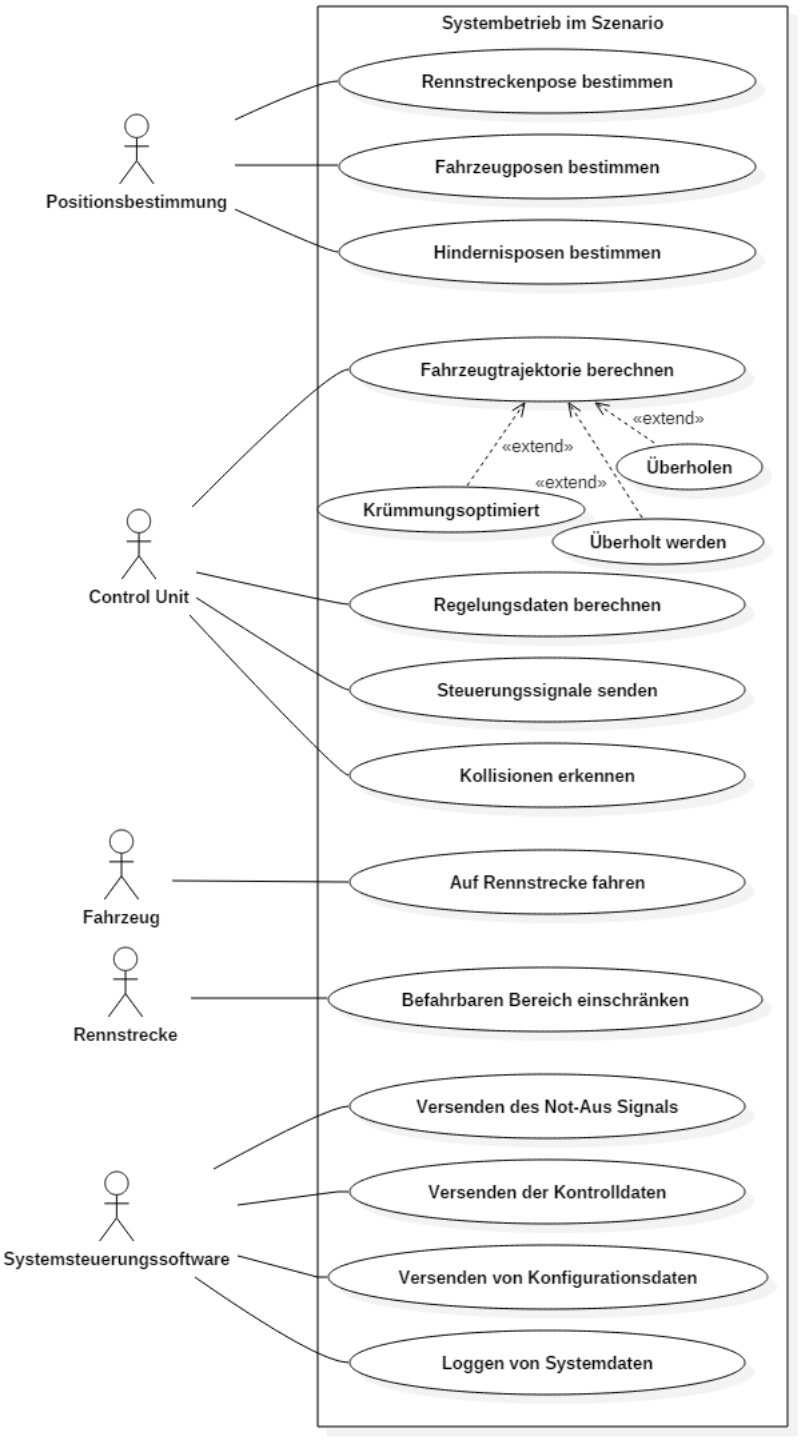


Abbildung 6.2.: Zweiter Teil des Anwendungsfalldiagramms für das Szenario „unfallfreier Überholvorgang“ mit den fünf Subsystemen sowie statischen Hindernissen, die jedoch zum Subsystem Rennstrecke gehören.

ID	B02
Beschreibung	Die Positionsbestimmung erkennt die Positionen und Ausrichtungen der Fahrzeuge und kann diese identifizieren.
Akteure	Positionsbestimmung
Auslöser	Ein Kamerabild wurde eingelesen und verarbeitet.
Vorbedingungen	Das System befindet sich in der Initialisierung oder im Rennmodus.
Ergebnis	Die Posen aller Fahrzeuge, die sich auf der Rennstrecke befinden wurden ermittelt und zugeordnet.
Standardablauf	1. Die Positionsbestimmung verarbeitet ein Bild. 2. Die Positionsbestimmung hat die Posen und Identitäten der Fahrzeuge ermittelt.
Anforderung	Sys1.2

Tabelle 6.11.: Anwendungsfalltabelle Pose und Identifizierung der Fahrzeuge.

ID	B03
Beschreibung	Die Positionsbestimmung erkennt die Positionen und Ausrichtungen aller Hindernisse auf der Rennstrecke.
Akteure	Positionsbestimmung
Auslöser	Ein Kamerabild wurde eingelesen und verarbeitet.
Vorbedingungen	Das System befindet sich in der Initialisierung, im Standby oder im Rennmodus.
Ergebnis	Die Posen der Hindernisse wurden ermittelt.
Standardablauf	1. Die Positionsbestimmung verarbeitet ein Bild. 2. Die Posen der Hindernisse wurden von der Positionsbestimmung ermittelt.
Anforderung	Rah1.7, Rah4.4, Rah4.5, Rah4.6,Rah4.8, SW1.3

Tabelle 6.12.: Anwendungsfalltabelle Erkennung der Hindernisse.

ID	B04
Beschreibung	Die Control-Unit kann statisch unterschiedliche Trajektorien planen.
Akteure	Control-Unit
Auslöser	Das System soll initialisiert und die Trajektorien statisch berechnet werden.
Vorbedingungen	Das System befindet sich in der Initialisierung. Fahrverhalten und Mindestdurchschnittsgeschwindigkeit wurden festgelegt.
Ergebnis	Die Control-Unit hat zum Abschluss der Initialisierung alle benötigten Trajektorien für das zu steuernde Fahrzeuge berechnet.
Standardablauf	1. Die Initialisierung der Control-Unit wurde gestartet. 2. Die Control-Unit empfängt alle benötigten Konfigurationsdaten von der Systemsteuerungssoftware. 3. Die Control-Unit empfängt Posen aller auf der Rennstrecke befindlichen Objekte von der Positionsbestimmung. 4. Die Control-Unit berechnet alle benötigten Trajektorien je nach Fahrverhalten.
Anforderung	Sys1.3, Sys4, SW2.1.3, SW2.1.4, SW2.1.5, SW2.1.9, SW2.1.10

Tabelle 6.13.: Anwendungsfalltabelle Berechnung Trajektorien.

ID	B05
Beschreibung	Die Control-Unit berechnet anhand der empfangenen Positionsdaten passende Regelungsdaten.
Akteure	Control-Unit
Auslöser	Der Rennbetrieb wurde gestartet.
Vorbedingungen	Positionsdaten werden empfangen. Vorplanung wurde abgeschlossen.
Ergebnis	Die Control-Unit berechnet für das zugewiesene Fahrzeug passende Regelungswerte.
Standardablauf	1. Positionsdaten werden eingelesen. 2. Vorplanungsdaten werden in Rechnung einbezogen. 3. Control-Unit berechnet passende Regelungswerte um der geplanten Trajektorie zu folgen.
Anforderung	SW2.1.2

Tabelle 6.14.: Anwendungsfalltabelle Berechnung von Regelungsdaten.

ID	B06
Beschreibung	Die Control-Unit sendet die berechneten Regelungswerte als Steurungssignale an ihr Fahrzeug.
Akteure	Control-Unit (CarControl)
Auslöser	Der Rennbetrieb wurde gestartet.
Vorbedingungen	Die Control-Unit hat Regelungswerte berechnet und in Steuerungssignale umgewandelt.
Ergebnis	Fahrzeug fährt nach empfangenen Steuerungssignalen.
Standardablauf	1. Control-Unit berechnet Regelungsdaten. 2. Control-Unit wandelt Regelungsdaten in Steuerungssignale um. 3. Control-Unit sendet über CarControl Steuerungssignale an das zugewiesene Fahrzeug.
Anforderung	SW2.1.11, SW2.2

Tabelle 6.15.: Anwendungsfalltabelle Steuerungssignale senden.

ID	B07
Beschreibung	Die Control-Unit erkennt, wenn während des Rennbetriebs eine Kollision des eigenen Fahrzeugs stattfindet.
Akteure	Control-Unit
Auslöser	Fahrzeug kollidiert mit anderem Fahrzeug, Hindernis oder der Rennstreckenbegrenzung.
Vorbedingungen	Das System ist im Rennbetrieb.
Ergebnis	Falls ein Kollision erkannt wird, wird das System in einen sicheren Fehlerzustand überführt.
Standardablauf	1. Control-Unit überwacht das eigene Fahrzeug auf Kollision. 2. Control-Unit reagiert entsprechend, falls Kollision stattfindet.
Anforderung	SW2.1.12

Tabelle 6.16.: Anwendungsfalltabelle Kollisionen erkennen.

ID	B08
Beschreibung	Ein Fahrzeug fährt auf der Rennstrecke.
Akteure	Fahrzeug
Auslöser	Das Fahrzeug hat die Steuerungsbefehle von der Control-Unit erhalten.
Vorbedingungen	Das System befindet sich im Rennbetrieb.
Ergebnis	Das Fahrzeug fährt auf dem befahrbaren Bereich der Rennstrecke.
Standardablauf	1. Das Fahrzeug empfängt die Steuerungsdaten von der Control-Unit. 2. Das Fahrzeug bewegt sich, basierend auf den Steuerungsdaten über die Rennstrecke.
Anforderung	Sys1.4, Sys4

Tabelle 6.17.: Anwendungsfalltabelle Fahrzeuge fahren auf der Rennstrecke.

ID	B09
Beschreibung	Ein auf der Rennstrecke befindliches Hindernis schränkt den befahrbaren Bereich ein. Das Hindernis ist ein Bestandteil des Subsystems Rennstrecke.
Akteure	Rennstrecke
Auslöser	Vor der Initialisierung des Systems wurde ein Hindernis auf dem befahrbaren Teil der Rennstrecke platziert.
Vorbedingungen	-
Ergebnis	Die Rennstrecke ist durch das Hindernis verengt.
Standardablauf	1. Das Hindernis wird auf der Rennstrecke platziert.
Anforderung	Rah5, HW4

Tabelle 6.18.: Anwendungsfalltabelle Hindernisse schränken befahrbaren Bereich ein.

ID	B10
Beschreibung	Die Systemsteuerungssoftware versendet ein Not-Aus Signal.
Akteure	Systemsteuerungssoftware
Auslöser	Es ist ein sicherheitskritischer Fehler im System aufgetreten.
Vorbedingungen	System wurde gestartet.
Ergebnis	Das Not-Aus Signal wurde versendet und das System geht in einen sicheren Fehlerzustand.
Standardablauf	1. Die Systemsteuerungssoftware erkennt einen sicherheitskritischen Fehler im System. 2. Ein Not-Aus Signal wird versendet.
Anforderung	SW3.2

Tabelle 6.19.: Anwendungsfalltabelle Systemsteuerungssoftware versendet Not-Aus Signal.

ID	B11
Beschreibung	Die Systemsteuerungssoftware versendet Kontrolldaten.
Akteure	Systemsteuerungssoftware
Auslöser	Es soll ein Wechsel der Systemzustände (siehe Tabelle A.2) der Subsysteme stattfinden.
Vorbedingungen	-
Ergebnis	Das entsprechende Subsysteme hat seinen Systemzustand entsprechend geändert.
Standardablauf	1. Die Systemsteuerungssoftware versendet Kontrolldaten an ein Subsystem 2. Das Subsystem ändert seinen Zustand entsprechend der mitgesendet Daten.
Anforderung	SW3.1.12

Tabelle 6.20.: Anwendungsfalltabelle Kontrolldaten der Systemsteuerungssoftware.

ID	B12
Beschreibung	Systemsteuerungssoftware versendet Konfigurationsdaten.
Akteure	Systemsteuerungssoftware
Auslöser	Der Adminstrator will Konfigurationsdaten versenden
Vorbedingungen	Die Inspektion wurde erfolgreich abgeschlossen.
Ergebnis	Es wurde ein Netzwerkpaket mit den Konfigurationsdaten versendet.
Standardablauf	1. Der Adminstrator stellt die Mindestdurchschnittsgeschwindigkeit, das Fahrverhalten, und die Control-Unit Zuweisung ein. 2. Ein Netzwerkpaket mit diesen Konfigurationsdaten wurde versendet.
Anforderung	SW3.1.4

Tabelle 6.21.: Anwendungsfalltabelle Konfigurationsdaten der Systemsteuerungssoftware.

ID	B13
Beschreibung	Die Systemsteuerungssoftware schreibt Systemdaten während des Betriebs mit.
Akteure	Systemsteuerungssoftware
Auslöser	Systemsteuerungssoftware wird gestartet.
Vorbedingungen	Systemsteuerungssoftware wurde gestartet.
Ergebnis	Datei mit Systeminformationen des letzten Ablaufs wurde erstellt.
Standardablauf	1. Systemsteuerungssoftware erstellt neue Log-Datei für aktuellen Systemlauf. 2. Systemsteuerungssoftware schreibt während des Betriebs Systeminformationen in eine Datei.
Anforderung	NSW1

Tabelle 6.22.: Anwendungsfalltabelle Loggen von Systemdaten.

7. Anforderungsspezifikation

Im folgenden Kapitel werden die an das System gestellten Anforderungen beschrieben. Diese stellen sicher, dass das Szenario unter sicherheitskritischen Aspekten erfüllt werden kann und leiten sich aus der Aufgabenstellung und dem Szenario (siehe Abschnitt 2.2 und 2.3), der Systemanalyse (siehe Abschnitt 5.8) sowie aus den Anwendungsfällen (siehe Kapitel 6) ab. Viele Anforderungen wurden aus dem Endbericht der Projektgruppe *RCCARS* [RCC16] übernommen, da das System auf dem Projekt aufbaut. Die übernommenen Anforderungen sind mit einem $*$ am Ende der Anforderung gekennzeichnet. Die einzelnen Anforderungen sind mit eindeutigen Sätzen spezifiziert, möglichst präzise und verständlich formuliert sowie hierarchisch angeordnet. Die Basisanforderungen werden somit durch die untergeordneten Anforderungen konkretisiert. Die Nummerierung der Anforderungen basiert auf den verschieden Subsystemen, die in Kapitel 5 beschrieben wurden. Die Anforderungen sind, wenn nötig, mit Referenzen auf die verschiedenen Kapitel versehen und mit einem kurzen Text erläutert. Zusätzliche Anforderungen die während der Entwicklungsphase identifiziert wurden, wurden nachträglich in die Anforderungsanalyse eingepflegt. Dedizierte Aspekte bzgl. der Sicherheit des Systems und daraus resultierende Sicherheitsanforderungen werden in Kapitel 8 gesondert betrachtet.

7.1. Rahmenbedingungen

Das System, das in diesem Projekt entwickelt werden soll, muss in der Lage sein zwei Fahrzeuge unfallfrei über die Rennstrecke zu steuern. Dabei sollen die Fahrzeuge sich überholen und Hindernissen auf der Strecke ausweichen können. Die folgenden Rahmenbedingungen beschreiben die grundlegenden Anforderungen, die an das System und die Systemumgebung gestellt werden. In den Bedingungen sind auch die sicherheitskritischen Voraussetzungen und Einschränkungen enthalten. Diese müssen eingehalten werden, um das in Kapitelabschnitt 2.3 beschriebene Szenario erfolgreich durchzuführen.

Bevor ein Rennen gestartet wird, muss durch ein Systemadministrator eine Inspektion durchgeführt werden. Hierzu gehört die Überprüfung der Hardwarekomponenten auf Vollständigkeit und Funktionsfähigkeit. Zudem überprüft der Administrator, ob die Rahmenbedingung Rah1 bis Rah3 und Rah5 vor Rennbeginn erfüllt sind. Diese Bedingungen müssen auch während des Rennbetriebes eingehalten werden. Des Weiteren muss auch während des Rennbetriebs die Bedingung Rah4 gültig bleiben.

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

Kontrollierte Umweltbedingungen bedeuten, dass das Rennen im Indoor-Bereich stattfinden muss. Dies ermöglicht, dass die Fahrbahn, die Fahrzeugreifen und alle Hindernisse 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 Sonnenlicht relativ viel Infrarotlicht enthält und 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 Hindernissen sein, die nicht von der Positionsbestimmung erfasst werden können.

Stehende Fahrzeuge und Quader gelten jeweils als Hindernis. (Hindernisse werden in HW4.4 definiert.)

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

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 als Eingabeparameter übergeben.

Rah3.1 Der Streckenaufbau muss den Eingabeparametern für den Rennstreckenverlauf entsprechen, welche an die Control-Unit übergeben werden.

In dieser Ausbaustufe muss der Streckenverlauf wie im Vorgängerprojekt dem Schema aus Abbildung 5.9 entsprechen.

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∗.

Durch eine beidseitige Streckenbegrenzung wird sichergestellt, dass kein Fahrzeug die Rennstrecke verlassen kann.

Rah3.5 Die Rennstrecke muss eben sein∗.

Es muss eine ebene Rennstrecke verwendet werden, da durch die zweidimensionale Bilderfassung 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. Die einzelnen Segmente dürfen keine Beschädigungen auf der Fahrbahn oder der Streckenbegrenzung aufweisen.

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.

Rah3.8 Die Rennstrecke muss breit genug für einen Überholvorgang sein.

Im Rahmen des Szenarios soll ein Überholvorgang möglich sein. Um diesen zu gewährleisten muss die Fahrbahn der Rennstrecke breit genug sein, dass sich zwei Fahrzeuge nebeneinander überholen können.

Rah4 **Die Lage der Rennstrecke, die Posen der Fahrzeuge und die Positionen der Hindernisse dürfen nach der Initialisierung des Systems nicht mehr manuell verändert werden.**

Das System muss bei der Initialisierung die Lage der Rennstrecke, die Posen der Fahrzeuge und die Positionen und Ausrichtungen der Hindernisse 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 Abschnitt 5.4.1).

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∗.

Rah4.4 Hindernisse dürfen nach der Initialisierung nicht mehr bewegt werden.

Rah4.5 Hindernisse dürfen nach der Initialisierung nicht mehr zur Rennstrecke hinzugefügt werden.

Rah4.6 Hindernisse dürfen nach der Initialisierung nicht mehr von der Rennstrecke entfernt werden.

Rah4.7 Die Fahrzeuge müssen initial auf der Start-/Ziellinie platziert werden.

Rah4.8 Die Hindernisse dürfen nicht auf der Start-/Ziellinie platziert werden.

Rah5 **Auf der Rennstrecke muss bei gewähltem Fahrverhalten ein Überholvorgang durchführbar sein.**

Ein Überholvorgang beinhaltet sowohl das Überholen eines fahrenden Fahrzeugs als auch das Umfahren eines Hindernisses. Ein Überholvorgang gilt als abgeschlossen, wenn das überholende Fahrzeug einen Mindestabstand von zwei Fahrzeuglängen zum überholten Fahrzeug aufweist.

Rah5.1 Die Mindestdurchschnittsgeschwindigkeiten der beiden Fahrzeuge müssen unterschiedlich sein.

Rah5.2 Ist es dem Fahrzeug nicht möglich das Hindernis ohne Kollision zu umfahren, muss das System in einen sicheren Fehlerzustand übergehen.

Rah5.3 Ein Fahrzeug muss einen Mindestabstand von zwei Fahrzeuglängen zu einem vorausfahrenden Fahrzeug einhalten.

Wird dieser Abstand unterschritten, muss ein Überholvorgang eingeleitet werden.

7.2. Funktionale Anforderungen

Die folgenden Anforderungen beschreiben die Funktionalitäten, die das System aufweisen muss. Diese sind unterteilt in „generelle Systemanforderungen“, „Hardwareanforderungen“ und „Softwareanforderungen“.

7.2.1. Generelle Systemanforderungen

Die generellen Systemanforderungen leiten sich aus der Aufgabenstellung und dem gewählten Szenario ab. Aus diesen Anforderungen leiten sich die spezifischen Hardwareanforderungen und Softwareanforderungen ab.

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

Dies ist die grundlegende Basisanforderung für das gesamte System der zweiten Ausbaustufe. Die folgenden Unteranforderungen entsprechen der Einteilung des Systems in fünf Subsysteme, so wie diese in Kapitel 5 und Abschnitt 5.8 beschrieben wurden. Da der Rennstreckenverlauf 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 zwei ferngesteuerte Fahrzeuge besitzen.

Es werden ferngesteuerte Modellrennwagen der *dNaNo*-Plattform von *Kyosho* 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 Posen der Fahrzeuge zu bestimmen∗.

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 zwei Fahrzeuge 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.

Sys1.4 Das System muss die autonome Fahrzeugsteuerung auf Grundlage der selbstständig ermittelten Fahrzeugpositionen vornehmen∗.

Sys1.5 Eine manuelle Korrektur der Pose der Fahrzeuge sowie der Fahrzeugsteuerung darf nicht vorgenommen werden∗.

Sys1.6 Das System muss eine Rennstrecke besitzen∗.

Die Rennstrecke muss die Rahmenbedingung Rah3 inklusive aller Unteranforderungen erfüllen. Aus dieser Anforderung leitet sich die Hardwarebasisanforderung HW5 ab.

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

Aus dem Szenario der zweiten Ausbaustufe (siehe Abschnitt 2.3) ergibt sich, dass das System weiterhin 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 HW5.3.1 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 vor Start des Rennbetriebes festgelegten Mindestdurchschnittsgeschwindigkeit über eine vorgegebene Rennstrecke zu steuern.**

Die Fahrzeuge sollen nach Möglichkeit mit der festgelegten Mindestdurchschnittsgeschwindigkeit über die Rennstrecke fahren. Damit aber Unfälle vermieden und Ausweichmanöver möglich bleiben, wurde auf eine festgelegte Mindestgeschwindigkeit verzichtet.

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

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

Sys4.2 Das System muss fähig sein, die Fahrzeuge so zu steuern, dass sie die Rennstrecke nicht verlassen∗.

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

Aus dieser Anforderung geht hervor, dass Unfälle, bei denen sich das Fahrzeug überschlägt oder auf die Seite kippt, nicht passieren dürfen. Aus dieser Anforderung leitet sich die Anforderung SW2.1.12.6 ab.

Sys4.4 Das System muss fähig sein, die Fahrzeuge mindestens 10 Runden fahren zu lassen.

Sys4.5 Das System muss fähig sein, die Fahrzeuge ohne Kollision mit anderen Fahrzeugen über die Strecke zu führen.

Sys4.6 Das System muss einen Unfall autonom erkennen können∗.

Sys4.7 Das System muss den Rennbetrieb stoppen, falls es einen Unfall erkennt∗.

Sys4.8 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 Abschnitt 5.4) könnten noch größere Verzögerungen zu Kollisionen mit der Fahrbahnbegrenzung führen.
Die Projektgruppe *RCCARS* hat 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 SW1.6 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.**

Sys5.1 Das System muss durch einen modularen Aufbau die Voraussetzungen schaffen, dass die autonome Steuerung der Fahrzeuge zu einem späteren Zeitpunkt optimiert werden kann.

Diese Anforderung leitet sich direkt aus dem längerfristigen Ziel des Systems ab, das darin besteht ein Testfeld zur Analyse und Entwicklung von Rennstrategien zu realisieren (siehe Kapitelabschnitt 2.2).

7.2.2. Hardwareanforderungen

In diesem Abschnitt werden die Anforderungen an die Hardware des Systems beschrieben. Diese leiten sich aus den Abschnitten der *RCCARS* Systemanalyse in Kapitel 5 und den notwendigen Anpassungen aus Abschnitt 5.8 ab. Die Hardwarebasisanforderungen HW1 bis HW4 sind nach den Subsystemen sortiert.
Des Weiteren sind in HW5 allgemeine Hardwareanforderungen definiert.

HW1 **Die Hardware des Subsystems 1 (Fahrzeug) muss aus mindestens zwei ferngesteuerten Fahrzeugen bestehen.**

HW1.1 Die Fahrzeuge müssen aus austauschbaren Komponenten bestehen∗.

Die Reifen, der Akku und die Aufhängungen sind Verschleißteile und müssen daher austauschbar sein. Ein modularer Aufbau bietet die Möglichkeit die Rennperformance der Fahrzeuge zu verbessern.

HW1.2 Die ferngesteuerten Fahrzeuge müssen vom Typ *dNaNo* sein∗.

HW1.3 Die Fahrzeuge müssen korrekt funktionieren.

HW1.3.1 Der Akku der Fahrzeuge muss fähig sein, die Fahrzeuge mit Spannung zu versorgen, ohne dass ein für die geforderte Mindestdurchschnittsgeschwindigkeit relevanter Leistungsverlust auftritt.

HW1.3.2 Die Fahrzeuge müssen vollständig sein∗.

HW1.4 Die Fahrzeuge müssen mit Reflexionsmarkern ausgestattet sein∗.

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

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.

Die Reflexionsmarker müssen laut Referenzprojekt [Rut10] mindestens $1\,\text{cm} \cdot 1\,\text{cm}$ groß sein.

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∗.

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

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

HW2.1 Das Subsystem 2 muss einen Rechner zur Ausführung der Positionsbestimmungssoftware besitzen∗.

Für die Steuerung der Fahrzeuge soll für jedes Fahrzeug ein eigener Rechner eingesetzt werden. Dementsprechend muss es noch einen weiteren Rechner geben auf dem die Positionsbestimmung läuft.

HW2.2 Das Subsystem 2 muss eine Beleuchtungseinheit besitzen∗.

Die Beleuchtungseinheit wird in Abschnitt 5.2 genauer beschrieben.

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.

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.

Diese Anforderungen gehen aus den entsprechenden Abschnitten der Systembeschreibung hervor (siehe Abschnitt 5.2).

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.

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.

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

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

HW2.3.2 Das Subsystem 2 muss ein zur Kamera kompatibles Weitwinkelobjektiv besitzen∗.

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 eine Halterung für die Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten besitzen.

HW2.4.1 Die Halterung muss die stabile Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten ermöglichen.

HW2.4.2 Die Halterung muss eine Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten zentral über der Rennstrecke ermöglichen.

HW2.4.3 Die Halterung muss eine Befestigung der Beleuchtungseinheit und der Bilderfassungskomponenten ermöglichen, sodass die gesamte Rennstrecke von der Kamera erfasst werden kann.

HW2.4.4 Die Beleuchtungseinheit und die Bilderfassungskomponenten müssen so dicht wie möglich nebeneinander an der Halterung befestigt werden.

HW2.4.5 Die Halterung muss an einem Gestell befestigt werden können.

HW2.4.6 Die Halterung muss an der Decke des Projektgruppenraumes angebracht werden können.

HW2.4.7 Die Halterung muss die Möglichkeit bieten drei Infrarotlichtquellen anzubringen.

HW3 **Die Hardware des Subsystems 3 (Control-Unit) muss alle erforderlichen Komponenten für die autonome Fahrzeugsteuerung beinhalten.**

Diese Anforderung leitet sich direkt aus den Systemanforderungen ab.

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

Die Berechnungen des Subsystem 3 werden auf verschiedenen Rechnern ausgeführt. Die genaueren Anforderungen an die Hardware der Rechner sind in HW5 beschrieben.

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

Der DA-Wandler ist in dem vorhandenen System auf einem Entwicklungsboard realisiert. Dieser wandelt die digitalen Signale der Regelungssoftware in analoge Signale um, welche über die Fernbedienung an die Fahrzeuge gesendet werden. Der Begriff CarControl bezeichnet die Einheit aus dem Entwicklungsboard und der angeschlossenen Fernbedienung.

HW3.2.1 Die CarControl muss per USB-Schnittstelle eine Verbindung zu einem Rechner mit Regelungssoftware besitzen.

HW3.2.2 Die CarControl muss zwei DAC besitzen.

Je ein DAC wird für die Längs- und Quersteuerung verwendet.

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

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.

Damit die Steuerung präzise genug ist muss die Software laut Anforderung SW2.3 mindestens 100 Regelungswerte pro Sekunde ausgeben.

HW3.3 Das Subsystem 3 muss mindestens zwei CarControl besitzen.

Die Anzahl der CarControl entspricht der Anzahl der Fahrzeuge.

HW3.3.1 Jede CarControl muss genau eine Fernbedienung besitzen.

Über die Fernbedienung werden die Regelungswerte an die Fahrzeuge gesendet. Die Fernbedienung ist vom Typ *Perfex KT-18* von *Kyosho*.

HW3.3.2 Die Fernsteuerung muss das Fahrzeug steuern können∗.

HW3.3.3 Jede Fernbedienung muss mit einer eigenen Frequenz senden.

HW3.3.4 Jede Frequenz wird nur von einer Fernbedienung verwendet.

HW3.3.5 Jede Fernbedienung muss an genau ein Fahrzeug senden.

HW3.3.6 Jedes Fahrzeug wird nur von einer CarControl gesteuert.

HW3.3.7 Die Fernsteuerung muss mit dem Fahrzeug kompatibel sein∗.

HW3.3.8 Die Potentiometer in der Fernsteuerung müssen deaktiviert werden∗.

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

Diese Anforderung leitet sich aus der Systemanforderung Sys1.6 ab.

HW4.1 Der Verlauf der Rennstrecke muss eine geschlossene Runde bilden∗.

HW4.2 Die Rennstrecke muss für die Fahrzeuge geeignet sein∗.

Die Fahrzeuge sind wie in HW1 beschrieben vom Typ *dNaNo* von *Kyosho*. Die Oberfläche, die Breite und Kurvenradien der Strecke müssen für diese Fahrzeuge geeignet sein.

HW4.3 An den äußeren Ecken der Rennstrecke müssen Reflexionsmarker platziert sein∗.

Über diese Reflexionsmarker wird die Position der Rennstrecke bestimmt. Zudem wird über diese Eckmarken ein Koordinatensystem aufgebaut. Hierfür werden Vektoren aus den Eckmarkern erstellt aus denen eine Ebene aufgespannt wird.

HW4.4 Die Rennstrecke muss die Möglichkeit bieten Hindernisse zu platzieren.

HW4.4.1 Ein Hindernis muss ein Quader sein.

HW4.4.1.1 Der Quader muss eine Höhe zwischen 1 cm und 3 cm besitzen.

HW4.4.1.2 Der Quader muss eine Länge von $i \cdot 5$ cm besitzen, $i \in \{1, 2, 3, 4\}$.

HW4.4.1.3 Der Quader muss eine Breite von $j \cdot 5$ cm besitzen, $j \in \{1, 2, 3, 4\}$.

HW4.4.1.4 Der Quader muss aus Material sein, das kein Infrarotlicht reflektiert.

HW4.4.2 Jedes Hindernis muss mit Reflexionsmarkern versehen sein.

HW4.4.3 Die Reflexionsmarker müssen so angeordnet sein, dass die Position des Hindernisses eindeutig erkannt wird.

HW4.4.4 Die Reflexionsmarker müssen so angeordnet sein, dass die Größe des Hindernisses eindeutig erkannt wird.

HW4.4.5 Die Reflexionsmarker müssen so angeordnet sein, dass das Hindernis eindeutig identifiziert wird.

HW4.4.6 Die Hindernisse dürfen die Fahrbahn nicht beschädigen.

HW5 **Zur Systemhardware müssen mehrere Rechner gehören.**

Diese Basisanforderungen beziehen sich auf die Rechner der verschiedenen Subsysteme.

HW5.1 Jeder Rechner muss einen ausreichend leistungsfähigen Prozessor besitzen∗.

HW5.2 Einer der Rechner muss fähig sein, die Positionsbestimmungssoftware auszuführen.

HW5.2.1 Der Rechner muss eine USB 3.0 Schnittstelle besitzen∗.

Über diese Schnittstelle werden die Daten der Kamera empfangen.

HW5.3 Einer der Rechner muss fähig sein, die Systemsteuerungssoftware auszuführen

HW5.3.1 Der Rechner muss einem Administrator Kontrollelemente für die Bedienung des Systems bereitstellen∗.

HW5.4 Die anderen Rechner müssen fähig sein, die Software der Control-Unit auszuführen.

HW5.5 Diese Rechner müssen eine USB-Schnittstelle zur Kommunikation mit der CarControl besitzen.

7.2.3. Softwareanforderungen

In diesem Abschnitt werden die an die Software gestellten Anforderungen erläutert. Sie leiten sich aus den Beschreibungen der Software-Komponenten ab.

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

Im Folgenden werden die Anforderungen an die Positionsbesimmungssoftware definiert, die sich aus den in Abschnitt 5.2.2 und 5.8 beschriebenen Eigenschaften und den sicherheitsrelevanten Aspekten (siehe Abschnitt 5.2.3) ergeben.

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 Fahrzeuge zu erkennen∗.

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

Hierzu gehört die Anwendung von rauschunterdrückenden Filtern und das Entzerren der Bilder.

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

Die Reflexionsmarker werden in Abschnitt 5.1 beschrieben.

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 fähig sein, aus den Bilddaten der Kamera die Position der Hindernisse zu identifizieren.

SW1.3.1 Die Positionsbestimmungssoftware muss durch Erkennung der Reflexionsmarker die Hindernisse erfassen.

SW1.3.2 Die Positionsbestimmungssoftware muss die zweidimensionalen Hinderniskoordinaten feststellen.

SW1.4 Die Positionsbestimmungssoftware muss die Daten der Fahrzeugposen an die Software der Control-Unit weiterleiten∗.

SW1.5 Die Positionsbestimmungssoftware muss die Daten der Hindernisposition an die Software der Control-Unit weiterleiten.

SW1.6 Die Positionsbestimmungssoftware muss fähig sein, die Bilderfassung, die Erkennung der Fahrzeugposen und Hindernispositionen und die Weiterleitung der ermittelten Daten an die Regelungssoftware in höchstens 10 ms durchzuführen.

SW1.7 Die Positionsbestimmungssoftware muss fähig sein, die Lage der Rennstrecke zu erfassen∗.

Die Fahrzeugposen basieren auf den relativen Koordinaten der Rennstrecke.

SW1.8 Die Positionsbestimmungssoftware muss Störungen erkennen∗.

SW1.8.1 Die Positionsbestimmungssoftware muss erkennen, ob die Rennstrecke verschoben wurde.

Wenn die Positionsbestimmungssoftware eine Verschiebung der Rennstrecke erkennt kann diese Verschiebung mehrere Ursachen haben. Zum einen kann die Kamera oder das Gestell, an der die Kamera befestigt ist, bewegt worden sein. Zum anderen kann einer der Eckmarker bewegt worden sein. Diese Veränderungen müssen an die Systemsteuerungssoftware weitergeleitet werden. Allerdings müssen kleine Erschütterung aus der Umgebung und Ungenauigkeiten der Bildgebung toleriert werden. Kleine Erschütterungen entstehen z.B. wenn eine Person am Setting vorbeiläuft. Diese Erschütterungen werden über das Gestell an die Kamera weitergeleitet.

SW1.8.2 Falls die Positionsbestimmungssoftware eine Störung erkennt, muss sie dies der Systemsteuerungssoftware mitteilen.

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

Im Folgenden werden die Anforderungen an die Software der Control-Unit definiert, die sich aus den in Abschnitt 5.3.2 und 5.8 beschriebenen Eigenschaften ergeben. Hierzu gehören auch die sicherheitsrelevanten Aspekte, die in Abschnitt 5.3.3 erläutert wurden.

SW2.1 Die Software des Subsystem 3 muss aus einzelnen Komponenten bestehen.

Zu den Komponenten der Software der Control-Unit gehören eine Regelungssoftware, eine Planungssoftware, eine Kommunikationssoftware und eine Kontrollsoftware. Alle Bestandteile kommunizieren miteinander und werden in ihren Anforderungen im Nachfolgenden spezifiziert.

SW2.1.1 Die Kommunikationssoftware 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∗.

SW2.1.4 Die Regelungssoftware muss den Verlauf der Rennstrecke und Referenztrajektorie des Fahrzeuges als Eingabeparameter empfangen∗.

SW2.1.5 Die Planungssoftware muss den Verlauf der Rennstrecke und Referenztrajektorie des Fahrzeuges als Eingabeparameter empfangen∗.

SW2.1.6 ~~Die Regelungssoftware muss das Fahrzeug gegen den Uhrzeigersinn um die Rennstrecke steuern~~∗.

Diese Anforderung ist für die Projektgruppe *RCCARSng* nicht mehr notwendig, da im aktuellen Setting auch eine Fahrt entgegen des Uhrzeigersinns ermöglicht wird.

SW2.1.7 Die Regelungssoftware muss fähig sein, das Fahrzeug mit der vorgegebenen Mindestdurchschnittsgeschwindigkeit über eine vorgegebene Rennstrecke zu steuern.

SW2.1.8 Die Regelungssoftware muss fähig sein, das Fahrzeug über die vorgegebene Mindestrundenanzahl von zehn Runden hinweg über die Rennstrecke zu steuern.

SW2.1.9 Die Planungssoftware muss fähig sein, Routen für die Hindernisumfahrung zu berechnen.

SW2.1.9.1 Die Planungssoftware muss fähig sein, Hindernisse zu berücksichtigen.

SW2.1.9.2 Die Planungssoftware muss fähig sein, die vorgegebene Route anzupassen.

SW2.1.10 Die Planungssoftware muss fähig sein, Routen für das Überholen von Fahrzeugen zu berechnen.

SW2.1.10.1 Die Planungssoftware muss fähig sein, Fahrzeugpositionen auszuwerten.

SW2.1.10.2 Die Planungssoftware muss fähig sein, die vorgegebene Route anzupassen.

SW2.1.10.3 Die Regelungssoftware muss fähig sein, das Überholen abzubrechen.

SW2.1.10.4 Bewegt sich ein Fahrzeug aus dem Planungshorizont eines zweiten Fahrzeugs muss der Überholvorgang des zweiten Fahrzeugs abgebrochen werden.

Ein Überholvorgang muss abgebrochen werden, sobald die Voraussetzungen für einen eingeleiteten Überholvorgang nicht mehr erfüllt sind.

SW2.1.11 Die Kommunikationssoftware muss die Regelungsdaten für die Fahrzeugsteuerung an die CarControl übermitteln∗.

SW2.1.12 Die Kontrollsoftware muss Unfälle und Störungen erkennen.

SW2.1.12.1 Die Kontrollsoftware muss erkennen, ob das Fahrzeug den befahrbaren Bereich der Rennstrecke verlassen hat∗.

Der befahrbare Bereich ist in Abschnitt 5.4 definiert.

SW2.1.12.2 Die Kontrollsoftware muss fähig sein, eine Kollision des Fahrzeugs mit der Streckenbegrenzung zu erkennen∗.

SW2.1.12.3 Die Kontrollsoftware muss fähig sein, eine Kollision des Fahrzeugs mit einem Hindernis zu erkennen.

SW2.1.12.4 Die Kontrollsoftware muss fähig sein, eine Kollision des Fahrzeugs mit anderen Fahrzeugen zu erkennen.

SW2.1.12.5 ~~Die Kontrollsoftware muss erkennen, ob das Fahrzeug in die richtige Richtung fährt~~∗.

Diese Anforderung ist für die Projekgruppe *RCCARSng* nicht mehr notwendig, siehe auch Anforderung SW2.1.6.

SW2.1.12.6 Die Kontrollsoftware muss auf den Verlust des Sichtkontakts zum Fahrzeug reagieren können.

Dieser Fall kann eintreten, falls sich das Fahrzeug überschlägt, bei einer Störung der Bilderfassung oder das Fahrzeug das Sichtfeld der Kamera verlässt. In einem solchen Fall sendet die Positionsbestimmungssoftware eine Fehlermeldung, auf welche die Kontrollsoftware reagieren muss.

SW2.1.12.7 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 1.2 m/s kann das Fahrzeug in dieser Zeit 2.4 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, sodass die Geschwindigkeit des Fahrzeugs bei der Kollision mit der Fahrbahnbegrenzung möglichst gering ist.

SW2.1.12.8 Die Kontrollsoftware muss fähig sein, den Verlust der Verbindung zur CarControl zu erkennen∗.

Die Kommunikationssoftware sendet Befehle über die CarControl an das Fahrzeug. Ist diese Verbindung unterbrochen, so ist keine Kontrolle über das Fahrzeug mehr möglich und es wird ein Fehler erzeugt.

SW2.1.12.9 Die Kontrollsoftware muss bei der Auswertung der Fahrzeugpose eine Plausibilitätsüberprüfung durchführen∗.

SW2.1.12.10 Falls die Kontrollsoftware eine Störung erkennt, muss das System in einen sicheren Fehlerzustand übergehen und die Regelungssoftware das Fahrzeug stoppen∗.

SW2.1.12.11 Falls die Kontrollsoftware eine Störung erkennt, muss sie dies der Systemsteuerungssoftware mitteilen∗.

Die Systemsteuerungssoftware muss in diesem Fall nur dafür sorgen, dass das System in den Fehlerzustand versetzt wird, da die Regelungssoftware dafür sorgt, dass die Fahrzeuge angehalten werden.

SW2.1.13 Bei der Entwicklung von Subsystem 3 muss das Werkzeug SCADE zum Einsatz kommen.

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

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 3V 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.

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

Da die Positionsbestimmung alle 10 ms neue Daten liefert, muss die Berechnung und Weiterleitung der Steuersignale in diesem Zeitraum abgeschlossen sein.

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

Diese Anforderung ergibt sich aus der Systemanforderung Sys2. Im Folgenden werden Anforderungen an die Systemsteuerungssoftware gestellt, die in Abschnitt 5.6 und 5.8 beschrieben wurden. Die untergeordneten Anforderungen leiten sich aus den Unteranforderung zu Sys2 ab.

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 eine Inspektion erfolgreich durchgeführt wurde.

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

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

SW3.1.4 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, alle notwendigen Konfigurationsparameter an das System zu übermitteln.

SW3.1.5 Das Benutzerinterface muss dem Administrator jederzeit die Möglichkeit bieten, den Rennbetrieb zu stoppen.

SW3.1.6 Das Benutzerinterface muss dem Administrator jederzeit die Möglichkeit bieten, das System auszuschalten.

SW3.1.7 Das Benutzerinterface muss dem Administrator jederzeit die Möglichkeit bieten, das System manuell in den Fehlerzustand zu versetzen.

SW3.1.8 Das Benutzerinterface muss dem Administrator über den aktuellen Systemzustand informieren.

SW3.1.9 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, die erwartete Anzahl der Fahrzeuge auf der Rennstrecke einzustellen.

SW3.1.10 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, eine individuelle Mindestdurchschnittsgeschwindigkeit für jedes Fahrzeug einzustellen.

SW3.1.11 Das Benutzerinterface muss dem Administrator die Möglichkeit bieten, ein individuelles Fahrverhalten für jedes Fahrzeug einzustellen.

SW3.1.12 Das Benutzerinterface muss dem Administrator die Möglichkeit beiten, den Status des Systems zu wechseln.

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

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

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 die Regelungssoftware müssen mit der höchsten Priorität ausgeführt werden∗.**

Damit die Positionsbestimmungssoftware und die Regelungssoftware die höchste Priorität im Vergleich zu anderen Prozessen erhalten muss ein Realzeitbetriebssystem als Ausführungsumgebung verwendet werden (siehe Abschnitt 5.2.2). Die Systemsteuerungssoftware muss dafür sorgen, dass sie die entsprechende Priorität enthalten. Daraus leitet sich folgende Anforderung an das Betriebssystem ab.

SW4.1 Das auf dem Rechner installierte Betriebssystem muss realzeitfähig sein.

7.3. Nicht-Funktionale Anforderungen

In diesem Abschnitt werden Anforderungen definiert, die die Qualität des Systems verbessern oder Entwicklungskosten reduzieren. Die Anforderungen werden in „Generelle Systemanforderungen“, „Hardwareanforderungen“ und „Softwareanforderungen“ unterteilt.

7.3.1. Generelle Systemanforderungen

In der aktuellen Ausbaustufe des Projektes *RCCARSng* 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.

7.3.2. Hardwareanforderungen

NHW1 **Die Projektgruppe muss die Anzahl der CarContol auf zwei erhöhen.**

7.3.3. Softwareanforderungen

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

Zur Nachvollziehbarkeit und Fehleranalyse werden die erzeugten Ausgaben der einzelnen Softwarekomponenten gespeichert. Hierzu gehört auch die Berechnungsdauer der Komponenten zur Erfüllung ihrer jeweiligen Aufgaben. Diese Protokollfunktion ist optional, da sie negativen Einfluss auf die Programmlaufzeit besitzt. Fehlerberichte hingegen werden immer ausgegeben und sind nicht optional.

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.

8. Sicherheitsanalyse/-konzept

In diesem Kapitel wird basierend auf den Arbeiten der Projektgruppe *RCCARS* ein Sicherheitskonzept für den erweiterten Systemaufbau erarbeitet, mit dem Ziel ein definiertes Schutzniveau zu erreichen. Im Fokus steht dabei 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 daraus anschließend entsprechende Sicherheitsanforderungen abgeleitet. Aus den Sicherheitsanforderungen wird dann für das System bzw. einzelne Teilsysteme ein Sicherheitskonzept erarbeitet, mit dem Ziel einen sicheren Fehlerzustand für das *RCCARS* System zu erreichen. Da das Zeitverhalten beim Zusammenspiel der einzelnen Subsysteme und beim Ansteuern des Fahrzeugs eine entscheidende Rolle spielt, wird der Betrachtung möglicher Schadenszenarien eine Realzeitbetrachtung vorangestellt.
Aufgrund der zeitlichen Befristung einer Projektgruppe ist diese Sicherheitsanalyse bei weitem nicht vollständig, sie bietet aber trotzdem einen Einblick in die Thematik und erläutert die entworfenen Mechanismen um ein gewisses Schutzniveau zu erreichen.

8.1. Realzeitbetrachtung

Die Realzeitbetrachtung des erweiterten *RCCARSng* Systems basiert auf den Erfahrungen und Evaluationsdaten der Projektgruppe *RCCARS* und wird an dieser Stelle wiederverwendet. Die rechtzeitige und zuverlässige Kommunikation zwischen Sensoren und Aktoren ist essentiell für das autonome Rennsystem *RCCARS*. Die Sensorik besteht aus einer Kamera, welche Bilder der Fahrzeuge auf der Rennstrecke erfasst. Die Positionsbestimmung ermittelt die jeweilige Fahrzeugpose und übermittelt diese über eine Netzwerkkommunikation an die entsprechende Control-Unit, welche die Regelungsparameter berechnet und diese mit Hilfe der CarControl an das Fahrzeug sendet. 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 der Projektgruppe *RCCARS* ist davon auszugehen, dass der Verlust von zwei Bildern in der Positionsbestimmung durch eine nur auf den Positionsdaten basierende Regelung nicht mehr auszugleichen ist. Daraus ergibt sich, dass auch die Berechnungen der Control-Unit und das Übertragen der Regelungsdaten an das Fahrzeug ebenfalls nicht länger als 10 ms benötigen sollten. In Abbildung 8.1 ist das Zusammenspiel der einzelnen parallel ausgeführten Subsysteme und der Verlust eines Bilds im Zeitabschnitt bis 50 ms exemplarisch dargestellt. Im Abschnitt 40 ms bis 80 ms sind anschließend die Auswirkungen beim Ausfall zweier Bilder dargestellt.

Das zum Zeitpunkt t_0 von der Positionsbestimmung erfasste Bild wird zum Zeitpunkt t_{20} vom Fahrzeug empfangen und liegt dort bis zum Zeitpunkt t_{30} für 10 ms an. Da die Positionsbestimmung zum Zeitpunkt t_{20} kein neues Bild erhält, werden keine neuen Positionsdaten berechnet und somit erhält die Control-Unit zum Zeitpunkt t_{30} keine neuen Positionsdaten. Dies führt dazu, dass das Fahrzeug weiterhin mit alten Regelungswerten angesteuert wird. Zum Zeitpunkt t_{30} erhält die Positionsbestimmung wieder ein Bild aus dem Videostream und folglich berechnet die Control-Unit zum Zeitpunkt t_{40} neue Regelungswerte welche spätestens zum Zeitpunkt t_{50} zum Fahrzeug gesendet werden.

Im zweiten dargestellten Fall erhält die Positionsbestimmung zu den Zeitpunkten t_{50} und t_{60} keine neuen Bildinformationen. Dies führt dazu, dass die Control-Unit im Zeitfenster von t_{50} bis t_{70} keine neuen Positionsdaten erhält und somit möglichst zeitnah zum Zeitpunkt $t_{70+\epsilon}$ einen Nothalt ausführt.

Wurde bei der Projektgruppe *RCCARS* von einer Systemausführung auf einem ein-

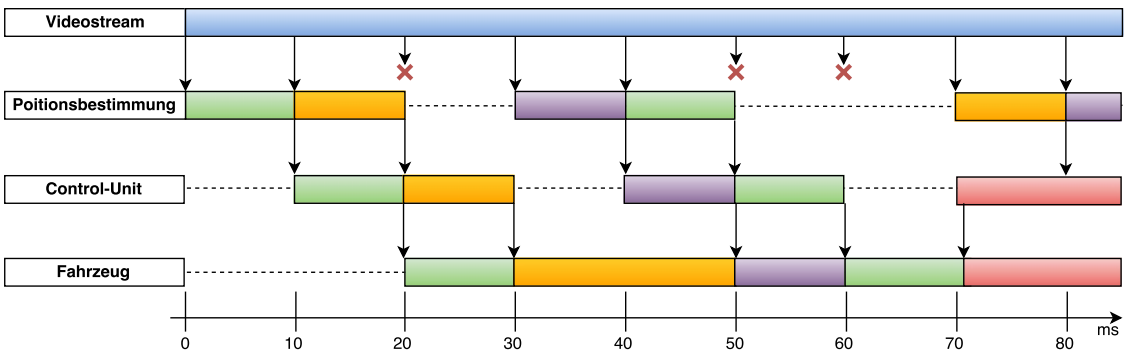


Abbildung 8.1.: Visualisierung des Realzeitverhaltens des Systems

zelnen Rechner ausgegangen (siehe Abbildung 5.10) und so von etwaigen Netzwerklatenzen abstrahiert, ist diese Annahme in dem aktuellen Aufbau mit unterschiedlichen Rechnern für Positionsbestimmung und den einzelnen Control-Units so nicht mehr haltbar. Da das System jedoch in einem eigenen Netzwerk ausgeführt wird, haben Analysen der Übertragungszeiten ergeben, dass diese weiterhin vernachlässigbar gering sind. Anders sieht es jedoch mit den Latenzen bei der Übertragung von Regelungswerten über die original *dNaNo* Fernbedienung aus. Eine Hardwareevaluation im Rahmen der PG *RCCARS* hat ergeben, dass die Übertragungszeiten erheblich schwanken und die Ursachen aufgrund der BlackBox Fernbedienung auch nicht detaillierter analysiert werden können. Aus diesem Grund muss die Control-Unit die Latenzen bei der Berechnung von Regelungswerten berücksichtigen um das Fahrzeug unter Einhaltung der festgelegten Realzeitanforderungen ansteuern zu können.

8.2. Schadenszenarien

Die Schadenszenarien betrachten die einzelnen Subsysteme und beleuchten verschiedene Fehlersituationen im Rennbetrieb. Diese werden unterteilt in die Kategorien **Störung**, **Ausfall** und **Fehler**. Eine Störung betrachtet einen zeitlich befristeten Effekt, wobei ein Ausfall zeitlich unbefristet ist.

8.2.1. Kommunikation

Störung Bei der Übertragung von Datenpaketen über das Netzwerk kann es zum Verlust einzelner Datenpakete durch Störungen kommen. Ebenso kann es zu Kommunikationsproblemen innerhalb einzelner Control-Units in Verbindung mit den CarControls oder zwischen CarControl und Fahrzeug kommen.

Ausfall Sollte die gesamte Kommunikation zwischen der Systemsteuerung, der Positionsbestimmung und den Control-Units abbrechen, können keine Netzwerkdatenpakete mehr gesendet und empfangen werden. In diesem Fall kann das System über die Systemsteuerung nicht mehr beeinflusst bzw. die Fahrzeuge nicht mehr angehalten werden. Des Weiteren können keine Positionsdaten mehr an die einzelnen Control-Units geschickt werden.

Fehler Beim Versenden der Daten über das Netzwerk kann es vorkommen, dass der Inhalt einzelner Datenpakete verändert und somit verfälschte bzw. ungültige Daten zwischen den einzelnen Kommunikationsteilnehmern ausgetauscht werden.

8.2.2. Positionsbestimmung

Störung Aufgrund von sporadisch einfallendem Sonnenlicht kann die Positionsbestimmung gestört werden, sodass die Rennstrecke, die Fahrzeuge oder die Hindernisse nicht mehr korrekt erkannt werden können.

Ausfall Ein Ausfall der Positionsbestimmung könnte durch einen Fehler im Videostream provoziert werden. Dies wäre z.B. durch einen Ausfall der Kamera, ein Bewegen der Rennstrecke oder des Kameragestells oder das Manipulieren der Rennstreckenmarker nach Start des Rennbetriebs möglich. In diesen Fällen könnten keine verlässlichen Fahrzeug- oder Hindernisposen mehr berechnet werden.

Fehler Die Positionsbestimmung könnte durch einen Softwarefehler abstürzen oder sich in einem Zustand befinden, in welchem dieselben Fahrzeugdatenpakete wiederholt ausgesendet werden. Des Weiteren wäre ein vertauschen der Fahrzeug IDs während des Betriebs ein möglicher Fehlerfall. Ein weiterer Fehler in der Software könnte dazu führen, dass die Positionsbestimmung ihre Realzeitanforderung von 10 ms nicht einhalten kann.

8.2.3. Systemsteuerung

Störung Durch eine Überlastung des Rechners kann es zu Verzögerungen bei der Übertragung von Befehlen von der Systemsteuerung zu den anderen Systemkomponenten kommen.

Ausfall Durch einen vollständigen Ausfall der Systemsteuerung kann der Rennbetrieb nicht mehr beendet oder ein Not-Aus Befehl ausgelöst werden.

Fehler 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. Die Systemsteuerungssoftware könnte durch einen Softwarefehler abstürzen. Im Fehlerfall der Systemsteuerung kann der Rennbetrieb nicht mehr beendet oder kein Not-Aus Befehl ausgelöst werden.

8.2.4. Control-Unit

Störung Bei der internen Kommunikation mit der CarControl oder auch der Ansteuerung des Fahrzeugs über die Funkstrecke kann es zu kurzzeitigen Störungen kommen.

Ausfall Die Control-Unit oder die CarControl können abstürzen. Dadurch kann ein reibungsloser Ablauf nicht mehr gewährleistet werden. Bricht die Verbindung zur CarControl ab, können keine Steuersignale oder Befehle an das Fahrzeug gesendet werden.

Fehler Bei der Berechnung der Regelungswerte für die Längs- bzw. Querführung können Fehler auftreten, sodass falsche oder nicht plausible Regelungswerte an das Fahrzeug übermittelt werden. Des Weiteren kann es vorkommen, dass die Control-Unit ihre Realzeitanforderung von maximal 10 ms im Laufe des Rennens nicht mehr einhalten kann. Das würde dazu führen, dass das Fahrzeug nicht schnell genug auf Abweichungen auf der Rennstrecke reagieren kann. Es kann ebenfalls vorkommen, dass das Fahrzeug mit der Fahrbahnbegrenzung, einem weiteren Fahrzeug oder einem Hindernis kollidiert und anschließend unkontrollierbar ist.

8.3. Sicherheitsanforderungen

Basierend auf den Schadenszenarien lassen sich folgende Sicherheitsanforderungen an das System bzw. die Subsysteme ableiten.

SA1 Ein Ausfall der Systemsteuerungssoftware muss erkannt werden.

SA2 Ein Ausfall der Positionsbestimmung muss erkannt werden.

SA3 Die Positionsbestimmung muss ihre Realzeitanforderung von 10 ms einhalten.

SA4 Die Positionsbestimmung muss einen Ausfall der Kamera erkennen.

SA5 Die Positionsbestimmung muss eine Veränderung der Rennstrecke erkennen.

SA6 Die Positionsbestimmung muss eine Veränderung der Fahrzeug IDs erkennen.

SA7 Ein Ausfall der Control-Unit muss erkannt werden.

SA8 Die Control-Unit muss ihre Realzeitanforderung von 10 ms erfüllen.

SA9 Die Control-Unit muss Kollisionen erkennen.

SA10 Die Control-Unit muss falsch berechnete Regelungswerte erkennen.

SA11 Ein Ausfall der Netzwerkkommunikation muss erkannt werden.

SA12 Fehlerhafte Netzwerkpakete müssen erkannt werden.

SA13 Die Positionsbestimmung, Control-Unit und Systemsteuerung müssen tolerant gegenüber Störungen sein.

SA14 Die Positionsbestimmung, Control-Unit und Systemsteuerung müssen im Fehlerfall eindeutige Fehlercodes senden.

8.4. Sicherer Fehlerzustand

Definition Als sicherer Fehlerzustand wird in dieser Projektgruppe der Zustand verstanden, indem das Fahrzeug und die Umgebung in einem Fehlerfall mit hoher Wahrscheinlichkeit nur minimalen Schaden nehmen werden. Da die Fahrzeuge die einzigen dynamischen Komponenten in diesem System sind, müssen sie im Fehlerfall schnellstmöglich zum Stehen gebracht werden. Dies kann einerseits durch das Senden von Bremssignalen bzw. eines Ausrollbefehls über die Fernbedienung oder andererseits durch ein deaktivieren der Fernbedienung und ein damit einhergehendes ausrollen der Fahrzeuge erreicht werden. Einmal in den sicheren Zustand überführt, kann das System nur durch einen vollständigen Neustart wieder in Betrieb genommen werden. Im Rahmen der Projektgruppe wird das deaktivieren der Fernbedienung nicht weiter betrachtet, da es einen tieferen eingriff in die Hardware bedeuten würde und dies aus Zeitgründen nicht realisierbar ist.

Überführung Da das Fahrzeug über die CarControl, welche eine Komponente der Control-Unit ist, angesteuert wird (siehe Abschnitt 5.3.1 und Abbildung 5.11) ist es notwendig, dass die Ausführung der Befehle zur Überführung in den sicheren Fehlerzustand über die Control-Unit abgewickelt werden. Das System kann über verschiedene Mechanismen in den sicheren Fehlerzustand überführt werden. Erstens durch das manuelle Betätigen des Not-Aus Befehls oder das Senden eines sicherheitskritischen Fehlercodes über die Systemsteuerung. Beide Nachrichtentypen werden von der Control-Unit empfangen dann über die CarControl ausgeführt. Zweitens werden auch sicherheitskritische Fehlercodes von der Positionsbestimmung bzw. weiteren Control-Units empfangen und verarbeitet. Drittens können die Komponenten einer Control-Unit eine Fehlfunktion erkennen und so einen Not-Halt auslösen. Sollten viertens einzelne Komponenten der Control-Unit nicht mehr reagieren ist die CarControl in der Lage selbständig das Not-Halt Signal auszuführen und so die Überführung in den sicheren Fehlerzustand einzuleiten. Dies geschieht über einen hardwareseitigen Watchdog bzw. den Not-Halt Taster auf der CarControl.

Sollte auch die CarControl nicht mehr reagieren, ist es nicht mehr möglich mit dem Fahrzeug zu interagieren. In diesem Fall kann man nur noch die CarControl und somit auch die Fernbedienung von der Stromversorgung trennen und dadurch den Transmitter deaktivieren. In diesem Fall bleibt das Fahrzeug ebenfalls stehen. Da es sich bei dem *dNaNo*-System um ein Blackbox System handelt, ist dieser Fehlerfall nicht anders abzufangen.

8.5. Sicherheitskonzept

Ausgangspunkt für das Sicherheitskonzept zur Realisierung der in Abschnitt 8.3 aufgestellten Sicherheitsanforderungen ist der Rennbetrieb des *RCCARSng* Systems, da nur in diesem Betriebszustand ein Schaden an den Fahrzeugen und der Umgebung entstehen kann. Das Sicherheitskonzept greift an mehreren Stellen, welche im Folgenden erläutert werden.

8.5.1. Systemebene

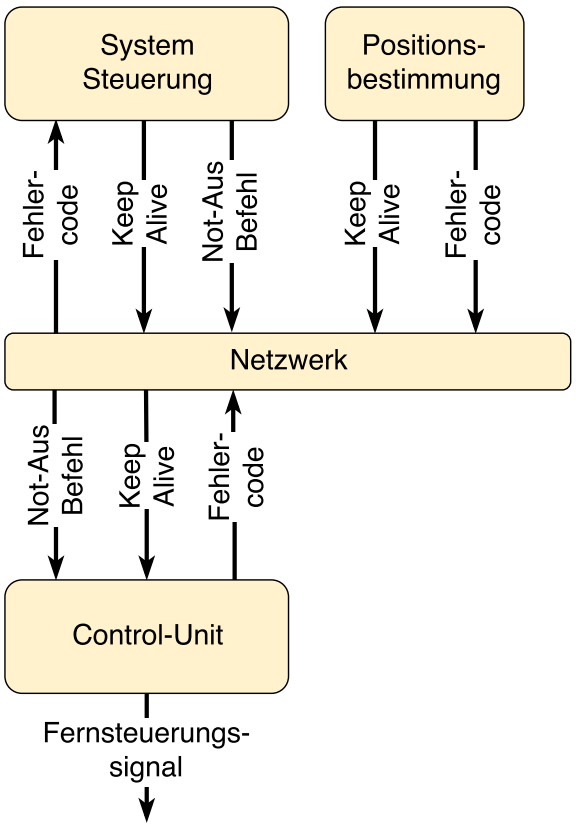


Abbildung 8.2.: Systemweites Sicherheitskonzept (Übersicht).

Auf Systemebene wird mithilfe von einem KeepAlive Signal zwischen der Systemsteuerung und den Control-Units sichergestellt, dass ein Ausfall der Systemsteuerung detektiert wird (SA1). Bleibt das KeepAlive über einen gewissen Zeitraum 1000 ms aus, wird in der Control-Unit ein Nothalt ausgelöst. Dieser Mechanismus greift ebenfalls bei einem Ausfall der Netzwerkkommunikation (SA11). Gleichzeitig werden für eine einfachere Fehleranalyse Fehlercodes zwischen einzelnen Subsystemen ausgetauscht (SA14, siehe auch Abschnitt 8.5.5). In Abbildung 8.2 ist dieser Zusammenhang grafisch dargestellt.

8.5.2. Positionsbestimmung

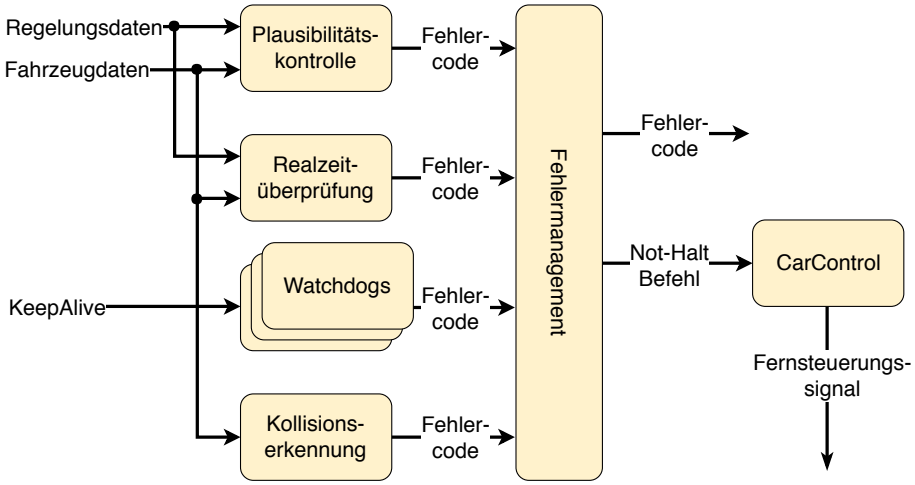


Abbildung 8.3.: Sicherheitskonzept Positionsbestimmung (Übersicht).

Im Rennbetrieb überwacht die Positionsbestimmung aktiv die Kameraverbindung und sendet im Fehlerfall einen sicherheitskritischen Fehlercode an die anderen Systemkomponenten (SA4). Des Weiteren werden der Zustand der Rennstrecke (SA5) und der Fahrzeuge (SA6) kontinuierlich überwacht und im Fehlerfall wird ebenfalls eine Fehlercode abgesetzt (SA14). Damit die Realzeitanforderung überprüft werden kann, wird in einer eigenen Komponente kontinuierlich die Zeit zwischen dem Eintreffen eines neuen Frames der Kamera bis zum Versenden der Positionsdaten von Fahrzeugen und Hindernissen berechnet und im Fall der Verletzung der Realzeitanforderung eine entsprechende Fehler ID versendet (SA3). Die Zusammenhänge des Sicherheitskonzepts der Positionsbestimmung sind in Abbildung 8.3 dargestellt. Die Anforderung bzgl. der Robustheit bei ausbleibenden Positionsdaten (SA13) wird in der Control-Unit realisiert, da dort auf Grundlage der Positionsdaten entsprechende Regelungswerte berechnet und so Vorkehrungen getroffen werden können.

8.5.3. Control-Unit

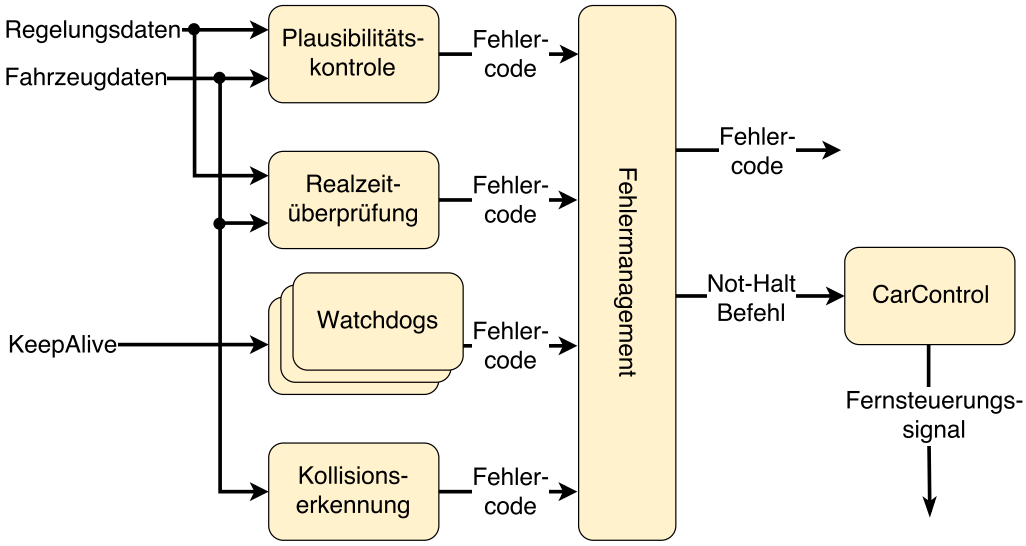


Abbildung 8.4.: Sicherheitskonzept Control-Unit (Übersicht).

Innerhalb der Control-Unit werden die von der Positionsbestimmung eingehenden Positionsdaten inkl. ihrer Zeitstempel, sowie die Zeitstempel der eigens berechneten Regelungswerte für die Längs- und Querführung stetig überprüft und so sichergestellt, dass die Realzeitanforderung der Control-Unit und auch der Positionsbestimmung eingehalten werden (SA8 und SA3). Dieser Mechanismus erlaubt ebenfalls die Erkennung einer nicht mehr funktionstüchtigen Positionsbestimmung (SA2), da damit folglich die Realzeitanforderung der Control-Unit verletzt wird. Des Weiteren werden alle berechneten Werte einer Plausibilitätsüberprüfung unterzogen (SA10) und schließlich sicherheitsrelevante Komponenten bzgl. ihrer Funktionsfähigkeit (SA7) mit einem Watchdog und einem KeepAlive Signal überprüft. Die möglichen Kollisionen mit der Bande, Hindernissen oder auch anderen Fahrzeugen werden in einer eigenen Komponente zur Kollisionserkennung behandelt (SA9). Sollten diese Mechanismen eine Verletzung der Spezifikation erkennen, wird dies über einen Fehlercode einer eigenen Fehlermanagementkomponente mitgeteilt, welche dann, je nach Toleranzbegriff (SA13) entweder einen Not-Halt über die CarControl einleitet oder einen Fehlercode an die weiteren Subsysteme weiterleitet. Diese Zusammenhänge sind in Abbildung 8.4 grafisch dargestellt.

8.5.4. Netzwerkkommunikation

Die Sicherheitsanforderung SA12 bzgl. der Datenintegrität der Netzwerkdatenpakete wird über ein eigenständiges CRC Feld in jedem Datenpaket realisiert. Dies ermöglicht die Erkennung von beschädigten Datenpaketen und kann von jedem Subsystem durchgeführt werden.

8.5.5. Fehlercodes

Ein Teil des in Abschnitt 8.5 entworfenen Sicherheitskonzepts ist der Einsatz von Fehlerdatenpaketen mit einem spezifischen Fehlercode, welche von den einzelnen Systemkomponenten erzeugt werden (SA14). Die Fehlercodes setzen sich jeweils aus sechs Ziffern zusammen, wobei jede Stelle eine bestimmte Bedeutung besitzt. So 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 Kodierung der Fehlercodes ist in Abbildung 8.5 schematisch dargestellt. 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.

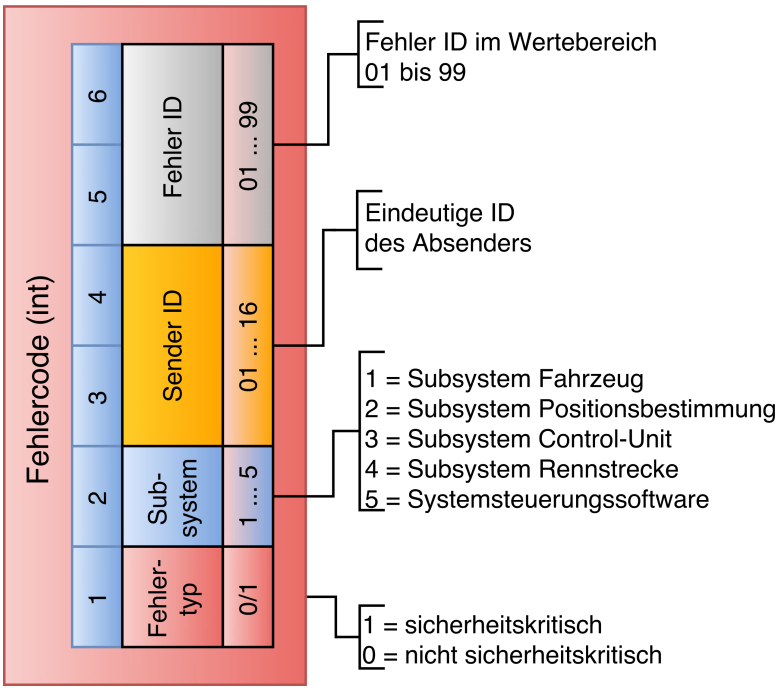


Abbildung 8.5.: Codierung des Fehlercodes.

9. Systemarchitektur und Schnittstellen

In diesem Kapitel wird zunächst die Systemarchitektur des gesamten Systems beschrieben. Nach der Architekturbeschreibung folgt eine detaillierte Beschreibung und Auflistung aller Schnittstellen im System. Dabei werden zunächst die Schnittstellen zwischen den fünf Subsystemen beschrieben, auf die eine Beschreibung der Schnittstellen innerhalb der Control-Unit folgt.

9.1. Systemarchitektur

Das vorliegende System gliedert sich in die fünf Subsysteme Fahrzeug (Subsystem 1), Positionsbestimmung (Subsystem 2), Control-Unit (Subsystem 3), Rennstrecke (Subsystem 4) und Systemsteuerungssoftware (Subsystem 5). Eine gesamte Übersicht inklusive aller Bestandteile der Subsysteme ist in Abbildung 9.1 aufgeführt. Im Folgenden werden Funktion und Aufbau der einzelnen Subsysteme beschrieben.

9.1.1. Subsystem 1: Fahrzeug

Das Fahrzeug besteht aus einer Karosserie, einer Aufhängung, vier Reifen, einem Akku, einer Platine und zwei Motoren. Auf der Karosserie befinden sich zudem Reflexionsmarker, die der Identifizierung dienen.

9.1.2. Subsystem 2: Positionsbestimmung

Die Positionsbestimmung bestimmt Position und Ausrichtung der Fahrzeuge und Hindernisse auf der Rennstrecke. Es wird in Hard- und Softwarekomponenten aufgeteilt. Die Hardware besteht aus einer Halterung, bestehend aus Gestell und Deckenhalterung, einer Kamera, einem Objektiv, einem Infrarotfilter, Infrarotstrahlern und einem Rechner. Mithilfe des Infrarotstrahlers werden Fahrzeuge und Hindernisse, die sich auf der Rennstrecke befinden, mit Infrarotlicht beleuchtet. Die Kamera ist in

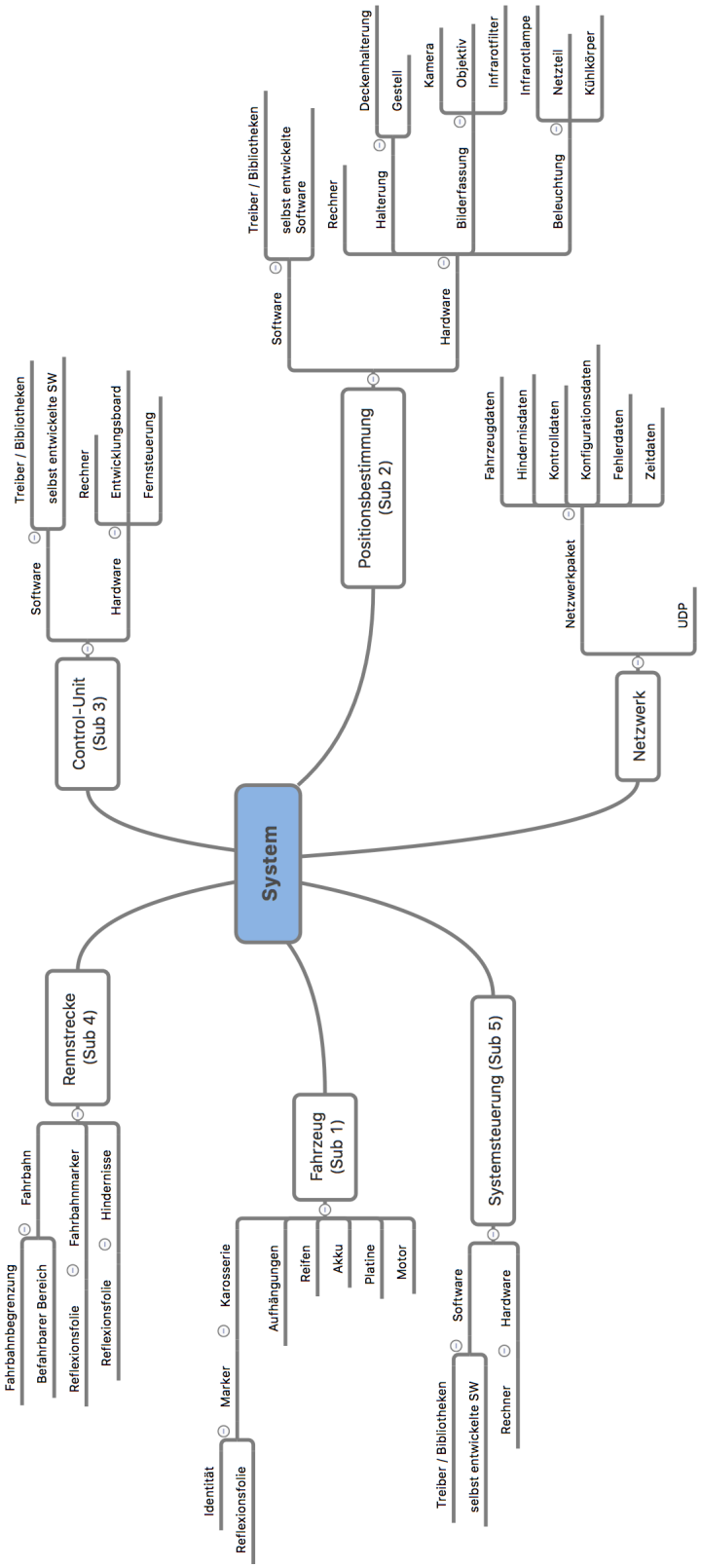


Abbildung 9.1.: Schematische Darstellung der Systemarchitektur

der Lage, die Fahrzeuge und Hindernisse bildlich zu erfassen und erhöht mittels Infrarotfilter die Sichtbarkeit der Reflexionsmarker von Fahrzeugen und Hindernissen. Durch das verwendete Weitwinkelobjektiv kann die gesamte Rennstrecke und somit Fahrzeuge und Hindernisse an jeder Position auf der Rennstrecke erfasst werden. Die auf dem Rechner selbst entwickelte Positionsbestimmungssoftware (siehe Abschnitt 5.2.2) berechnet mithilfe von zusätzlichen Bibliotheken aus dem erfassten Bild der Kamera die Fahrzeug- und Hindernispositionen.

9.1.3. Subsystem 3: Control-Unit

Die Control-Unit unterteilt sich ebenfalls in Hard- und Softwarekomponenten. Sie besteht hardwareseitig aus einem Rechner, einem Entwicklungsboard und einer Fernsteuerung für die Verbindung zum Fahrzeug. Die Software übernimmt die Fahrzeugsteuerung. Sie unterteilt sich intern in die Bestandteile Vorplanung, Kollisionserkennung, Regelungsmodule, CarControl, Konfigurationsparameter, Kommunikationsinterface und Management (siehe Abbildung 9.2). Für jedes Fahrzeug, welches auf der Rennstrecke fährt, existiert eine eigene Control-Unit. Eine nähere Beschreibung der einzelnen Komponenten folgt in den nächsten Abschnitten. Jeder Komponente ist dabei eine Modulnummer zugewiesen, welche Tabelle A.4 zu entnehmen ist.

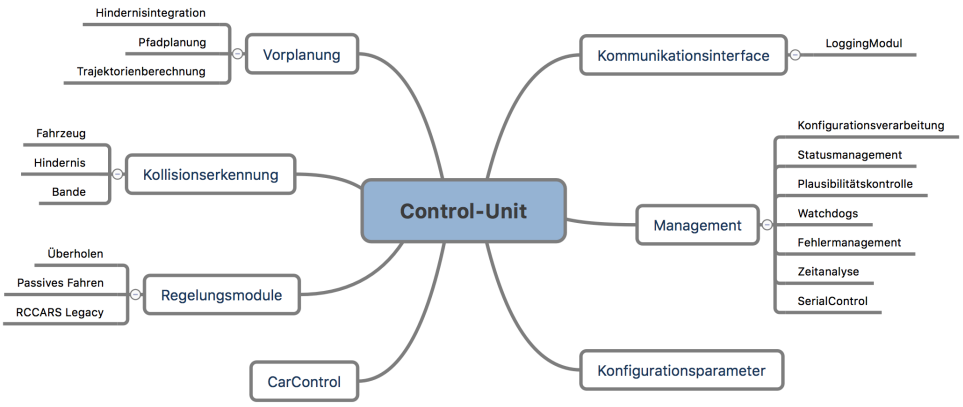


Abbildung 9.2.: Schematische Darstellung der Control-Unit Architektur

Management

Das Management ist die zentrale Steuereinheit der Control-Unit. Es ist innerhalb des Moduls weiter hierarchisch aufgeteilt und enthält die folgenden Komponenten:

Fehlermanagement Das Fehlermanagement verwaltet jegliche Fehler, die in der Control-Unit auftreten können. Es regelt allerdings auch die globale Fehlerkommunikation mit den Komponenten außerhalb der Control-Unit.
Im Falle, dass ein internes Modul der Control-Unit einen Fehlerzustand feststellt, sendet dieses einen internen Fehlercode an das Fehlermanagement. Diese sind in der internen Fehlercodetabelle (siehe Anhang C.2) nachzulesen. Es wird zwischen Quelle und Kritikalität des Fehlers unterschieden. Bei kritischen Fehlern wird ein Not-Halt-Befehl versendet, um das System so möglichst schnell in einen sicheren Fehlerzustand zu führen. In jedem Fall wird der Fehlercode über das Kommunikationsinterface und das Netzwerk an die Systemsteuerungssoftware weitergeleitet. Diese entscheidet, wie mit dem Fehler weiter umgegangen wird und kann bei kritischen Fehlern einen Not-Aus-Befehl versenden, der das gesamte System in einen sicheren Zustand überführt. Die Fehler sind gemäß der Beschreibung aus Abschnitt 8.5.5 zusammengesetzt. Weitere Fehlerinformationen werden an das Statusmanagement gesendet, um über dieses die Control-Unit gegebenenfalls in einen Fehlerzustand zu versetzen.

Eingabe	Fehlercode
Ausgabe	Externer Fehlercode
	Fehlerinformationen
	Heartbeat
	Not-Aus Befehl

Tabelle 9.1.: Ein- und Ausgabeschnittstellen des Fehlermanagements.

Konfigurationsverarbeitung Nachdem alle Konfigurationsdaten und -parameter eingelesen wurden, stellt die Konfigurationsverarbeitung für die einzelnen Module die entsprechenden Parameter bereit. Weiterhin erstellt die Konfigurationsverarbeitung aus den Hindernisdaten von der Positionsbestimmung und Rennstreckeninformationen statische Umgebungsdaten, welche Informationen über die Rennstrecke und Hindernisse zusammengefasst bereitstellen.

Eingabe	Hindernisdaten
	Konfigurationsdaten
	Konfigurationsparameter
Ausgabe	Kollisionserkennungsparameter
	Regelungsparameter
	Statische Umgebungsdaten
	Vorplanungsparameter

Tabelle 9.2.: Ein- und Ausgabeschnittstellen der Konfigurationsverarbeitung.

Plausibilitätskontrolle In der Plausibilitätskontrolle werden sämtliche Daten, die zur Laufzeit generiert werden, auf ihre Validität überprüft. So werden die Netzwerkpakete, die an die Control-Unit gesendet wurden, aber auch Daten, die innerhalb der Control-Unit generiert werden, geprüft.

Eingabe	Hindernisdaten
	Fahrzeugdaten
	Konfigurationsdaten
	Kontrolldaten
Ausgabe	Fehlercode
	Heartbeat

Tabelle 9.3.: Ein- und Ausgabeschnittstellen der Plausibilitätskontrolle.

Realzeitüberprüfung Die Realzeitüberprüfung überprüft, ob die Fahrzeug- und Regelungsdaten innerhalb eines definierten Zeitintervalls erneut eingegangen sind. Zudem wird in der Realzeitüberprüfung berechnet, wie lange ein Regelungsmodul braucht, um zu einem Fahrzeugdatenpaket die entsprechenden Regelungsdaten zu berechnen. Wenn die Arbeitszeit einer der Regelungskomponente außerhalb des definierten Intervalls liegt, sendet die Realzeitüberprüfung einen entsprechenden Fehlercode an das Fehlermanagement.

Eingabe	Fahrzeugdaten
	Regelungsdaten
Ausgabe	Fehlercode
	Heartbeat

Tabelle 9.4.: Ein- und Ausgabeschnittstellen der Realzeitüberprüfung.

Renndatenberechnung Die Renndatenberechnung analysiert die Fahrzeugdaten und berechnet aus ihnen die aktuellen Geschwindigkeiten, Rundenanzahl und Rundengeschwindigkeiten der Fahrzeuge. Diese Daten werden als Realzeitdaten ausgegeben.

Eingabe	Fahrzeugdaten
Ausgabe	Fehlercode
	Heartbeat
	Realzeitdaten

Tabelle 9.5.: Ein- und Ausgabeschnittstellen der Renndatenberechnung.

SerialControl Diese Komponente bekommt als Eingabeparameter die Regelungsdaten oder bei einem systemkritischen Fehler einen Not-Aus Befehl. Aus den Regelungsdaten berechnet die SerialControl die Stellgrößen für die CarControl und leitet diese als Regelungswerte über die serielle Schnittstelle weiter. Sollte ein Not-Aus Befehl empfangen worden sein, wird dieser ebenfalls an die CarControl weitergeleitet, um das Fahrzeug anzuhalten. Zudem tauscht die SerialControl Kontrollbefehle mit der CarControl aus.

Eingabe	Not-Aus Befehl
	Kontrollbefehl
	Regelungsdaten
Ausgabe	Fehlercode
	Heartbeat
	Kontrollbefehl
	Not-Aus Befehl
	Regelungswerte

Tabelle 9.6.: Ein- und Ausgabeschnittstellen der SerialControl.

Statusmanagement Über das Statusmanagement werden die Status der einzelnen Module geregelt. Dabei können Statusbefehle über das Statusmanagement verschickt werden, um Zustände der Control-Unit zu wechseln und damit einzelne Module zu de- oder aktivieren.
Das Statusmanagement durchläuft zur Steuerung einen Zustandsautomaten, der in Abb. 9.3 dargestellt ist. Die einzelnen Zustände werden nun beschrieben:

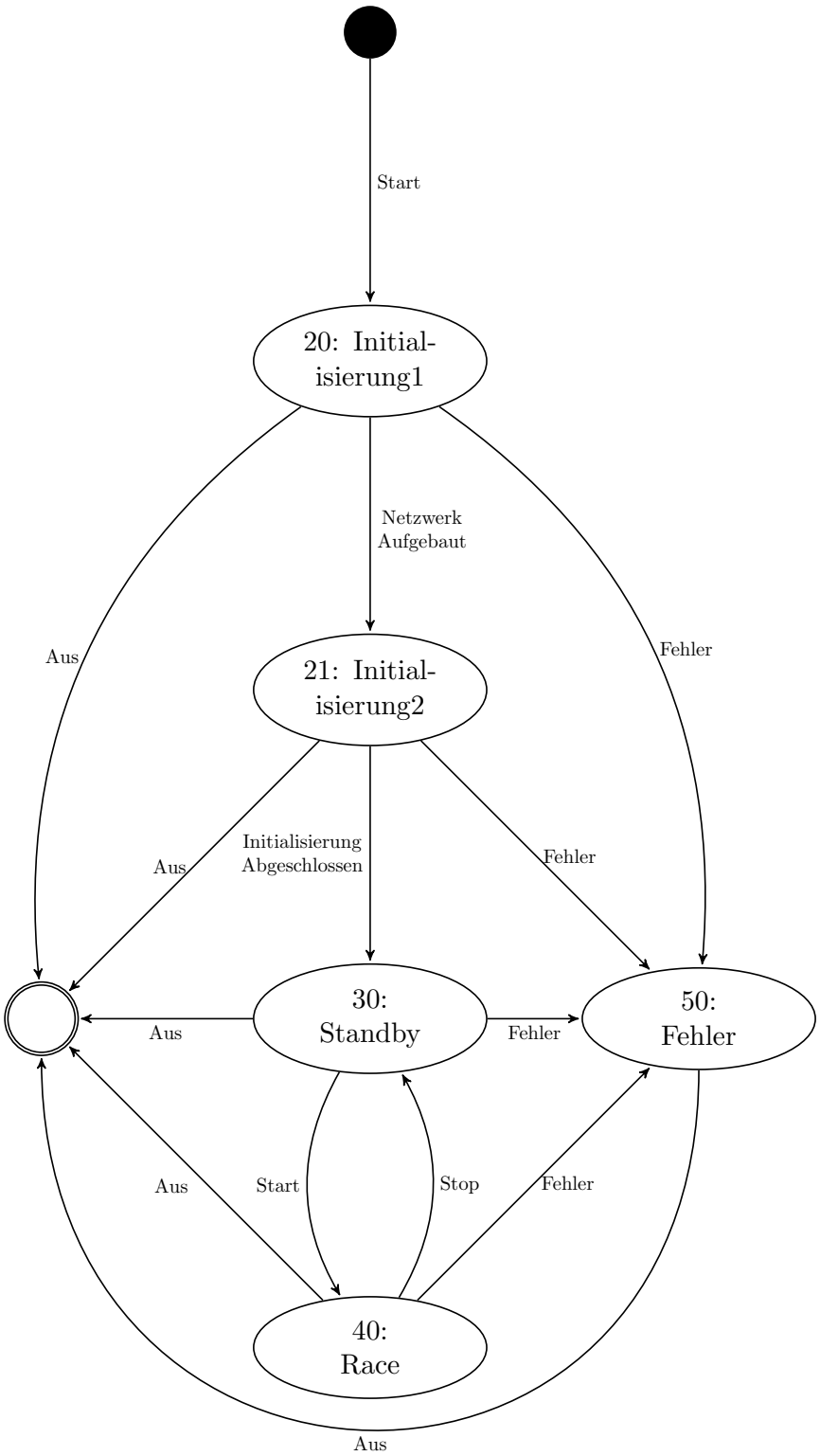


Abbildung 9.3.: Zustandsautomat der Control-Unit.

Initialisierung 1

In der ersten Initialisierungsphase werden Watchdog und Fehlermanagement als Thread gestartet. Anschließend beginnt die Konfigurationsverarbeitung mit dem Einlesen der Config-Files, dann beginnt das Kommunikationsinterface, die UDP-Netzwerkkommunikation mit den anderen Subsystemen aufzubauen und die SerialControl nimmt Verbindung zur CarControl auf. Sind diese Operationen erfolgreich abgeschlossen, teilt die Control-Unit diesen Abschluss der ersten Initialisierungsphase der Systemsteuerungssoftware über das Netzwerk mit.

Initialisierung 2

In der zweiten Initialisierungsphase wartet die Control-Unit auf Konfigurationsparameter von der Systemsteuerungssoftware. Sind diese eingetroffen, beginnt die Konfigurationsverarbeitung zusammen aus den Konfigurationsdaten, Parameter für Vorplanung, Kollisionserkennung und Regelungsmodule zu erstellen. Stehen die Parameter bereit, wird die Vorplanung gestartet, anschließend wird in den Standby-Zustand gewechselt.

Standby

In diesem Zustand wartet die Control-Unit auf den Befehl, den Rennmodus zu starten.

Race

Im Race-Zustand erfolgt die Rennsteuerung des Fahrzeugs. Dazu werden die Kollisionserkennung, die Plausibilitätskontrolle, die Realzeitüberprüfung, die Regelungsmodule, die Renndatenberechnung und die SerialControl als Thread gestartet.

Fehler

Falls ein sicherheitskritischer Fehler auftritt, wechselt die Control-Unit in den Fehlerzustand. Hier wird das Fahrzeug zum Stehen gebracht.

Aus

Im Aus-Zustand wird das System heruntergefahren. Alle laufenden Threads, sowie die Kommunikation zur CarControl und zum Netzwerk werden geschlossen.

Eingabe	Konfigurationsdaten
	Kontrolldaten
	Fehlerinformation
Ausgabe	Statusbefehl
	Fehlercode
	Heartbeat

Tabelle 9.7.: Ein- und Ausgabeschnittstellen des Statusmanagements.

Watchdogs Der Watchdog dient zur Überwachung von Komponenten. Dazu werden von einzelnen Komponenten regelmäßig sogenannte Heartbeats versendet, um einen intakten Arbeitszustand zu signalisieren. Jeder Heartbeat setzt einen moduleigenen Timer zurück. Sollte dieser Timer nicht zurückgesetzt werden, wird ein sicherheitskritischer Fehlercode an das Fehlermanagement gesendet, der das System beendet. Durch dieses Verfahren kann überprüft werden, ob einzelne Komponenten abgestürzt sind oder sich noch in einem funktionsfähigen Zustand befinden.

Eingabe	Heartbeat
Ausgabe	Fehlercode

Tabelle 9.8.: Ein- und Ausgabeschnittstellen des Watchdogs.

Vorplanung

In der Vorplanung erfolgt die Berechnung von Referenztrajektorien und zugehörigen Geschwindigkeits- und Beschleunigungsprofilen für die Regelung. Für die Berechnung der Trajektorie zieht die Vorplanung statische Umgebungsdaten heran, die Rennstreckendaten und Hindernisdaten enthalten. Um die fahrdynamischen Profile berechnen zu können, werden Fahrzeugdaten verwendet. Darüber hinaus nutzt die Vorplanung eigens für sie zusammengestellte Vorplanungsparameter. Die berechnete Trajektorie und die fahrdynamischen Profile gibt die Vorplanung zusammengefasst in Form von Planungsdaten aus. Falls bei der Berechnung Fehler auftreten, versendet die Vorplanung ebenfalls Fehlercodes. Während der Berechnung signalisiert die Vorplanung mithilfe von Heartbeats, dass sie noch arbeitet.

Eingabe	Fahrzeugdaten
	Statische Umgebungsdaten
	Vorplanungsparameter
Ausgabe	Fehlercode
	Heartbeat
	Planungsdaten

Tabelle 9.9.: Ein- und Ausgabeschnittstellen der Vorplanung.

Kollisionserkennung

Die Kollisionserkennung stellt während des Rennbetriebs fest, ob Kollisionen aufgetreten sind oder Sicherheitsabstände nicht eingehalten werden. Dazu verwendet sie statische Umgebungsdaten, um den Verlauf der Rennstrecke und Position der Hindernisse zu erhalten. Zusätzlich nutzt sie Fahrzeugdaten für die Positionserkennung und Kollisionserkennungsparameter, um auf konfigurierte Sicherheitsabstände zugreifen zu können. Zu den Objekten, mit denen Fahrzeuge kollidieren oder zu nah kommen können, zählen Hindernisse, der Fahrbahnrand, sowie andere Fahrzeuge. Sicherheitsabstandsverletzungen oder Kollisionen gibt die Kollisionserkennung mithilfe von Fehlercodes aus.

Eingabe	Fahrzeugdaten
	Statische Umgebungsdaten
Ausgabe	Fehlercode
	Heartbeat

Tabelle 9.10.: Ein- und Ausgabeschnittstellen der Kollisionserkennung.

Regelungsmodule

Die Regelungsmodule der Control-Unit umfassen alle für die Erfüllung des Szenarios benötigten Regelungen der Quer- und Längsführung. Ein Regelungsmodul erhält Planungsdaten, Regelungsparameter, statische Umgebungsdaten, sowie aktuelle Fahrzeugdaten. Hieraus berechnet es je nach ausgewähltem Fahrverhalten die entsprechenden Regelungsdaten, welche sie abschließend ausgibt. Durch die Regelungsmodule wird unter anderem gewährleistet, dass Fahrzeuge nicht gegen andere Objekte fahren, ausbrechen oder andere Fahrzeuge überholen können.

Eingabe	Fahrzeugdaten
	Planungsdaten
	Regelungsparameter
	Statische Umgebungsdaten
Ausgabe	Fehlercode
	Heartbeat
	Regelungsdaten

Tabelle 9.11.: Ein- und Ausgabeschnittstellen der Regelungsmodule.

Kommunikationsinterface

Durch das Kommunikationsinterface werden alle ein- und ausgehenden Nachrichten der Control-Unit bearbeitet und verschickt. Ein zusätzlicher Bestandteil ist ein Logging Modul. Dieses ist in der Lage, ausgewählte Ereignisse mitzuschreiben und auszugeben.

CarControl

Die CarControl ist für das Versenden der Fernsteuerungssignale an das Fahrzeug zuständig. Hierzu empfängt sie die Regelungswerte, Kontrollbefehle und gegebenenfalls einen Not-Halt Befehl. Durch Kontrollbefehle kommuniziert die CarControl mit der SerialControl.

Eingabe	Kontrollbefehl
	Not-Halt Befehl
	Regelungswerte
Ausgabe	Fernsteuerungssignal
	Kontrollbefehl

Tabelle 9.12.: Ein- und Ausgabeschnittstellen der CarControl.

9.1.4. Subsystem 4: Rennstrecke

Die Rennstrecke besteht aus zusammensteckbaren Streckensegmenten, die sich in einem befahrbaren Bereich und die Fahrbahnbegrenzung aufteilen lassen. Zudem ist die Rennstrecke mit Fahrbahnmarkern an den Ecken für die Positionsbestimmung eingegrenzt, welche den Reflexionsmarkern der Fahrzeuge ähneln. Des Weiteren gehören Hindernisse auf der Rennstrecke zu Subsystem 4, die ebenfalls mittels Reflexionsfolie markiert sind.

9.1.5. Subsystem 5: Systemsteuerungssoftware

Damit ein Administrator das System überwachen und bedienen, beziehungsweise zwischen den Systemzuständen wechseln kann, wurde die Systemsteuerungssoftware implementiert. Diese bietet dem Administrator ein Benutzerinterface, mit der unter anderem das Starten und Stoppen des Rennbetriebs und das Auslesen von Informationen wie beispielsweise Fehlercodes möglich ist. Auch die Fahrverhalten und Mindestdurchschnittsgeschwindigkeiten können ausgelesen werden. Die Systemsteuerungssoftware besteht aus einer alleinstehenden Software und kommuniziert mit der Positionsbestimmung und den Control-Units.

9.2. Schnittstellen

In diesem Abschnitt werden die Schnittstellen der einzelnen Subsysteme, die in der Architektur aufgeführt sind, abschnittweise aufgezeigt und erläutert. Dabei wird sowohl auf die Schnittstellen zwischen den Subsystemen, als auch auf die internen Schnittstellen der Control-Unit eingegangen. Eine Übersicht aller Schnittstellen zwischen den Subsystemen ist in Abbildung 9.4 dargestellt. Neue Schnittstellen sind grün gefärbt, während veränderte Schnittstellen gelb gefärbt sind. Schnittstellen, die ohne Veränderung übernommen wurden, sind in grau dargestellt.

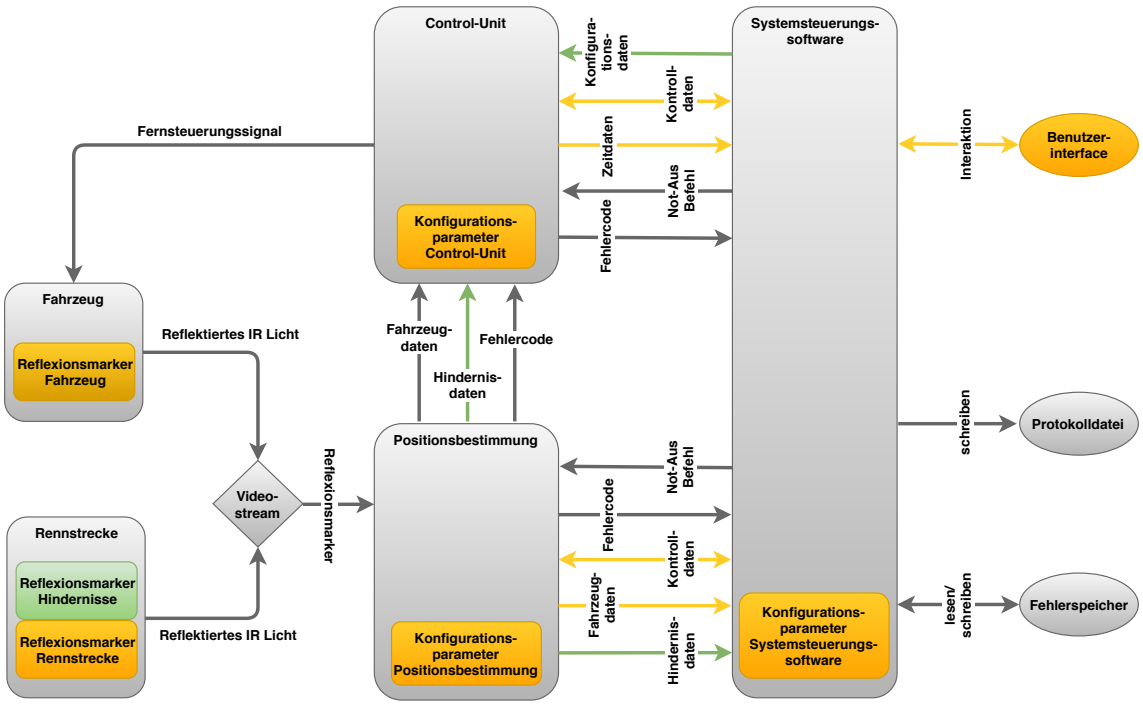


Abbildung 9.4.: Schematische Darstellung der Schnittstellen des Systems

9.2.1. Subsystem 1: Fahrzeug

Das Fahrzeug empfängt die analogen 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.

Schnittstellen

- **Fernsteuerungssignal**
  Das Fahrzeug ist mit der Control-Unit drahtlos verbunden und empfängt analoge Fernsteuerungssignale über die CarControl der zugehörigen Control-Unit.
- **Reflektiertes Infrarotlicht**
  Das Infrarotlicht, welches auf die Reflexionsmarker fällt, wird im Einfallswinkel zur Lichtquelle zurück reflektiert. Neben der Lichtquelle ist eine Kamera befestigt, welche das reflektierte Infrarotlicht erfasst. Die Reflexionsmarker sind in einem festgelegten Muster angeordnet (siehe Abbildung 9.5). Das vordere und hintere Rechteck geben die Fahrzeugrichtung an. Die bis zu vier mittig platzierten Quadrate geben die Identität an. Über die Identität lassen sich bis zu 16 Fahrzeugen codieren.

Eingabeschnittstellen

Eingabeschnittstellen - Subsystem Fahrzeug				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Fernsteuerungs-signal	—	—	—	—
Infrarotlicht	—	—	—	—

Tabelle 9.13.: Eingabeschnittstellen des Subsystems Fahrzeug.

Ausgabeschnittstellen

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

Tabelle 9.14.: Ausgabeschnittstellen des Subsystems Fahrzeug.

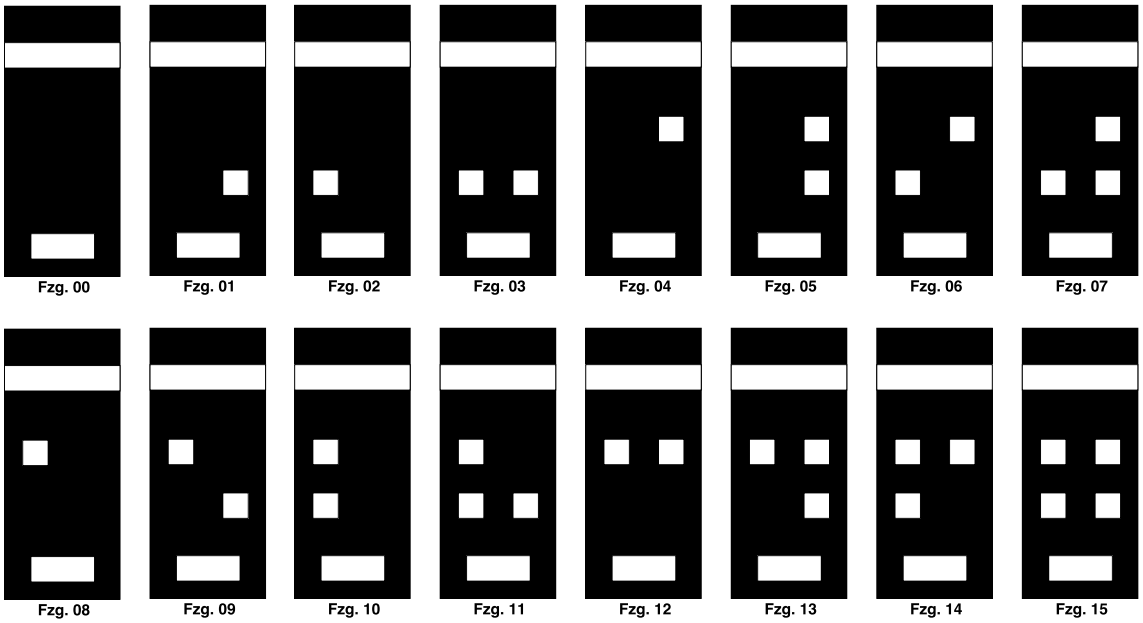


Abbildung 9.5.: Fahrzeugmuster. Anordnung der Marker.

9.2.2. Subsystem 2: Positionsbestimmung

Die Positionsbestimmung ist für die Erfassung der Fahrzeugpose und dessen Weiterleitung an die Control-Unit und die Systemsteuerung verantwortlich. Für den Betrieb der Positionsbestimmung werden Konfigurationsparameter und ein kontinuierlicher Videostream der Kamera benötigt. Des Weiteren soll die Positionsbestimmung Kontrolldaten erhalten, um die Fahrzeug- und Hinderniserkennung starten oder stoppen und gegebenenfalls einen Not-Aus auslösen zu können.
Mithilfe des Videostreams ist die Positionsbestimmung in der Lage, die Pose der Fahrzeuge, sowie die Position der Hindernisse zu ermitteln und - falls nötig - einen Fehlercode auszugeben. Die Lage der Fahrzeuge und Hindernisse wird dabei bezüglich des Koordinatensystems angegeben, das in Abschnitt 5.4.1 und der zugehörigen Abbildung 5.9 beschrieben wird. Die Fahrzeugpose wird durch den Ausrichtungswinkel des Fahrzeugs bzgl. der x-Achse gegeben. Er beträgt also 0 ° beziehungsweise 360 ° , falls das Fahrzeug in Richtung der x-Achse ausgerichtet ist; 90 ° falls das Fahrzeug in Richtung der y-Achse ausgerichtet ist; 180 °, falls das Fahrzeug entgegen der x-Achse ausgerichtet ist und 270 °, falls das Fahrzeug entgegen der y-Achse ausgerichtet ist. Die definierten Ein- und Ausgabeschnittstellen, sowie die Konfigurationsparameter der Positionsbestimmung werden im Folgenden beschrieben.

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.

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. Eine Übersicht der Konfigurationsdatei ist im Anhang unter B.1 zu finden.

Schnittstellen

- **Fahrzeugdaten**
  Damit die Control-Unit die aktuelle Position 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 erkannten Fahrzeuge und stellt diese als Double-Wert (double) im Bereich 0.00 bis 236.0 bzw. 0.00 bis 236.0 dar. Zusätzlich werden die Ausrichtungswinkel sämtlicher erkannten 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. Alle Daten werden in einem jeweiligen Array versendet. Die Position im Array gibt an, zu welchem Fahrzeug die Positionsdaten und die Ausrichtung gehören.

- **Fehlercode**
  Die Positionsbestimmung ist in der Lage, 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 C.1 aufgelistet ist.

- **Hindernisdaten**
  Damit die Control-Unit die aktuellen Positionen der Hindernisse kennt und aufgrund dieser Positionen eine neue Trajektorie berechnen kann, um das Fahrzeug über die Strecke zu steuern, wird ein Hindernisdatenpaket an die Control-Unit und die Systemsteuerungssoftware gesendet. Hierbei wird zunächst die Anzahl der Hindernisse ermittelt und als Integer-Wert (int) übergeben. Es können maximal 16 Hindernisse erkannt werden, wobei 0 für kein Hindernis und 16 für die maximale Anzahl der Hindernisse steht. Die Positionsbestimmung ermittelt die X und Y-Koordinaten der vier Eckpunkte aller erkannten Hindernisse und stellt diese als Double-Wert (double) im Bereich 0.00 bis 236.0 bzw. 0.00 bis 236.0 in cm dar. Die Werte werden jeweils in einem Array versendet.

- **Kontrolldaten**
  Für Kommunikation der Systemzustände mit der Systemsteuerungssoftware können Kontrolldaten versendet oder empfangen werden. Die Kontrolldaten beinhalten die ID der Positionsbestimmung, die auf den Integerwert 200 codiert ist, und einen Systemzustand, der ebenfalls als Integerwert codiert ist. Die Codierung der einzelnen Systemzustände ist in Tabelle A.2 dargestellt. Die Kontrolldaten werden von der Systemsteuerungssoftware verwendet, um einen Zustandswechselbefehl mitzuteilen. Ebenfalls teilt die Positionsbestimmung den aktuellen Zustand durch Kontrolldaten mit.

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

Eingabeschnittstellen

Eingabeschnittstellen - Subsystem Positionsbestimmung				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Videostream	Videostream	CameraStream IOManager	object	—
Kontrolldaten	Positionsbestimmung ID	int	200	—
	Wechsel der Systemzustände	int	> 0	—

Tabelle 9.15.: Eingabeschnittstellen des Subsystems Positionsbestimmung.

Ausgabeschnittstellen

Ausgabeschnittstellen - Subsystem Positionsbestimmung				
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1	—
	X-Koordinate	double[16]	0.0 ... 236.0	cm
	Y-Koordinate	double[16]	0.0 ... 236.0	cm
	Ausrichtung	double[16]	0.0 ... 360.0	°
	Zeitstempel der Fahrzeugpose	timeval	—	—
Hindernisdaten	Gefundene Hindernisse	int	0 ... 16	—
	X-Koordinate Punkt 1	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 1	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 2	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 2	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 3	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 3	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 4	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 4	double[16]	0.0 ... 236.0	cm
Kontrolldaten	Positionsbestimmung ID	int	200	—
	Aktueller Zustand	int	> 0	—
Fehlercode	Fehlercode	int	> 0	—

Tabelle 9.16.: Ausgabeschnittstellen des Subsystems Positionsbestimmung.

9.2.3. 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 Schnittstellen werden in externe und interne Schnittstellen unterteilt.

Externe Schnittstellen der Control-Unit

Externe Schnittstellen sind Schnittstellen zu anderen Subsystemen. Eine Übersicht der externen Schnittstellen, ist in neben den internen Schnittstellen in Abbildung 9.6 dargestellt.
Die Control-Unit empfängt Fahrzeug- und Hindernisdaten sowie von der Positionsbestimmung. Von der Systemsteuerungssoftware können Kontrolldaten, Konfigurationsdaten und ein Not-Aus Befehl empfangen werden. Fehlercodes können von weiteren Control-Units, der Positionsbestimmung und der Systemsteuerungssoftware empfangen werden.
Die Control-Unit liest einmalig zum Start Konfigurationsparameter ein.

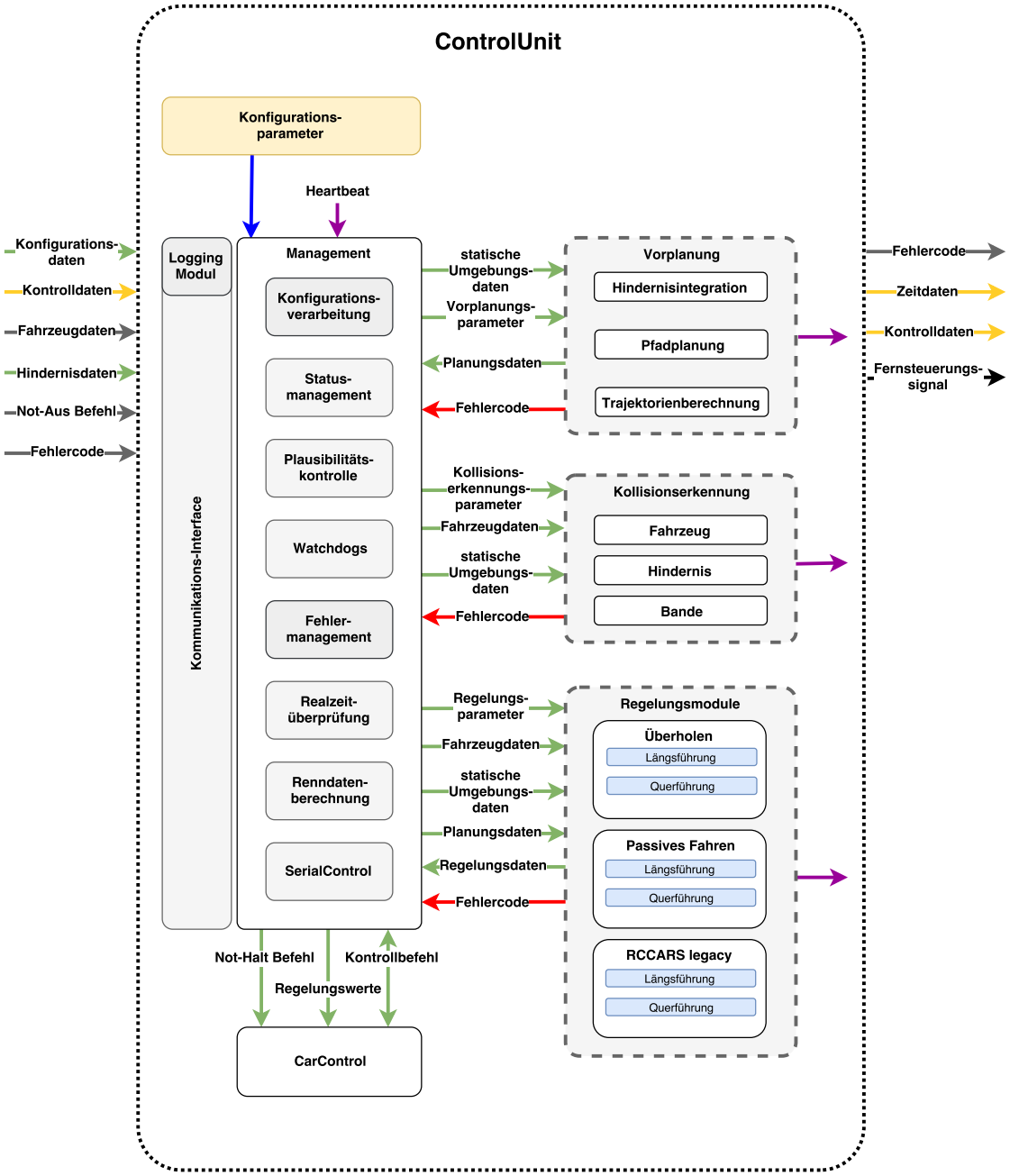


Abbildung 9.6.: Schematische Darstellung der CU mit internen und externen Schnittstellen

Sie sendet ein digitales Fernsteuerungssignal an ein Fahrzeug, welches auch den Not-Halt Befehl beinhalten kann. Fehlercodes, Zeitdaten und Kontrolldaten werden über das Netzwerk an alle Netzwerkteilnehmer gesendet.
Im folgenden werden die Schnittstellen näher beschrieben.

- **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 sämtlich erkannter Fahrzeuge als Double-Wert (double) jeweils im Bereich 0.00 cm bis 236.0 cm. Zusätzlich werden die Ausrichtungswinkel sämtlicher erkannter Fahrzeuge als Double-Wert (double) im Bereich von 0.00 ° bis 360.0 ° , die Verfügbarkeit der Fahrzeuge und ein Zeitstempel vom Datentypen timeval, für die Feststellung des Alters der Fahrzeugposen und als Fingerabdruck des Netzwerkdatenpakets, mitgeteilt.

- **Hindernisdaten**
  Die Control-Unit empfängt Hindernisdaten von der Positionsbestimmung und kann so eine Trajektorie für das Fahrzeug berechnen. Hierzu wird die Anzahl der gefundenen Hindernisse als Integer (int) versendet, wobei maximal 16 Hindernisse erkannt werden können. Die X- und Y-Koordinaten der vier Eckpunkte eines Hindernisses werden jeweils in einem Array als Double-Werte (double) verschickt. Die Koordinaten der Hindernisse werden im Bereich von 0.00 cm bis 236.0 cm. angegeben.

- **Konfigurationsdaten**
  Von der Systemsteuerungssoftware werden verschieden Parameter als Konfigurationsdaten gesendet, die die Regelung der Fahrzeuge beeinflussen.
  So wird jeder Control-Unit ein Fahrzeug zugewiesen. Hierzu wird ein Array aus Integern verschickt, wobei der Index des Arrays für die ID der jeweiligen Control-Unit steht und der Wert an dem Index der ID eines Fahrzeugs entspricht. Der Wert -1 steht dabei für keine Zuweisung.
  Nach dem gleichen Prinzip werden die Fahrverhalten für die verschiedenen Control-Units angegeben. Hierbei steht der Wert des Arrays plus 301 jeweils für ein Fahrverhalten. Der Wert „0“ steht für kein Fahrverhalten, der Wert „1“ für „passives Fahren“, der Wert „2“ für „Überholen“ und der Wert „3“ für „Überholt werden“.
  Auch die Mindestdurchschnittsgeschwindigkeiten werden als Array verschickt. Hierbei wird die Geschwindigkeit als Double-Wert (double) in m/s angegeben.

- **Kontrolldaten**
  Für das Wechseln der Systemzustände muss die Systemsteuerungssoftware Kontrolldaten senden. Diese Daten bestehen aus einer ID einer Control-Unit,

die als Integer (int) versendet wird, und einem weiteren Integer, der den Wechsel des Systemzustands einer Control-Unit veranlasst. Die einzelnen Systemzustände sind in Tabelle A.2 aufgeführt. Um einen Zustandswechsel mitzuteilen, werden die Kontrolldaten ebenfalls von der Contro-Unit gesendet. Dabei wird auch die ID der jeweiligen Control-Unit übertragen.

- **Fehlercode**
  Damit die Control-Unit auf sicherheitskritische Fehlfunktionen des Systems schnellstmöglich reagieren kann, empfängt diese Fehlercodes der einzelnen Subsysteme. Der Fehlercode wird als Integer-Wert (int) empfangen. Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in C.1 aufgelistet ist. Fehlercodes werden auch von der Control-Unit gesendet, falls während des Betriebs ein Fehler auftritt.

- **Not-Halt Befehl**
  Damit das Fahrzeug im Fehlerfall sofort die Längs- und Querführung auf die Nullstellung 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 den DA-Wandler geschickt. Der Wert lautet hierbei 0x4E (char „N“). Auslöser für den Not-Halt Befehl können intern erkannte Fehler als auch eingehende Fehlercodes oder der Not-Aus Befehl sein.

- **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 von der Systemsteuerungssoftware entsandt und ist die Reaktion auf einen Fehlercode.

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

- **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, sowie der ID der jeweiligen Control-Unit an die

Systemsteuerungssoftware übermittelt. Diese Schnittstelle ist im System vorhanden. Sie wird jedoch nicht mehr benutzt, da diese Überprüfung innerhalb der Control-Unit stattfindet.

Eingabeschnittstellen

Eingabeschnittstellen - Control-Unit				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1	—
	X-Koordinate	double[16]	0.0 ... 236.0	cm
	Y-Koordinate	double[16]	0.0 ... 236.0	cm
	Ausrichtung	double[16]	0.0 ... 360.0	°
	Zeitstempel der Fahrzeugposen	timeval	—	—
Hindernisdaten	Gefundene Hindernisse	int	0 ... 16	—
	X-Koordinate Punkt 1	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 1	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 2	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 2	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 3	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 3	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 4	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 4	double[16]	0.0 ... 236.0	cm
Konfigurations-daten	Fahrzeugzuweisung zu CU	int[16]	-1 ... 15	—
	Fahrverhalten der CU	int[16]	0 ... 3	—
	Mindestdurchschnitts-geschw. der CU	double[16]	0.0 ... 4.0	m/s
Kontrolldaten	Control-Unit ID	int	301 ... 316	—
	Wechsel der Systemzustände	int	> 0	—
Fehlercode	Fehlercode	int	> 0	—
Not-Aus Befehl	Wechsel in den Fehlerzustand	int	1	—

Tabelle 9.17.: Eingabeschnittstellen der Control-Unit.

Ausgabeschnittstellen

Ausgabeschnittstellen - Control-Unit				
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Digitales Fernsteuerungs-signal	—	—	—	—
Kontrolldaten	Control-Unit ID	int	301 ... 316	—
	Aktueller Zustand	int	> 0	—
Fehlercode	Fehlercode	int	> 0	—
Zeitdaten	Control-Unit ID	int	301 ... 316	—
	Zeitstempel der Fahrzeugpose	timeval	—	—
	Zeitstempel der gesendeten Regelungswerte	timeval	—	—

Tabelle 9.18.: Ausgabeschnittstellen der Control-Unit.

Interne Schnittstellen der Control-Unit

In diesem Abschnitt werden die internen Schnittstellen der Control-Unit detailliert beschrieben. Eine Übersicht der internen Schnittstellen, ist in Abbildung 9.7 dargestellt. Hierbei wird vom Management Modul ausgegangen. Eine vollständige Darstellung dieses Moduls ist in Abbildung 9.6 zu sehen.

- **Konfigurationsparameter**
  Im Rahmen einer Vorinitialisierung der Control-Unit müssen Konfigurationsparameter für die Initialisierung und den Betrieb festgelegt werden. Hierzu gehören u.a. die ID der Control-Unit, die Schnittstelle zur CarControl, Parameter für die Netzwerkkommunikation, sowie notwendige Parameter für alle internen Module der Control-Unit. Die gesamten Parameter werden aus einem Config-File während der Initialisierung der Control-Unit ausgelesen. Die Konfigurationsparameter sind im Anhang unter B.2 zu finden.
- **Fahrzeugdaten**
  Fahrzeugdaten beinhalten Informationen über bis zu 16 Fahrzeuge. Diese beinhalten die X- und Y-Koordinaten der erkannten Fahrzeuge als Double-Wert (double) im Bereich 0.00 cm bis 236.0 cm, sowie die Ausrichtungswinkel der erkannten Fahrzeuge als Double-Wert (double) im Bereich von 0.00° bis 360.0°. Die Verfügbarkeit der Daten wird durch einen Integer (int) deutlich gemacht.

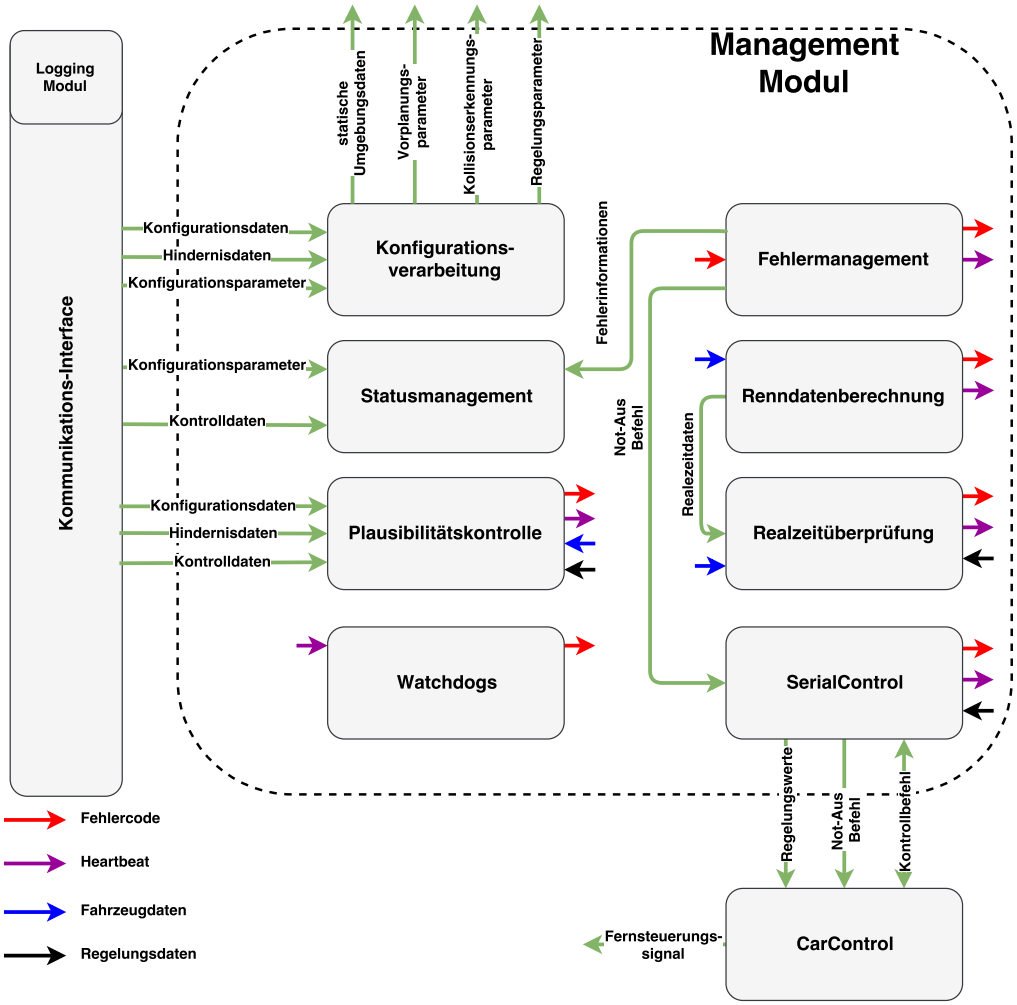


Abbildung 9.7.: Schematische Darstellung der internen Schnittstellen anhand des Management Moduls der Control-Unit.

Fahrzeugdaten			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Verfügbare Fahrzeuge	int[16]	0 / 1	—
X-Koordinate	double[16]	0.0 ... 236.0	cm
Y-Koordinate	double[16]	0.0 ... 236.0	cm
Ausrichtung	double[16]	0.0 ... 360.0	°

Tabelle 9.19.: Zusammenfassung - Fahrzeugdaten.

- **Vorplanungsparameter**
  Für die Vorplanungsebene gibt es zwei Arten von Parametern: Fahrzeugparameter und Simulationsparameter. Fahrzeugparameter umfassen die minimale und maximale Beschleunigung a_{min}, a_{max}, die maximale Querbeschleunigung q_{max}, sowie Maximalgeschwindigkeit v_{max} und Durchschnittsgeschwindigkeit $vAverage$ des Fahrzeugs. Zu den Simulationsparametern zählen die Anzahl n

der Stützstellen, um Rennstrecke und Trajektorie in Stützstellen einzuteilen, die Anzahl n_{print} der Stützstellen zur grafischen Darstellung der Simulationsumgebung, die Anzahl der Iterationen $numberOfIterationen$ in der Trajektorienberechnung, sowie der Sicherheitsabstand sd. Um dem Fahrverhalten eine entsprechende Trajektorie zu berechnen werden weitere Parameter benötigt. Eine gesamte Auflistung der Vorplanungsbene ist in Tabelle 9.20 dargestellt.

- **Regelungsparameter**
  Die Regelungsparameter (siehe Tabelle 9.21) enthalten wie die Vorplanungsparameter neben den Fahrzeugparametern, bestehend aus der minimalen und maximalen Beschleunigung a_{min}, a_{max},den Radstand L dem minimalen δ_{min} und maximalen δ_{max} Lenkwinkel des Fahrzeugs, auch Gewichte w_v, w_x, w_y, w_ψ und Kosten für die Regelung ru_1, ru_2, Anzahl der Schritte für die Regelung N, Schrittweite dieser T, Latenz $latenz$ und Trennpunkte für den Trajektorienwechsel $splitPoint0y$, $splitPoint0x$, $splitPoint1x$, $splitPoint1y$. Zusätzlich wird eine Fahrzeug-ID $FzID$ übergeben.

Vorplanungsparameter			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Fahrzeugparameter			
Minimale Beschleunigung a_{min}	double	-10.0 ... 10.0	m/s^2
Maximale Beschleunigung a_{max}	double	-10.0 ... 10.0	m/s^2
Maximale Querbeschleunigung q_{max}	double	-10.0 ... 10.0	m/s^2
Maximale Geschwindigkeit v_{max}	double	-10.0 ... 10.0	m/s
Simulationsparameter			
Durchschnittsgeschwindigkeit *vAverage*	double	-10.0 ... 10.0	m/s
Anzahl der Stützstellen n	int16	0 ... 10000	—
Anzahl der Stützstellen Grafiken *printn*	int16	0 ... 10000	—
Anzahl der Iterationen *NumberOfIterations*	int	0 ... 255	—
Sicherheitsabstand *safetyDistanceKruemmungsOpt*	double	0.0 ... 1.0	m
1. Knotenpunkt Überholtrajektorie *splitPoint*0*x*	double	0.0 ... 2.36	m
1. Knotenpunkt Überholtrajektorie *splitPoint*0*y*	double	0.0 ... 2.36	m
2. Knotenpunkt 1 Überholtrajektorie *splitPoint*1*x*	double	0 .0 ... 2.36	m
2. Knotenpunkt Überholtrajektorie *splitPoint*1*y*	double	0.0 ... 2.36	m
Simulationsparameter			
Länge von Streckensegmenten *lengthStraightSegments*	double	0.0 ... 0.5	m
Halbe Breite Fahrbahnaussenränder *widthOutsideCenterline*	double	0.0 ... 0.5	m
Breite Fahrbahnrand *widthBorder*	double	0.0 ... 0.5	m
Radius Kurven Mittellinie *radianCenterlineCurvesegments*	double	0.0 ... 0.5	m
Radius kleine Kurven Fahrbahnrand *smallRadian*	double	0.0 ... 0.5	m
Größe der Rennstrecke x *roadsizeX*	double	0.0 ... 2.36	m
Größe der Rennstrecke y *roadsizeY*	double	0.0 ... 2.36	m
Speicherort Matlabfehler *LoggingPath*	string	—	—

Tabelle 9.20.: Auflistung aller Vorplanungsparameter.

Regelungsparameter			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Fahrzeugparameter			
Minimale Beschleunigung a_{min}	double	-10.0 ... 10.0	m/s^2
Maximale Beschleunigung a_{max}	double	-10.0 ... 10.0	m/s^2
Minimaler Lenkwinkel δ_{min}	double	-π ... π	rad
Maximaler Lenkwinkel δ_{max}	double	-π ... π	rad
Radstand	double	0.062	m
MPC-Paramter			
Gewicht w_v	double	0.0...1.0	—
Gewicht w_x	double	0.0...1.0	—
Gewicht w_y	double	0.0...1.0	—
Gewicht w_ψ	double	0.0...1.0	—
Kosten ru_1	double	0.0...1.0	—
Kosten ru_2	double	0.0...1.0	—
1. Knotenpunkt Überholtrajektorie $splitPoint0x$	double	0.0 ... 2.36	m
1. Knotenpunkt Überholtrajektorie $splitPoint0y$	double	0.0 ... 2.36	m
2. Knotenpunkt 1 Überholtrajektorie $splitPoint1x$	double	0 .0 ... 2.36	m
2. Knotenpunkt Überholtrajektorie $splitPoint1y$	double	0.0 ... 2.36	m
Latenz $latenz$	int	0...100	—
Schrittweite T	double	0.0...0.5	s
Schrittanzahl N	int	0...40	—

Tabelle 9.21.: Zusammenfassung - Regelungsparameter.

- **Kontrollbefehl**
  Der Kontrollbefehl ist ein Signal um die CarControl anzusteuern. Es wird in Form eines Characters gesendet. Dabei sind drei verschiedene Befehle möglich. „C“ wird an die CarControl gesendet, sobald vom Management ein Verbindung aufgebaut wurde. Mit einem „M“ kann die CarControl in den Betriebsmodus versetzt werden. Um die CarControl zu reseten, muss ein „R“ verschickt werden.

Kontrollbefehl			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Kontrollbefehl - Prüfungsanfrage d. Verbindung	byte	0x43	—
Kontrollbefehl - Wechseln in Betriebsmodus	byte	0x4D	—
Kontrollbefehl - Auslösen eines SoftReset	byte	0x52	—

Tabelle 9.22.: Zusammenfassung - Kontrollbefehl.

- **Kollisionserkennungsparameter**
  Der einzige Parameter, der an die Kollisionserkennung übergeben wird, ist der Sicherheitsabstand in Metern. Sobald dieser Abstand zur Fahrbahnbegrenzung, zu Hindernissen oder zu anderen Fahrzeugen unterschritten wird, wird eine Kollision erkannt und diese Information von der Kollisionserkennung an das Management geliefert.

Kollisionserkennungsparameter			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Sicherheitsabstand	double	0.0 ... 1.0	m

Tabelle 9.23.: Zusammenfassung - Kollisionserkennungsparameter.

- **Heartbeat**
  Der Heartbeat ist ein Integer-Wert von 1, der von der Vorplanung, der Kollisionserkennung, der Fehlermanagement, der Realzeitüberprüfung und der Plausibilitätskontrolle versendet wird. Der Heartbeat sorgt dafür, dass gewährleistet werden kann, dass keine Komponente für längere Zeit unbemerkt abgetürzt ist.

Heartbeat			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Heartbeat	int	1	—

Tabelle 9.24.: Zusammenfassung - Heartbeat.

- **Fehlercode**

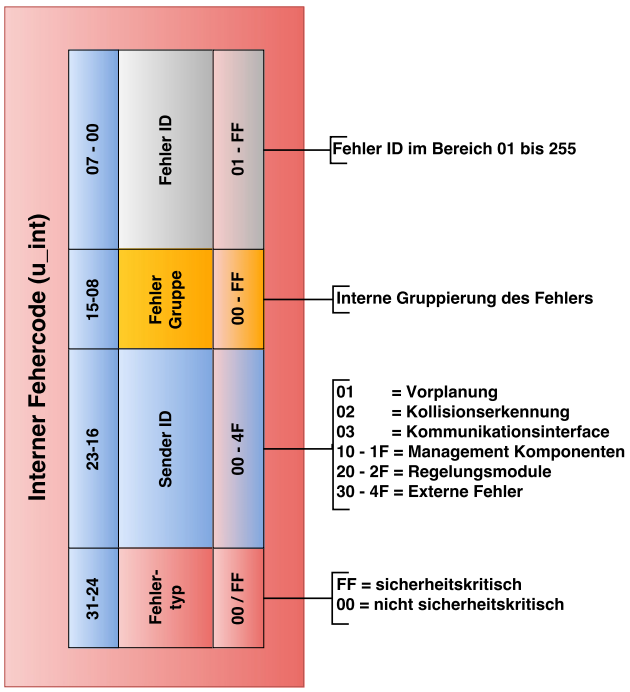


Abbildung 9.8.: Aufbau der Control-Unit internen Fehlercodes.

Fehlercodes werden immer dann ausgesendet, wenn in einer Komponente ein Fehler auftritt. Dabei wird zwischen internen Fehlercodes der Control-Unit und externen Fehlercodes, die mit den anderen Subsystemen ausgetauscht werden, unterschieden. Damit die Control-Unit auf sicherheitskritische Fehlfunktionen des Systems schnellstmöglich reagieren kann, empfängt das Fehlermanagement innerhalb der Managementkomponente der Control-Unit die Fehlercodes der einzelnen Module. Die internen Fehlercodes werden als Integer-Wert (int) versendet. Um die Fehlercodes eindeutig zu klassifizieren, werden die Bits aufgeteilt und jeweils unterschiedlich genutzt.
Zur Codierung werden unsigned Integer verwendet. Diese haben eine Bitbreite von 4 Byte. Somit lassen sich Fehlercodes leicht in verschiedene Blöcke unterteilen. Eine Übersicht die Blöcke der internen Fehlercodes ist in Abbildung 9.8 zusehen. Der eigentliche Fehler wird im Bereich Bit 0 bis Bit 7 codiert. Glei-

che Fehlerarten können zu Gruppen zusammengefasst werden. Diese Gruppen werden im Bereich von Bit 8 bis Bit 15 codiert. Jedes interne Modul der Control-Unit besitzt, wie in Abschnitt 9.1.3 beschrieben, eine eigene ID. Diese wird im Bereich von Bit 16 bis Bit 23 codiert. Ob der codierte Fehler kritisch ist oder nicht, wird im letzten Bereich von Bit 24 bis Bit 31 codiert.
Jeder Fehlercode entspricht einem fest definierten Fehler, welcher in der Fehlercodetabelle im Anhang unter C.2 aufgelistet ist. Der Fehlercode wird zusätzlich als Integer-Wert (int) an alle Netzwerkteilnehmer übermittelt.

Fehlercode			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Fehlercode	u_int32	0 ... 2^{32}	—

Tabelle 9.25.: Zusammenfassung - Fehlercode.

- **Planungsdaten**
  In den Planungsdaten sind Informationen zur Referenztrajektorie angegeben. Dazu ist die Trajektorie nach ihrer Bogenlänge durch s parametrisiert. Für diskrete Parameter sind die Punktkoordinaten, der Vektor in Fahrtrichtung, die Krümmung, sowie geplante Geschwindigkeit, Beschleunigung und Ausrichtung des Fahrzeugs angegeben. Die Daten sind zeilenweise, aufsteigend nach s gelistet.

Planungsdaten			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Format der Zeile: $s, x, y, \phi, k, v, a, s_r$			
Trajektorienparameter: s	double	0.0 ... 10.0	m
x-Koordinate: x	double	0.0 ... 10.0	m
y-Koordinate: y	double	0.0 ... 10.0	m
Ausrichtung: ϕ	double	0.0 ... 2.0π	rad
Krümmung: k	double	-5.0 ... 5.0	1/m
Zielgeschwindigkeit: v	double	0.0 ... 5.0	m/s
Zielbeschleunigung: a	double	-10.0 ... 10.0	m/s^2
Wegparameter: s_r	double	0.0 ... 10.0	m

Tabelle 9.26.: Zusammenfassung - Planungsdaten.

- **Regelungswerte**
  Damit das Fahrzeug die korrekte Längs- und Querführung umsetzen kann, werden Regelungswerte an die CarControl gesendet. Die Regelungswerte werden als Integer-Wert für jeweils die Längs- und Querführung bereitgestellt.

Beide Werte haben einen Wertebereich von 0 bis 4095. Um eine Trennung der Befehle zu ermöglichen, wird vor jedem Datenwort ein Erkennungssymbol und nach jedem Datenwort ein Trennsymbol gesendet. Diese Symbole werden als char übertragen. Das Erkennungssymbol ist entweder ein „S“ oder ein „T“. Das Trennsymbol ist der char „CR“ (*„Carriage Return“*)

Regelungswerte			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Längsführung	int	0 ... 4095	-
Querführung	int	0 ... 4095	-
Erkennungssymbol	char	„S“, „T“	-
Trennsymbol	char	„CR“	-

Tabelle 9.27.: Zusammenfassung - Regelungswerte.

- **Not-Halt Befehl**
  Damit das Fahrzeug die Längs- und Querführung sofort auf die Nullstellung bringt und somit anhält, kann ein Not-Halt Befehl empfangen werden. Der Not-Halt Befehl wird in Form einer fest definierten Bytefolge „N“ empfangen.

Not-Halt-Befehl			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Not-Halt-Befehl	char	„N“	—

Tabelle 9.28.: Zusammenfassung - Not-Halt-Befehl.

- **Regelungsdaten**
  Die Regelungsdaten sind die Werte, die von der Regelungsebene berechnet wurden. Dabei handelt es sich um einen Beschleunigungswert zwischen $-10\,\mathrm{m/s^2}$ und $10\,\mathrm{m/s^2}$ und einem Lenkwinkel in Bogenmaß.

Regelungsdaten			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Beschleunigung	double	-10.0 ... 10.0	$\mathrm{m/s^2}$
Lenkwinkel	double	0.0 ... 2.0 π	rad

Tabelle 9.29.: Zusammenfassung - Regelungsdaten.

- **Statische Umgebungsdaten**
  Die statischen Umgebungsdaten beinhalten die Umgebungsinformationen, die sich während des Rennbetriebes nicht ändern. Zu den statischen Umgebungsdaten zählen zum Einen die Rennstrecke, zum Anderen die Hindernisse. Sie werden im CSV-Format ausgetauscht und haben folgenden Aufbau:

1. Infozeile:
   In der Infozeile ist festgelegt, in welchen Zeilen Rennstreckendaten und Hindernisdaten zu finden sind.

Format der Zeile:	r_0, r_1, h_0, h_1
Startzeile Rennstreckendaten:	r_0
Endzeile Rennstreckendaten:	r_1
Startzeile Hindernisdaten:	h_0
Endzeile Hindernisdaten:	h_1

2. Rennstreckendaten:
   Die Rennstreckendaten bestehen aus einer Auflistung von Rennstreckensegmenten. Es gibt ein lineares Streckensegment, sowie drei Kurvensegmente. Das Format der Zeilen ist:

Zeilenformat für lineare Segmente	$t, p0_x, p0_y, p1_x, p1_y, 0, wi, wa$
Zeilenformat für Kurvensegmente	$t, pm_x, pm_y, \phi_0, \phi_1, R, wi, wa$

   Die einzelnen Einträge sind in der Tabelle 9.30 abgebildet.

Rennstreckendaten		
Beschreibung	**Wertebereich**	**Einheit**
Streckentyp: t (0: Geradensegment, 1: vorderes Kurvensegment, 2: mittleres Kurvensegment, 3: hinteres Kurvensegment)	0, 1 ,2 ,3	—
Startpunkt: $SP = (p0_x, p0_y)$	0.0 .. 10.0	m
Endpunkt: $EP = (p1_x, p1_y)$	0.0 .. 10.0	m
Abstand Fahrbahnrand in innere Richtung: wi	0.0 .. 3.0	m
Abstand Fahrbahnrand in äußere Richtung: wa	0.0 .. 3.0	m
Mittelpunkt der Kurvenelemente: $MP = (pm_x, pmy)$	0.0 .. 10.0	m
Startwinkel der gesamten Kurve: ϕ_0	-2.0π .. 2.0π	rad
Startwinkel der gesamten Kurve: ϕ_1	-2.0π .. 2.0π	rad
Radius von MP zu Mittellinie: R	0.0 .. 3.0	m
Abstand Mittellinie Fahrbahnrand in innere Richtung: r	0.0 .. 3.0	m

Tabelle 9.30.: Komponenten der Rennstreckendaten.

3. Hindernisdaten:
   Für jedes Hindernis werden in einer Zeile die Eckpunkte angegeben. Format der Hinderniszeile:

   Eckpunktkoordinaten: e_0x, e_0y, ... , e_nx, e_ny

   Die Punkte e_i und e_{i+1} sind dabei miteinander verbunden. Der letzte Punkt e_n ist mit e_1 verbunden, damit das Hindernis in sich geschlossen ist.

Hindernisdaten			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
x-Koordinate eines Eckpunktes: e_ix	double	0.0 .. 10.0	m
y-Koordinate eines Eckpunktes: e_iy	double	0.0 .. 10.0	m

Tabelle 9.31.: Komponenten der Hindernisdaten.

- **Renndaten**
  Die Renndaten beinhalten aktuelle Informationen aller Fahrzeuge zu Durchschnittsgeschwindigkeiten, Momentangeschwindigkeiten und Rundenanzahl.

Renndaten			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Durchschnittsgeschwindigkeit	double	0.0 ... 5.0	m/s
Momentangeschwindigkeit	double	0.0 ... 5.0	m/s
Rundenanzahl	int	0 ... 500	—

Tabelle 9.32.: Zusammenfassung - Renndaten.

9.2.4. Subsystem 4: Rennstrecke

Das Subsystem Rennstrecke ist ein Subsystem, welches ausschließlich aus Hardwarekomponenten besteht. Über Reflexionsmarker sollen sowohl die Rennstrecke, als auch die Hindernisse, die sich auf der Strecke befinden, ständig von der Positionsbestimmung erfasst werden können. Es hat definierte Ein- und Ausgabeschnittstellen, welche im Folgenden beschrieben werden.

Schnittstellen

- **Infrarotlicht**
  Die Rennstrecke besitzt Reflexionsmarker an den Streckenbegrenzungen, welche mit Infrarotlicht beleuchtet werden. Die Hindernisse sind mit ähnlichen Markierungen versehen.

- **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. Die Anordnung der Marker definiert die Maße und Ausrichtung der Rennstrecke und ermöglicht die Erfassung einer möglichen Verschiebung der Rennstrecke.
  Durch die Markierungen auf einem Hindernis entsteht ein weiteres Muster, wodurch die Größe festgestellt werden kann. Über das Muster auf den Hindernissen wird zudem die Pose und eine ungewollte Verschiebung erkannt.

Eingabeschnittstellen

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

Tabelle 9.33.: Eingabeschnittstellen des Subsystems Rennstrecke.

Ausgabeschnittstellen

Ausgabeschnittstellen - Subsystem Rennstrecke				
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Reflektiertes / Infrarotlicht	Hindernispose / -maße	—	—	—
	Rennstrecken-pose/-maße	—	—	—

Tabelle 9.34.: Ausgabeschnittstellen des Subsystems Rennstrecke.:

9.2.5. Subsystem 5: 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, Hindernisdaten, Fehlercodes, Kontrolldaten 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.

Benutzerinterface

Das Benutzerinterface der Systemsteuerungssoftware besitzt Buttons für das Wechseln der Systemzustände, sowie Textfelder für die Ausgabe von Systeminformationen und die Eingabe von Konfigurationsparametern. Zudem wurde von der Gruppe *RCCARSng* zusätzliche Textfelder zur Angabe der Mindestdurchschnittsgeschwindigkeit, der Fahrverhalten und der Fahrzeug ID hinzugefügt. Es werden Buttons für das Bestätigen der Inspektion, zum Starten des Rennbetriebs, zum Abschalten des Systems und für einen manuellen Not-Aus bereitgestellt. Zusätzlich gibt es einen Button zur Bestätigung der Fahrzeuganzahl.

Benutzerinterface - Systemsteuerungssoftware			
Beschreibung	**Datentyp**	**Wertebereich**	**Einheit**
Textfeld - Erwartete Fahrzeuganzahl	int	0 ... 16	—
Textfeld - Mindestdurchschnittsgeschwindigkeit	double	0.0 ... 4.0	—
Textfeld - Fahrverhalten	int	0 ... 3	—
Textfeld - Fahrzeug ID	int	0 ... 15	—
Textbereich - Systeminformationen	String	—	—
Textbereich - Fehlermeldungen	String	—	—
Button - Inspektion bestätigen	—	—	—
Button - Rennbetrieb starten	—	—	—
Button - Wartung	—	—	—
Button - System beenden	—	—	—
Button - Manueller Not-Aus	—	—	—
Button - Fahrzeuganzahl bestätigen	—	—	—

Tabelle 9.35.: Schnittstellen des Benutzerinterfaces der Systemsteuerungssoftware. Nicht aufgeführt sind die Textfelder sämtlicher Konfigurationsparameter.

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

Konfigurationsparameter

Vor der Initialisierung der Systemsteuerungssoftware müssen Konfigurationsparameter für den Betrieb festgelegt werden. Hierzu gehören insbesondere Parameter für die Festlegung von bestimmten Toleranzen und die Vorgabe der einzuhaltenden Realzeitanforderungen.

Schnittstellen

- **Fahrzeugdaten**
  Damit die Systemsteuerungssoftware die Positionsbestimmung überwachen kann und den aktuellen Standpunkt sämtlicher Fahrzeuge kennt, wird das Fahrzeugdatenpaket von der Systemsteuerung empfangen. Die Systemsteuerung erhält von der Positionsbestimmung die X- und Y-Koordinaten sämtlicher erkannter Fahrzeuge jeweils als Double-Wert (double) im Bereich 0.00 cm bis 236.0 cm. Zusätzlich werden die Ausrichtungswinkel sämtlicher erkannter Fahrzeuge als Double-Wert (double) im Bereich 0.00° bis 360.0°, die Verfügbarkeit der Fahrzeuge und ein Zeitstempel vom Datentypen timeval für die Feststellung des Alters der Fahrzeugposen mitgeteilt.

- **Zeitdaten**
  Um die Einhaltung der Realzeitanforderungen überprüfen zu können, wird der Zeitstempel, 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. Zur Identifizierung der CarControl und somit des Fahrzeugs wird die ID jeder Control-Unit im Zeitdatenpaket mitgesendet. Diese ist als Integer-Wert (int) angegeben.

- **Hindernisdaten**
  Das Hindernisdatenpaket umfasst die X- bzw. Y-Koordinaten der vier Eckpunkte aller erkannten Hindernisse. Diese werden als Double-Wert (double) im Bereich 0.00 cm bis 236.0 cm dargestellt. Die Anzahl der identifizierten Hindernisse wird an die Systemsteuerungssoftware übergeben. Die Anzahl der Hindernisse wird als Integer-Wert (int) im Wertebereich 0 bis 16 empfangen.

- **Kontrolldaten**
  Um der Systemsteuerungssoftware einen erfolgreichen Zustandswechsel der Control-Units und der Positionsbestimmung mitzuteilen, wird von der jeweiligen Komponente ein Kontrolldatenpaket versendet. Dieses enthält die ID als Integer-Wert (int) des Senders und den jeweiligen Systemzustand, in dem sich die Komponente befindet. Der Wertebereich 301 bis 316 entspricht aller möglichen Control-Units. Der Wert 200 als Sender ID ist für die Positionsbestimmung reserviert. Für das Wechseln der Systemzustände sendet die Systemsteuerungssoftware ein Kontrolldatenpaket. Mittels Kontrollbefehls als Integer-Wert (int) kann die Systemsteuerungssoftware das Wechseln des Systemzustandes aller Komponenten verlangen. Durch eine Empfänger ID wird die gewünschte Control-Unit oder die Positionsbestimmung angesprochen. Die unterschiedlichen Systemzustände sind in der Tabelle A.2 dargestellt.

- **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 Abschnitt C.1 aufgelistet ist.

- **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 und in Abschnitt C.1 aufgelistet sind. Der Fehlerspeicher muss vor dem Start des Systems leer sein und gegebenenfalls manuell vom Administrator im Vorfeld geleert werden.

- **Konfigurationsdaten**
  Von der Systemsteuerungssoftware werden verschiedene Parameter als Konfigurationsdaten gesendet, die die Regelung der Fahrzeuge beeinflussen.
  So wird jeder Control-Unit ein Fahrzeug zugewiesen. Hierzu wird ein Array aus Integern verschickt, wobei der Index des Arrays plus 301 für die ID der jeweiligen Control-Unit steht und der Wert an dem Index der ID eines Fahrzeugs entspricht. Der Wert „-1“ steht dabei für keine Zuweisung.
  Nach dem gleichen Prinzip werden die Fahrverhalten für die verschiedenen Control-Units angegeben. Hierbei steht der Wert des Arrays jeweils für ein Fahrverhalten.

Der Wert „0“ steht für kein Fahrverhalten, der Wert „1“ für „passives Fahren“, der Wert „2“ für „Überholen“ und der Wert „3“ für „Überholt werden“.
Auch die Mindestdurchschnittsgeschwindigkeiten werden als Array verschickt. Hierbei wird die Geschwindigkeit als Double-Wert (double) in m/s angegeben.

- **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.

- **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 zum Systemstart aktuellen Uhrzeit und dem Tagesdatum benannt. So wird sichergestellt, dass Protokolldateien nicht überschrieben werden.

- **GUI**
  Innerhalb der Schnittstelle zwischen Benutzerinterface und Systemsteuerungssoftware werden Daten zum Fahrverhalten, der Anzahl der Fahrzeuge und der Mindestdurchschnittsgeschwindigkeiten ausgetauscht.

Eingabeschnittstellen

Eingabeschnittstellen - Systemsteuerungssoftware				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Fahrzeugdaten	Verfügbare Fahrzeuge	int[16]	0 / 1	—
	X-Koordinate	double[16]	0.0 ... 236.0	cm
	Y-Koordinate	double[16]	0.0 ... 236.0	cm
	Ausrichtung	double[16]	0.0 ... 360.0	°
	Zeitstempel der Fahrzeugposen	timeval	—	—
Zeitdaten	Control-Unit ID	int	301 ... 316	—
	Zeitstempel der Fahrzeugpose	timeval	—	—
	Zeitstempel der gesendeten Regelungswerte	timeval	—	—
Hindernisdaten	Gefundene Hindernisse	int	0 ... 16	—
	X-Koordinate Punkt 1	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 1	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 2	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 2	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 3	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 3	double[16]	0.0 ... 236.0	cm
	X-Koordinate Punkt 4	double[16]	0.0 ... 236.0	cm
	Y-Koordinate Punkt 4	double[16]	0.0 ... 236.0	cm
Kontrolldaten	Sender ID	int	200, 301 ... 316	—
	Aktueller Zustand des Senders	int	> 0	—
Fehlercode	Fehlercode	int	> 0	—
Fehlerspeicher	Fehlerspeicher	—	—	—
GUI	Interaktion mit GUI	—	—	—

Tabelle 9.36.: Eingabeschnittstellen der Systemsteuerungssoftware.

Ausgabeschnittstellen

Ausgabeschnittstellen - Systemsteuerungssoftware				
Ausgabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Konfigurationsdaten	Fahrzeugzuweisung zu CU	int[16]	-1 ... 15	—
	Fahrverhalten der CU	int[16]	0 ... 3	—
	Mindestdurchschnittsgeschwindigkeit	double[16]	0.0 ... 4.0	m/s
Kontrolldaten	Empfänger ID	int	200, 301 ... 316	—
	Wechsel der Systemzustände	int	> 0	—
Not-Aus Befehl	Wechsel in den Fehlerzustand	int	1	—
Fehlerspeicher	Fehlerspeicher	—	—	—
Protokoll	Protokolldatei	—	—	—
GUI	Interaktion mit GUI	—	—	—

Tabelle 9.37.: Ausgabeschnittstellen der Systemsteuerungssoftware.

10. Fahrzeugevaluation

In diesem Kapitel werden die Evaluationsaufbauten und die durchgeführten Versuche beschrieben. Zudem wird die Bestimmung der Parameter beschrieben.

10.1. Dutycycle

Die Control-Unit berechnet Beschleunigungs- bzw. Bremssignale und Steuerwerte für ein Fahrzeug. Die Control-Unit sendet hierbei Throttle- und Steeringwerte an ein Fahrzeug in einem Bereich von 0 - 4095 für jeweils Beschleunigen und Steuern. Für das Beschleunigen entspricht ein Throttlewert von 0 einer maximalen Beschleunigung, der Wert von 4095 einer maximalen Bremsung, wenn das Fahrzeug fährt, und einer maximalen Rückwärtsbeschleunigung, wenn das Fahrzeug steht. Der Wert 2130 entspricht der Neutralstellung.
Bei der Lenkung entspricht ein Steeringwert von 0 einem maximalen Linkseinschlag, der Wert 2130 der Neutralstellung und der Wert 4095 dem maximalen Rechtseinschlag.
Die CarControl empfängt diese Werte und leitet sie an das Fahrzeug weiter. Auf dem Fahrzeug werden diese Werte in einen Dutycycle, der die prozentuale Auslastung des Motors angibt, umgerechnet. Hierbei entspricht ein Wert von 1 der vollen Auslastung, der Wert von 0 keiner Auslastung.
Damit das Fahrzeug sicher über die Rennstrecke gesteuert werden kann, musste evaluiert werden, wie die Throttlewerte mit dem Dutycycle auf dem Fahrzeug zusammenhängen. Hierzu wurden feste Throttlewerte an das Fahrzeug gesendet. Ein Oszilloskop wurde parallel an den Motor angeschlossen, um den resultierenden Dutycycle zu messen. Eine Abbildung des Versuchsaufbaus ist in Abbildung 10.1 zu sehen und ein Ausschnitt, der zeigt wie das Oszilloskop angeschlossen wurde in Abbildung 10.2.
Zur Ermittlung des Zusammenhangs zwischen Dutycycle und Throttlewert wurde der Throttle in fünfziger Schritten erhöht und der resultierende Dutycycle notiert. Die Ergebnisse dieser Messung sind in Abbildung 10.3 für die Vorwärtsfahrt und Abbildung 10.4 für die Rückfährsfahrt zu sehen. Mittels einer linearen Regression

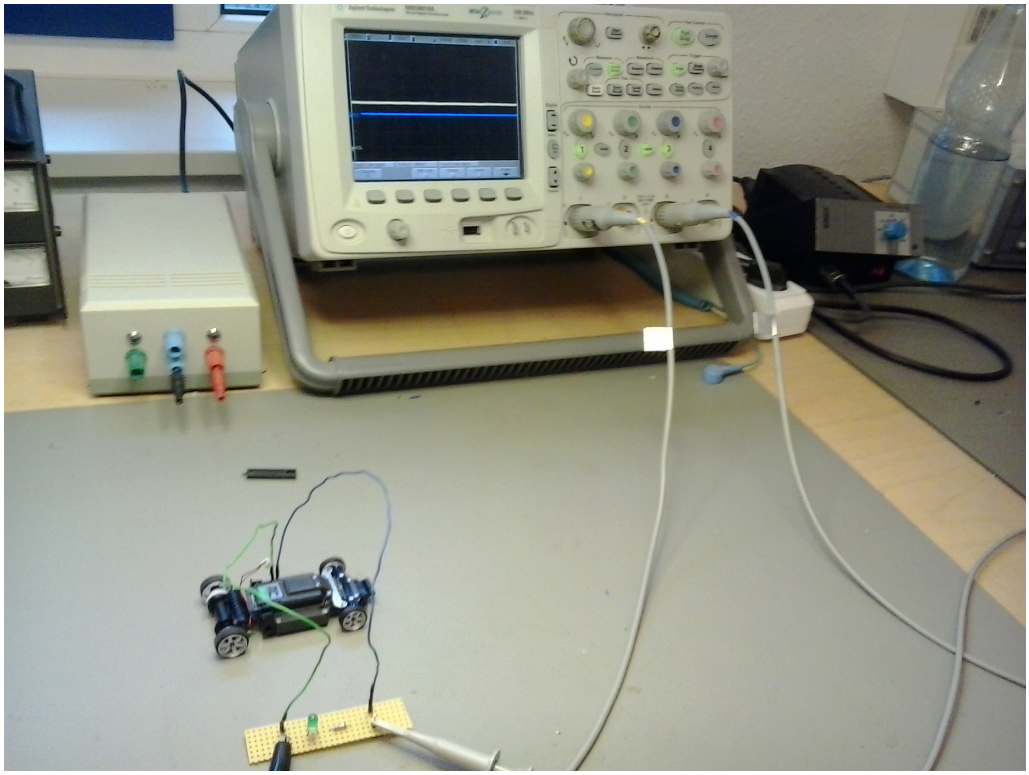


Abbildung 10.1.: Hier zu sehen ist das Fahrzeug, welches über eine Platine an das Oszilloskop angeschlossen ist.

wurde anschließend eine lineare Funktion ermittelt, die nun genutzt werden kann, um beliebige Throttlewerte in Dutycycle umzurechnen und anders herum. Die Funktion für die Vorwärtsfahrt sieht folgendermaßen aus:

$$Throttlewert = -427 * Dutycycle + 1988 \tag{10.1}$$

Für die Rückwärtsfahrt lautet die Funktion:

$$Throttlewert = 1018 * Dutycycle + 2259 \tag{10.2}$$

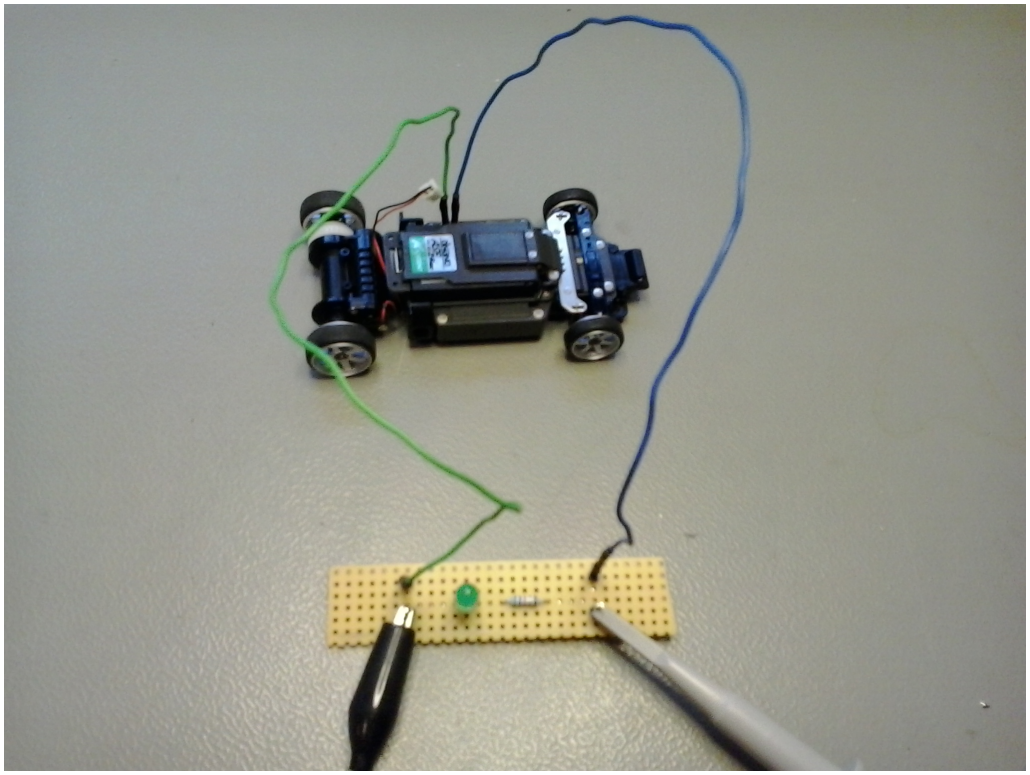


Abbildung 10.2.: Hier zu sehen ist die Platine, mit der der Fahrzeugmotor an das Oszilloskop geschlossen wurde.

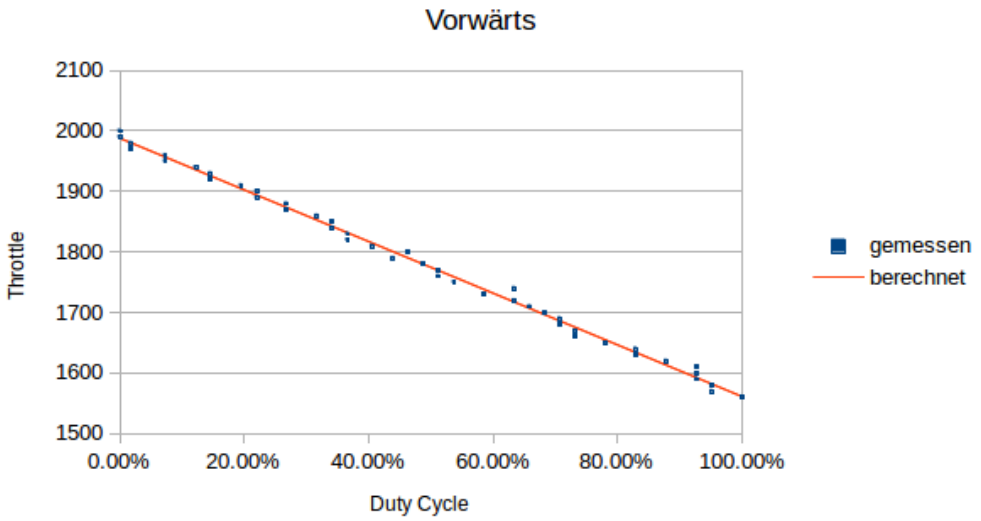


Abbildung 10.3.: Hier zu sehen ist der Zusammenhang zwischen Throttlewerten und Dutycycle bei der Vorwärtsfahrt. Die blauen Punkte stellen die gemessenen Werte dar, die rote Linie stellt eine lineare Funktion dar, die aus den Messwerten ermittelt wurde.

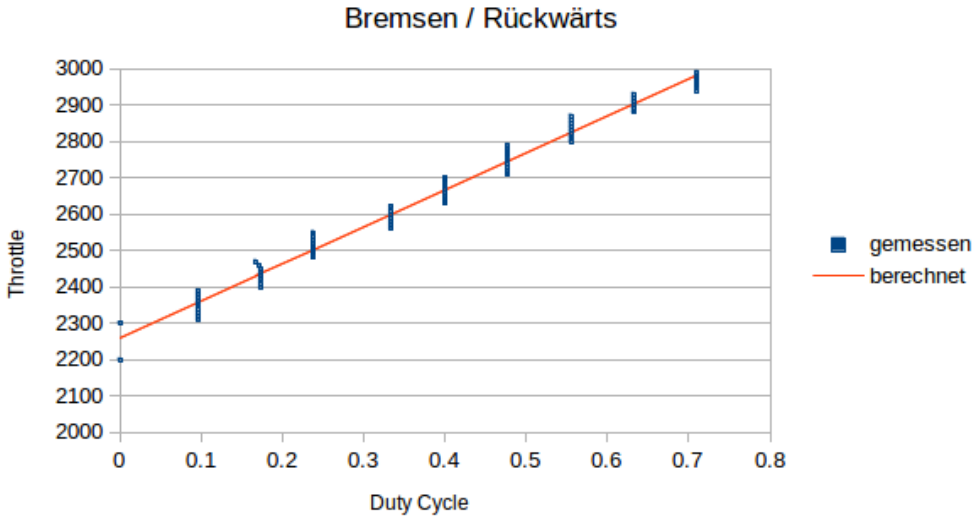


Abbildung 10.4.: Hier zu sehen ist der Zusammenhang zwischen Throttlewerten und Dutycycle bei der Rückwärtsfahrt. Die blauen Punkte stellen die gemessenen Werte dar, die rote Linie stellt eine lineare Funktion dar, die aus den Messwerten ermittelt wurde

10.2. Lenkwinkel

Um das Lenkverhalten des Fahrzeugs zu untersuchen, musste der Zusammenhang zwischen Steeringwert und Lenkwinkel evaluiert werden. Hierzu wurden feste Steeringwerte an das Fahrzeug gesendet und der resultierende Lenkwinkel gemessen. Der Versuchsaufbau ist in Abbildung 10.5 gezeigt.

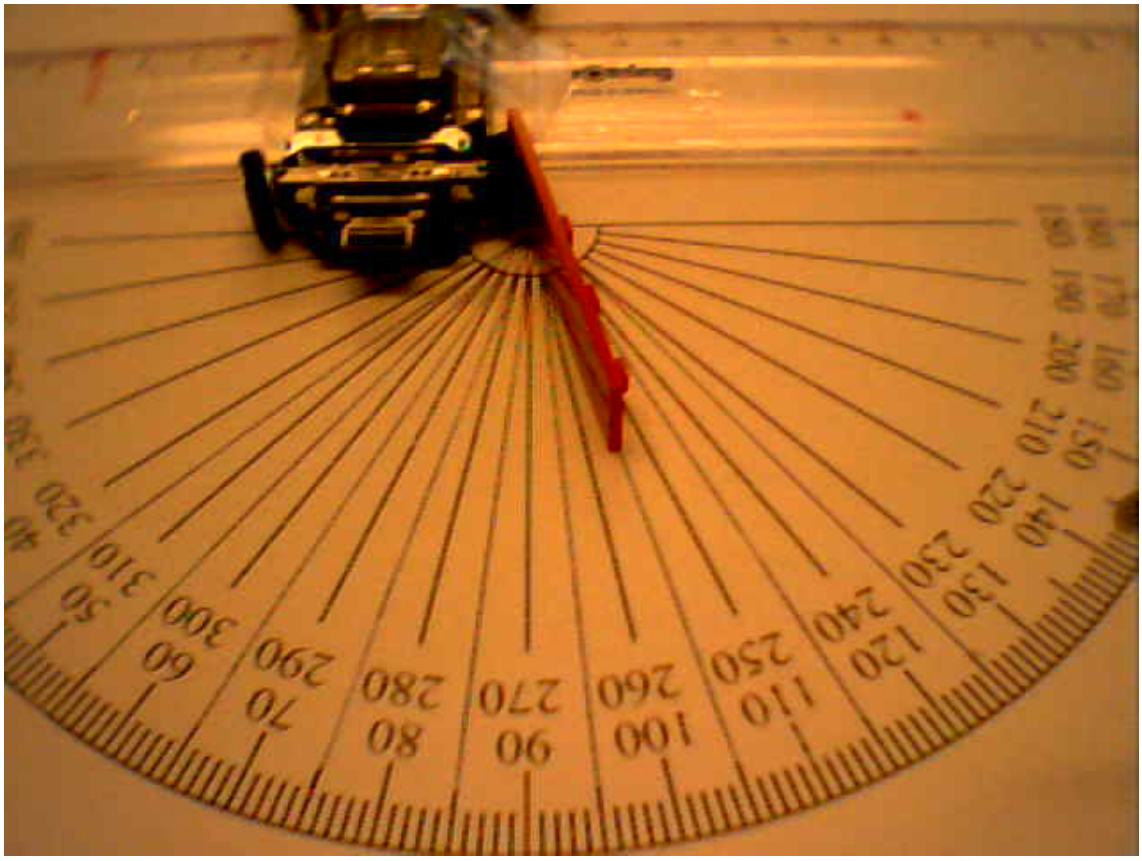


Abbildung 10.5.: In dieser Abbildung ist der Versuchsaufbau zur Lenkwinkelevaluation zu sehen. Das Fahrzeug wurde hierzu auf einem Kreisbogen fixiert, sodass die Lenkwinkel abgelesen werden konnten. Der rote Balken diente als Verlängerung der Reifen, sodass die Ausrichtung dieser zuverlässiger abgelesen werden konnte.

Hierbei wurde festgestellt, dass die Ausrichtung der Reifen ein gewisses Spiel besitzt. Dieses Spiel kann bis zu 5° betragen. Deswegen wurde für jede Messung der maximale Linkseinschlag sowie der maximale Rechtseinschlag gemessen. Diese beiden Lenkausschläge wurden gemittelt, und auf diesen Werten eine lineare Regression durchgeführt, um einen Zusammenhang zwischen Steeringwert und Lenkwinkel herzustellen. Die Ergebnisse dieser Messreihe sind in Abbildung 10.6 zu sehen.
Diese Messreihen wurde für zwei Fahrzeuge durchgeführt. Da beide Messreihen jedoch ähnliche Ergebnisse lieferten konnte zur Umrechnung von Steeringwerten zu

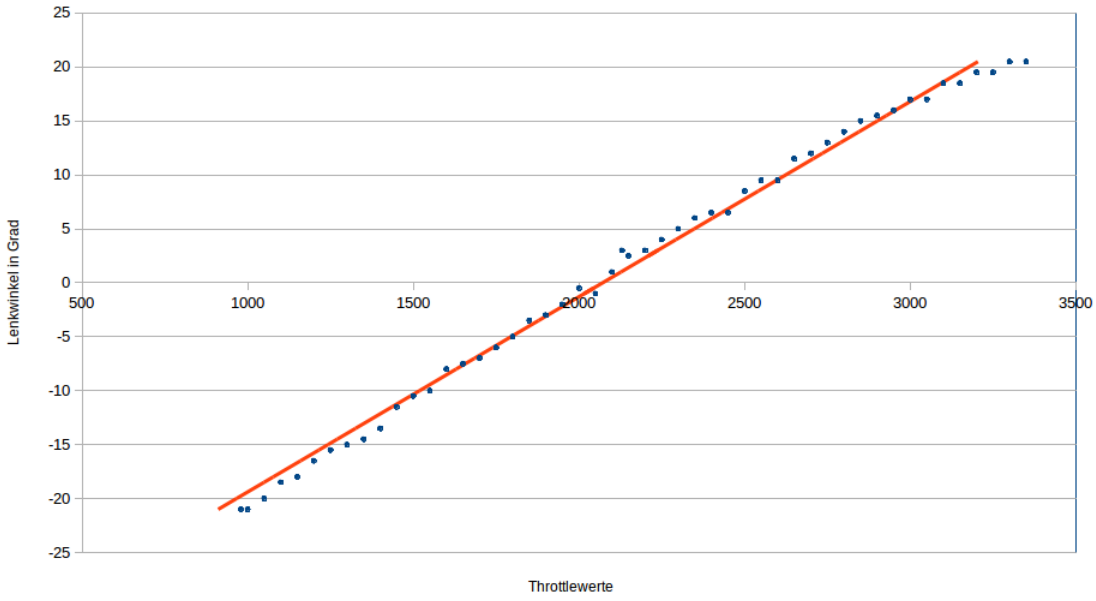


Abbildung 10.6.: Auf dieser Abbildung lassen sich die Messergebnisse der Lenkwinkelevaluation ablesen. Die blauen Punktes stellen die Messwerte dar, die rote Linie ist die Funktion, die sich aus der linearen Regression ergeben hat.

Lenkwinkeln die gleiche Formel verwendet werden. Die Formel zur Umrechnung lautet:

$$Steeringwert = 55 * Lenkwinkel + 2071 \tag{10.3}$$

10.3. Beschleunigung

Um die maximale Beschleunigung der Fahrzeuge zu messen, wurde ein Evaluationsstand entwickelt. Dieser bestand aus einem 8 m langen und 5.5 cm breiten Holzkanal in den, in regelmäßigen Abständen, Lichtschranken eingebaut wurden. Ein Abschnitt dieses Kanals ist in Abbildung 10.7 zu sehen. Die Lichtschranken bestanden aus je einem Fotowiderstand und einer Laserdiode. Um die Zuordnung und Ausrichtung von Laserdioden und Fotowiderständen zu vereinfachen, wurden passende Löcher gleichzeitig gebohrt. Die Fotowiderstände besitzen einen geringen Widerstand, wenn sie direkt von der Laserdiode angestrahlt werden und einen hohen, wenn sie nicht angestrahlt werden. Die an dem Widerstand abfallende Spannung wurde gemessen und an eine CarControl übertragen. Aus den Lichtschrankendaten konnte die genaue Position von Fahrzeugen in dem Kanal abgeleitet werden. Die Beschleunigungsmessung wurde mit der in Michael Bukowskis Masterarbeit (siehe Abschnitt 4.4.2) evaluierten Hardwareplatine, welche an den Fahrzeugen befestigt wurde, durchgeführt. Auf dieser Platine ist ein Bluetooth-Modul und ein Beschleunigungssensor vorhanden.

Des Weiteren wurde eine Fahne an dem Fahrzeug befestigt. Diese Modifikationen sind in Abbildung 10.8 zu sehen. Die Fahne hat das Auslesen der Fotowiderstände ermöglicht, da die Bohrlöcher mit den Laserdioden und Fotowiderständen sich über dem Fahrzeug befanden. Zudem wurden die evaluierten Fahrzeuge vor Durchführung der Messung gereinigt, und es wurde ein neuer Motor eingebaut.

Abbildung 10.7.: Teil des aufgebauten Evaluationskanals.

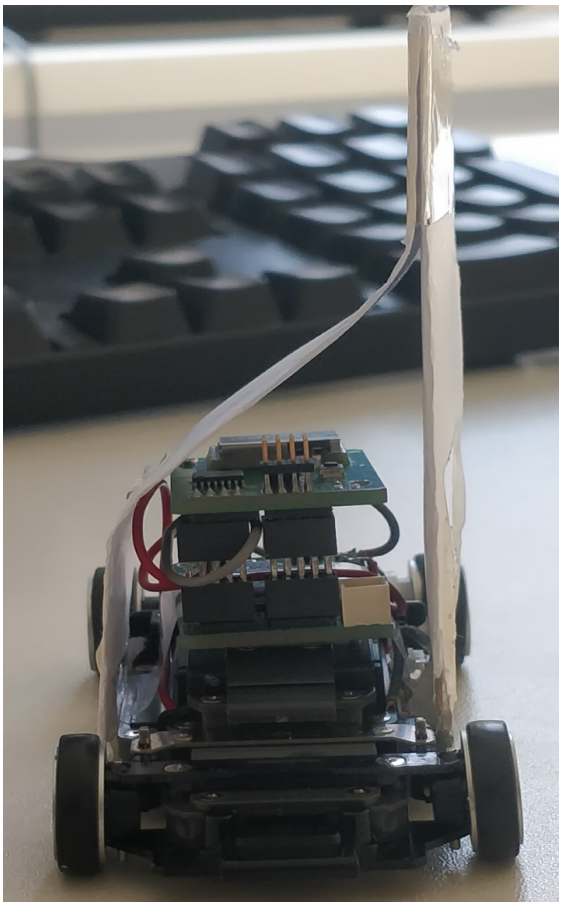


Abbildung 10.8.: Modifiziertes Fahrzeug mit Fahne und Platine.

Der Beschleunigungssensor wurde vor jeder Messung kalibriert. Die Messung des Beschleunigungssensors startete erst dann, wenn sich das zu evaluierende Fahrzeug bewegt und der Sensor Werte berechnet, die außerhalb des durch die Kalibrierung bestimmten Bereiches liegt.

Neben der Erweiterung der CarControl-Software, die das Auslesen der Lichtschranken ermöglichte, wurde der CarControl noch ein Evaluationsmodus hinzugefügt. In diesem Zustand konnte eine Befehlskette übertragen werden, welche erst bei einem weiteren Befehl ausgeführt wird. Die Befehlskette hatte den folgenden Aufbau:

$$ES\#\#B\#T\#\#\#\#ET\#\#$$

E Bezeichnet den Start des Evaluationsstrings

S## Beschreibt die Lichtschranke, bei der das Fahrzeug das Verhalten ändern soll

B# Beschreibt, ob das Fahrzeug bremsen soll oder nicht

T#### Beschreibt den Throttlewert, den das Fahrzeug halten soll

ET## Beschreibt die letzte durchfahrene Lichtschranke (Ende der Messung)

Hat ein Fahrzeug die letzte Lichtschranke nicht mehr durchfahren, wurde die Messung automatisch nach 10 s beendet. So wurden verschiedene Messreihen für jedes Fahrzeug durchgeführt. Es wurden, wie bei der Dutycycle Messung, die Throttlewerte in fünfziger Schritten erhöht. So wurde der Bereich der Throttlewerte von 1500 bis 1850 abgedeckt. Da die Fahrzeuge bei Werten > 1850 nicht mehr losgefahren wären, da die Kraft nicht mehr ausreichte, und Throttlewerte ≤ 1500 einem Dutycycle von 1 entsprechen und somit keine Veränderung hervorgerufen haben, sind in diesem Bereich alle Werte abgedeckt, die während des Betriebs erreicht werden können. Diese Messungen wurden für jeden Throttlewert mehrfach durchgeführt, sowohl mit als auch ohne Bremsen. Die vom Beschleunigungssensoren aufgenommenen Daten wurden über das Bluetooth-Modul an einen Rechner übertragen.

10.4. Parameterevaluation

Damit die Regelungssoftware des Systems ein Fahrzeug sicher über die Rennstrecke steuern kann, war es nötig ein Fahrzeugmodell zu nutzen, um somit Steuersignale ausgehend von diesem Modell zu berechnen. Eine solche modellprädikative Regelung benötigt Modellgleichungen mit verschiedensten Parametern, die das Verhalten des Fahrzeugs auf der Rennstrecke beschreiben.
Für das Projekt der Gruppe *RCCARSng* wurde das Slip-Free Modell, welches von Patrick Spengler und Christoph Gammeter in [SG10] beschrieben wurde, genutzt. Die Bestimmung des Modells erfolgte auf Basis der Daten der gleichen Fahrzeuge, die auch hier benutzt werden, weswegen sich dieses Modell gut für dieses Projekt anbietet. Die folgenden Formeln beschreiben das Fahrzeugmodell:

$$\begin{aligned}
\dot{X} &= v\cos(\Psi + C_1\delta) \\
\dot{Y} &= v\sin(\Psi + C_1\delta) \\
\dot{\Psi} &= v\delta C_2 \\
\dot{v} &= C_{m1}D - C_{m2}Dv - C_{r2}v^2 - C_{r0} - (v\delta)^2 C_2 C_1
\end{aligned} \tag{10.4}$$

Hierbei beschreibt X die X-Position des Fahrzeuges, Y die Y-Position des Fahrzeuges, Ψ die Ausrichtung und v die Geschwindigkeit. Das D gibt den Dutycycle an. Die Parameter C_{m1} und C_{m2} sind Motorparameter, C_{r2} und C_{r0} Reibungsparameter. Die Parameter C_1 und C_2 sind geometrische Parameter des Fahrzeugs und konnten ausgemessen werden.
Die übrigen Parameter wurden bestimmt, indem die in Abschnitt 10.3 beschriebenen Versuchsreihen ausgewertet wurden. Hierzu wurde im Wesentlichen die Gleichung, die die Veränderung der Geschwindigkeit beschreibt, also die Beschleunigung, genutzt. Da in den Versuchsreihen das Fahrzeug bedingt durch die Fahrbahnbegrenzungen eine gerade Strecke zurückgelegt hat, konnte angenommen werden, dass $\delta = 0$ gilt und sich die oben beschriebene Gleichung zu

$$\dot{v} = C_{m1}D - C_{m2}Dv - C_{r2}v^2 - C_{r0} \tag{10.5}$$

vereinfacht.
Nun war $\dot{v}$ durch die Messungen des Beschleunigungssensors bekannt, der Dutycycle D durch die in Abschnitt 10.1 beschriebene Evaluation. Die Geschwindigkeit v wurde berechnet, indem die gemessenen Beschleunigungswerte $\dot{v}$ integriert wurden.

Um die Parameter C_{m1}, C_{m2}, C_{r2} und C_{r0} zu bestimmen, wurde das Tool Matlab genutzt. Dazu wurden für jede Messreihe zeilenweise die bekannten Koeffizienten v und D in eine Matrix A geschrieben. Die Beschleunigungswerte $\dot{v}$ wurden in einen Vektor y geschrieben.
Nun konnte mit der Methode der kleinsten Quadrate (siehe [lsq]) der Parametervektor C mit folgender Gleichung bestimmt werden:

$$C = (A^T A)^{-1} A^T y \tag{10.6}$$

So konnte für jede Messreihe die Parameter C bestimmt werden. Die Ergebnisse dieser Evaluation sind in Tabelle 10.1 einzusehen. Zum Vergleich wurden dort auch die ermittelten Werte der ETH Zürich und der KU Leuven gelistet.
Die Parameterwerte liegen in ähnlichen Größenordnungen, wie die Werte für die Parameter, die von den anderen genannten Universitäten ermittelt wurden. Lediglich die Werte für C_{r2} von 0.006 und -0.003 weichen stark von den erwarteten Werten ab. Diese Abweichung konnte nicht endgültig geklärt werden. Tests haben jedoch gezeigt, dass die Fahrzeuge mit dem Wert, der von der ETH Zürich ermittelt wurde, besser fahren. Deswegen wurde dieser Wert für das System übernommen.

Weiterhin wurden die Messdaten des Beschleunigungssensors durch die Lichtschrankenmessungen validiert. Durch die Lichtschrankenmessungen war bekannt, zu wel-

Parameter	Fzg. 1 *RCCARSng*	Fzg. 2 *RCCARSng*	ETH Zürich	KU Leuven
C_1	0.5	0.5	0.5	0.5
C_2	16.13	16.13	16.13	17.06
C_{m1}	12.08	9.93	11.52	12.0
C_{m2}	2.19	2.08	2.74	2.17
C_{r2}	0.006	-0.003	0.05	0.1
C_{r0}	0.46	0.47	0.54	0.6

Tabelle 10.1.: Hier sind die Ergebnisse der Parameterevaluation der Projektgruppe *RCCARSng* für zwei Fahrzeuge, von Spengler und Gammeter von der ETH Zürich (Siehe [SG10]), und von Verschuren von der KU Leuven (siehe [Ver14])

chem Zeitpunkt das Fahrzeug eine bestimmte Position auf der Messstrecke erreichte. Um an dieser Stelle die Beschleunigungswerte zu nutzen, wurden diese doppelt mittels der Trapezregel integriert, um die Strecke zu erhalten. Wie dieses Matching aussieht, ist beispielhaft in Abbildung 10.9 zu sehen.
Hier fällt auf, dass die integrierte Strecke wesentlich größer ausfällt, als es die Werte der Lichtschranken vermuten ließen. Diese Tatsache lässt sich in allen Messreihen feststellen.

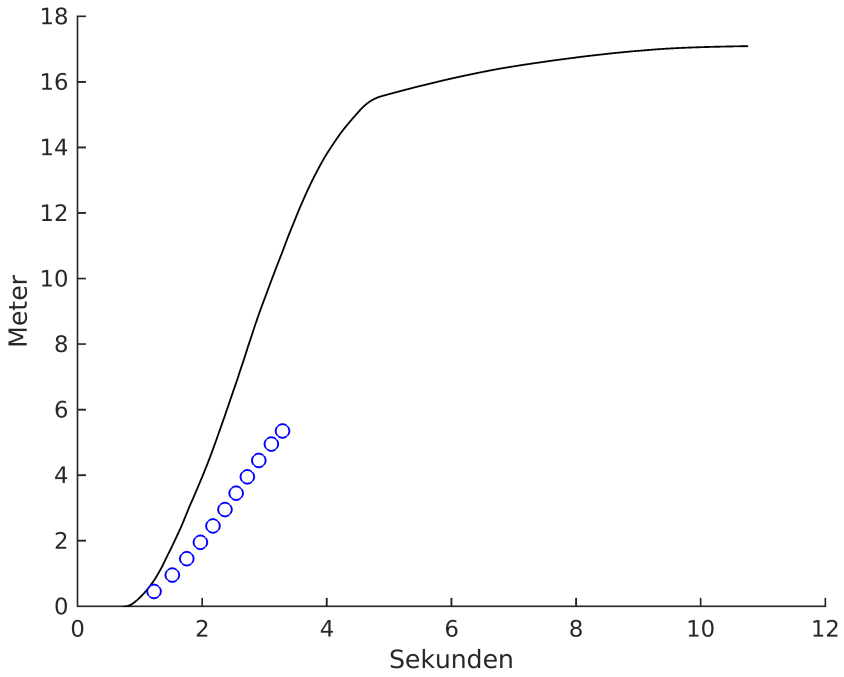


Abbildung 10.9.: In dieser Abbildung sind als blaue Punkte die Messpunkte der Lichtschranken einer Messreihe beispielhaft dargestellt. Die durchgezogene Linie stellt die integrierte Strecke aus den Beschleunigungsdaten dar.

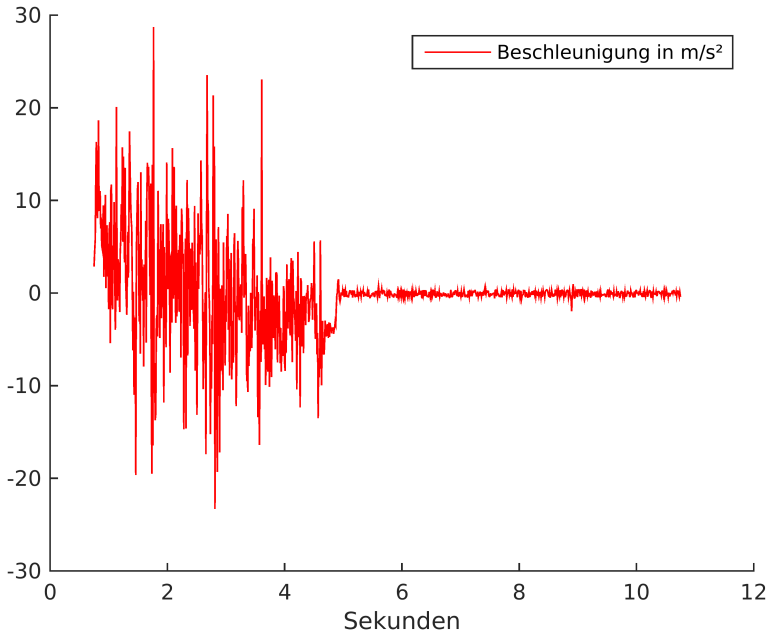


Abbildung 10.10.: In dieser Abbildung sind die Daten des Beschleunigungssensors zu sehen. Besonders in dem Zeitraum, in dem sich das Auto bewegt hat (<5 Sekunden), ist dieses Rauschen besonderes deutlich zu sehen.

Eine mögliche Erklärung für diese Abweichung wäre, dass die Beschleunigungsdaten zu stark verrauscht waren, als dass man auf diesen Berechnungen ausführen konnte, um brauchbare Ergebnisse zu erhalten. Diese Beschleunigungsdaten sind in Abbildung 10.10 zu sehen. Eine weitere Überlegung war, dass die Daten um einen konstanten Faktor verzerrt waren, und dass eine Multiplikation mit diesem Faktor zu besseren Ergebnissen führen könnte. Es konnte ein Stauchfaktor m berechnet werden, der für alle Messungen in der gleichen Größenordnung ($0.4 < m < 0.5$) blieb. Dieses m wurde bestimmt, indem mittels der Methode der kleinsten Quadrate der Faktor gesucht wurde, sodass die Abweichung zwischen den Lichtschrankenmessungen und der berechneten Strecke aus den neuen Beschleunigungsdaten minimal ist. Dieses m wurde auf 0.46 ermittelt. Mit diesem Stauchfaktor m konnte ein wesentlich besseres Matching erzielt werden (siehe Abbildung 10.11). Diese Stauchung hatte starken Einfluss auf die Berechnung der Motorparameter. Eine Gegenüberstellung der Parameter findet sich in Tabelle 10.2.
Es hat sich gezeigt, dass die Fahrzeuge sowohl mit den gestauchten als auch den nicht gestauchten Parameter über die Rennstrecke gesteuert werden können. An dieser Stelle wäre von Interesse, wie gut die Prognosen der modellprädikativen Regelung mit den aktuellen und zukünftigen Fahrzeugposen übereinstimmt, aus Zeitgründen konnte so eine Betrachtung jedoch leider nicht mehr durchgeführt werden.

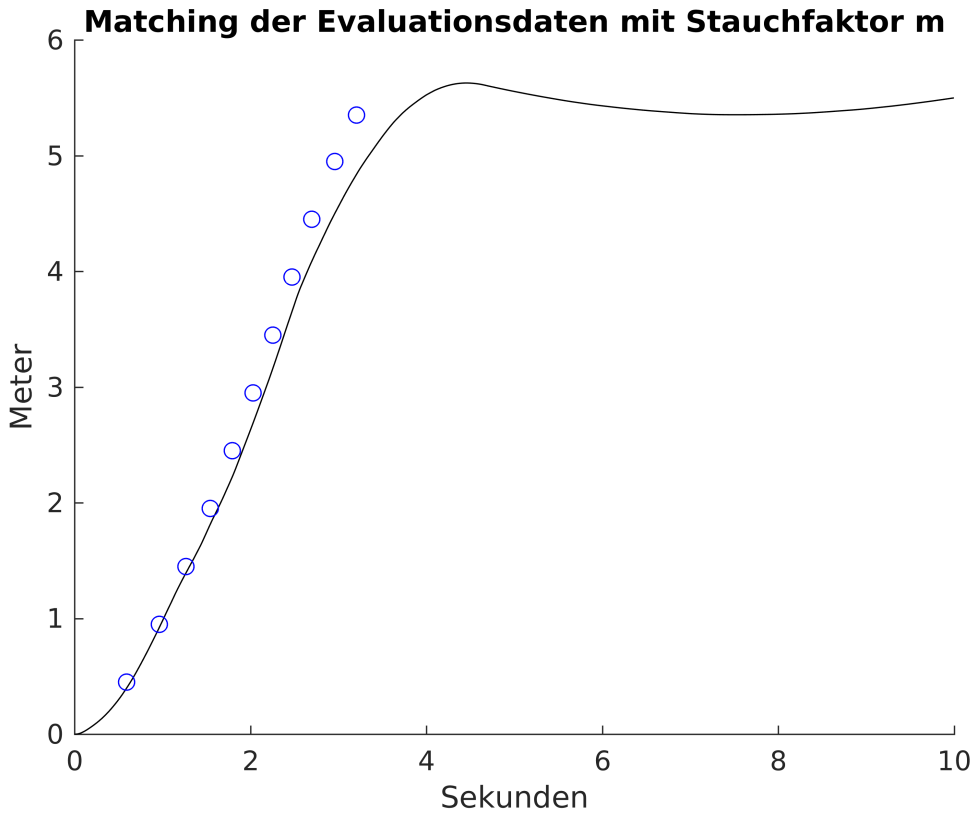


Abbildung 10.11.: In dieser Abbildung sind als blaue Punkte die Messpunkte der Lichtschranken einer Messreihe beispielhaft dargestellt. Die durchgezogene Linie stellt die integrierte Strecke aus den Beschleunigungsdaten, die mit dem Faktor m multipliziert wurden, dar.

Parameter	Fzg. 1 ohne m	Fzg. 2 ohne m	Fzg. 1 mit m	Fzg. 2 mit m
C_{m1}	12.08	9.93	4.53	3.52
C_{m2}	2.19	2.08	0.79	0.43
C_{r2}	0.006	-0.003	0.3	0.15
C_{r0}	0.46	0.47	0.15	0.2

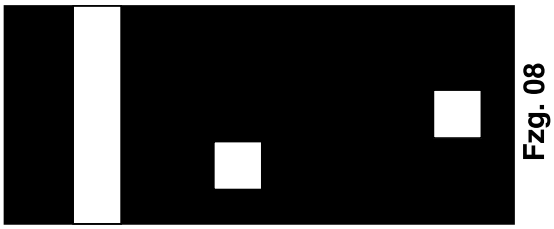
Tabelle 10.2.: In dieser Tabelle sind die Evaluationen der Parameter für die beiden Fahrzeuge ohne und mit dem Stauchfaktor m aufgeführt.

11. Entwicklungsstand bis zum 15.11.2017

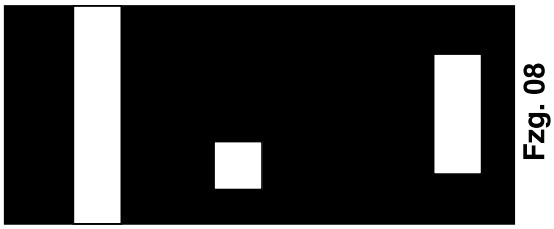
In diesem Kapitel werden alle Zwischenziele, die für das Erfüllen des in Abschnitt 2.3 beschriebenen Szenarios notwendig sind, erläutert. Des Weiteren wird der Entwicklungsstand der einzelnen Subsysteme aufgeführt. Somit werden alle Änderungen, die am Ausgangszustand (siehe Kapitel 5) vorgenommen wurden, erläutert.

11.1. Subsystem 1: Fahrzeug

Die Fahrzeuge wurden nur geringfügig geändert. Die einzige Änderung bezieht sich auf die Markierungen. Diese sehen nun wie in Abbildung 11.1(b) gezeigt aus. In Abbildung 11.1(a) ist zu sehen wie die Markierungen der Projektgruppe *RCCARS* aussahen. In dieser Gegenüberstellung ist zu sehen, dass der Marker, der das Ende eines Fahrzeuges anzeigt, von 1 cm × 1 cm zu 1 cm × 2.5 cm vergrößert wurde.



(a) Alte Fahrzeugmarkierungen



(b) Neue Fahrzeugmarkierungen

Abbildung 11.1.: Gegenüberstellung der veränderten Fahrzeugmarkierungen

Diese Vergrößerung erleichtert die Erkennung der Fahrzeuge durch die Positionsbestimmung. Hier wurden zu Testzwecken Dummyfahrzeuge erstellt. In Abbildung 11.2 ist ein Fahrzeug und ein solches Dummyfahrzeug zu sehen. Beide sind als Fahrzeug mit der ID 12 deklariert.

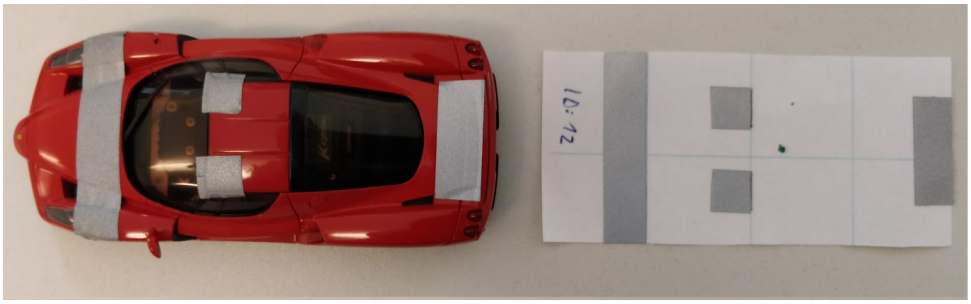


Abbildung 11.2.: Dummyfahrzeug und Originale zum Testen der Positionsbestimmung.

11.2. Subsystem 2: Positionsbestimmung

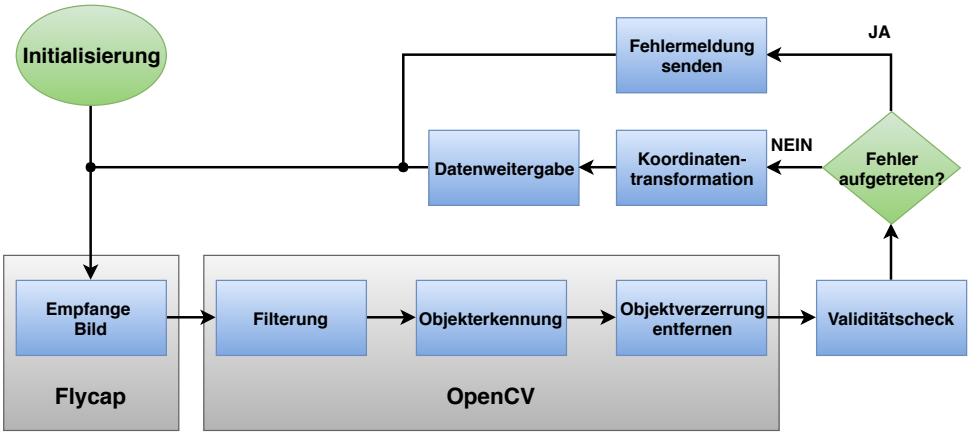


Abbildung 11.3.: Programmablauf der Positionsbestimmung.

Die Software der Positionsbestimmung wurde bezüglich der neuen Objekte, wie die Hindernisse, auf der Rennstrecke erweitert. Der Ablauf der Bildverarbeitung ist in Abbildung 11.3 zu sehen. Nach der Initialisierung werden die Bilder der Kamera eingelesen und verarbeitet. Aus diesen verarbeiteten Bildern werden die Objekte erkannt. Die Objekterkennung geschieht mit Hilfe von *OpenCV*. Diese Library stellt viele Bildbearbeitungsmethoden zur Verfügung. Alle Objekte werden über eindeutige Konturen gefunden und durch verschiedene Kriterien in unterschiedliche Kategorien eingeordnet. Die Positionsbestimmung kann die folgenden 5 Konturen auseinanderhalten:

Rennstreckenmarker sind 4.5 cm × 4.5 cm groß. Diese Marker beschreiben die Grenzen und Position der Rennstrecke.

ID-Marker sind 1 cm × 1 cm groß. Die Anordnung dieser Marker beschreibt die ID des Fahrzeuges.

Fahrzeugfrontmarker sind 4.5 cm × 1 cm groß und kennzeichnen die vordere Grenze des Fahrzeugs.

Fahrzeugheckmarker sind 2.5 cm × 1 cm groß und kennzeichnen die hintere Grenze des Fahrzeugs.

Hindernismarker sind zusammenhängende 1 cm breite Streifen aus Reflexionsfolie, die an den Kanten der Hindernisse (genauer beschrieben in Abschnitt 11.4) befestigt sind.

In Abbildung 11.4 ist zu sehen, wie die verschiedenen Marker zusammen arbeiten, damit einzelne Objekte erkannt werden können. Anhand dieser Marker ist die Positionsbestimmung nun in der Lage mehrere Fahrzeuge zu einem Zeitpunkt zu unterscheiden und ihnen eine eindeutige ID zuzuweisen. Zudem kann die Positionsbestimmung Hindernisse erkennen und weist auch diesen Hindernissen eine ID zu, die unter anderem das Debuggen erleichtert.

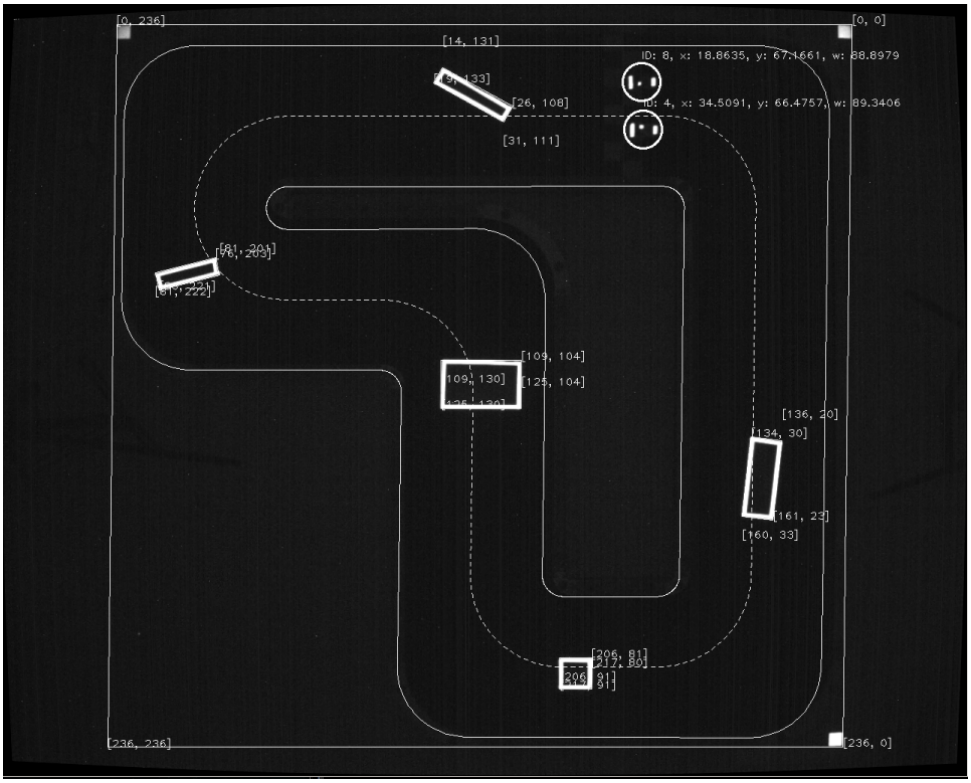


Abbildung 11.4.: Ausgabe der Positionsbestimmung mit fünf Hindernissen und zwei Fahrzeugen.

11.2.1. Fahrzeugerkennung

Damit die Positionsbestimmung mehrere Fahrzeuge gleichzeitig erkennen kann, mussten verschiedene Änderungen an der bestehenden Software vorgenommen werden. Dies wurde für das Erfüllen von Zwischenziel 1 benötigt. Ein großes Problem war das Wiederfinden der Fahrzeuge nachdem ein neues Bild aufgenommen wurde. Dieses Problem wurde gelöst, indem jeweils die Position des Fahrzeugfrontmarkers aus dem vorherigen Bild gespeichert wird, so dass es beim nächsten Bild als Referenz zur Verfügung steh. Diese wird nach jeder Aufnahme eines Bildes erneut gespeichert. Durch die Änderung der Fahrzeugheckmarker konnte die Berechnung der Ausrichtung deutlich vereinfacht werden. *OpenCV* stellt eine Methode zu Verfügung, mit der ein minimales Rechteck um eine Kontur gelegt werden kann. Dieses Rechteck liefert einen einen Ausrichtungswinkel, der zwischen 0 ° und 90 ° liegt. Diese Funktion wird auf die Kontur des Fahrzeugfrontmarkers angewendet. Zusammen mit dem bekannten Abstand zwischen den verschiedenen Markern können verschiedene Fahrzeuge, auch wenn sie nah beieinander liegen, auseinander gehalten werden.
Die Fahrzeugerkennung hat zur Zeit noch einige Probleme. Ein Problem ist das Trennen von Konturen. In Abbildung 11.5 ist zu sehen, dass die Fahrzeuge nicht erkannt werden. Dies liegt daran, dass die Software die Fahrzeugfrontmarker, wenn sie direkt nebeneinander liegen, als eine Kontur erfasst. Die Trennung dieser Kontur ist noch nicht fehlerfrei möglich.

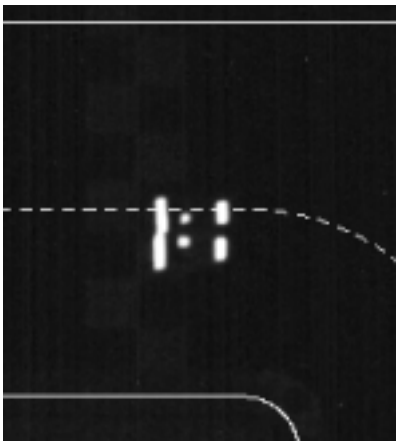


Abbildung 11.5.: Fahrzeuge, deren Konturen zu nah bei einander liegen.

Ein weiteres Problem tritt auf, wenn durch Ausleuchtungsprobleme die Konturen über mehrere Zyklen nicht korrekt erkannt werden. Hier kann es vorkommen, dass das System mehrere Fahrzeugposen an die gleiche Position legt.

11.2.2. Hinderniserkennung

Die Hinderniserkennung ist in den bisherigen Programmablauf eingebaut worden. Die Erkennung von Hindernissen wurde für das erste Zwischenziel benötigt. Die *OpenCV* Methode `findContours` liefert neben den Konturen selbst auch Informationen über die Anordnung von diesen. Ist eine Kontur komplett innerhalb einer anderen wird die Innere als Kind der Äußeren bezeichnet. Diese Hierarchie der Konturen ist in Abbildung 11.6 zusehen. Alle gefundenen Objekte werden als Kontur erkannt. In dieser Abbildung ist die gefundene Kontur 3a das Kind der Kontur 3. Hindernisse verfügen über die gleiche Anordnung von Markierungen und haben somit eine vergleichbare Hierarchieanordnung. Damit Hindernisse unterschieden werden können, wird ein minimales Rechteck um die innere Kontur gelegt und von diesem die Eckpunkte bestimmt. Die innere Kontur wurde gewählt, um direkt aneinander liegende Hindernisse voneinander zu trennen. Dieser Fall ist in Abbildung 11.7 zu sehen.

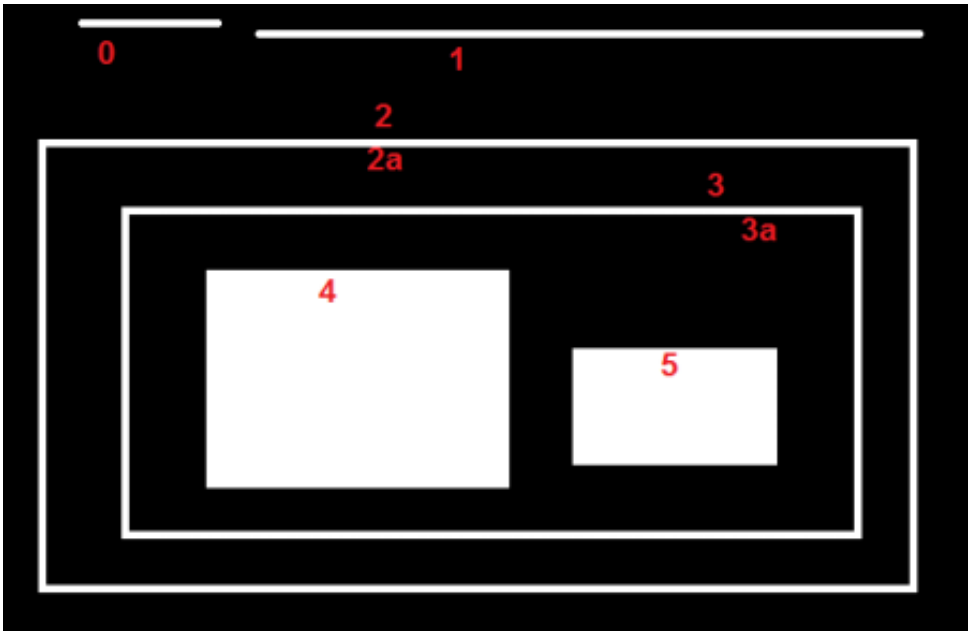


Abbildung 11.6.: Hierarchie mit verschiedene Konturen.

Zudem wird ein Fehler gesendet, wenn ein Hindernis bewegt wurde. Dies wird auch in der Bildausgabe angezeigt (siehe Abbildung 11.8).

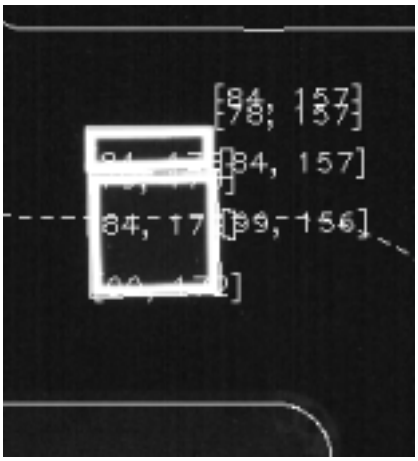


Abbildung 11.7.: Hindernisse, deren Konturen nah bei einander liegen.

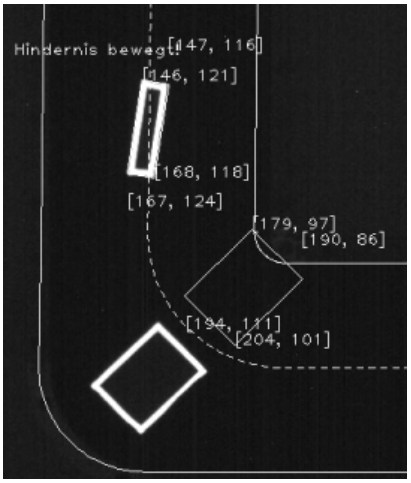


Abbildung 11.8.: Bewegtes Hindernis mit Fehlermeldung in der Ausgabe.

11.3. Subsystem 3: Control-Unit

Die in Kapitel 5.3.2 beschriebene Software der Control-Unit muss für eine Umsetzung der gesetzten Ziele angepasst werden. Um zunächst nur das Zwischenziel „Offline Trajektorienberechnung durch Simulation“ (siehe Abschnitt 3.3.2) zu realisieren, hat sich die Projektgruppe *RCCARSng*, ähnlich wie die Entwickler der ETH Zürich, dazu entschieden, in der Fahrzeugsteuerung zwischen Pfadplanung und Fahrzeugregelung zu unterscheiden (siehe Kapitel 4.1.1). Dies ist auch hinsichtlich einer modellprädikativen Regelung (MPC) von Vorteil, da diese vorherige Pfadplanungen voraussetzt. Für die Regelung des Fahrzeuges wird vorerst weiterhin der PID-Regler verwendet, der mit einer aus der Pfadplanung erstellten CSV-Datei arbeitet. Für den modellprädikativen Ansatz müssen noch weitere Voraussetzungen erfüllt werden. Die MPC braucht zum Einen eine Referenztrajektorie, zum Anderen konvexe Fahrbahnconstraints. Diese sind, insbesondere für vorhandene Hindernisse, nicht im Voraus gegeben.

11.3.1. Softwareentwicklung unter Matlab

Die Implementierung der Pfadplanung ist bislang in Matlab [Mat] erfolgt. Matlab bietet die Möglichkeit, mathematische Verfahren zur Pfadplanung zu implementieren und simulativ zu testen. Des Weiteren hat der Student Tom Reske seine Arbeiten an einer modellprädikativen Regelung hauptsächlich in Matlab implementiert, so dass die Weiterentwicklung der Software zunächst nur in Matlab erfolgte.

11.3.2. Repräsentation der Rennstrecke

Zunächst wurde eine Repräsentation der Rennstrecke entwickelt, die sowohl zur graphischen Darstellung als auch zu Berechnungen genutzt werden kann. Dazu wird die Mittellinie mit der Rennstrecke als Referenzlinie gewählt und deren Länge mit $L = 6.4274\,\mathrm{m}$ durch ausrechnen bestimmt. Anschließend wird sie mit dem Kurvenparameter $s \in [0, L]$ nach Bogenlänge parametrisiert. L wurde durch Ausmessen bestimmt. Der Wertebereich von s spannt sich demnach über die ganze Mittellinie. Jedes s repräsentiert einen Mittellinienpunkt P, für den die folgenden Größen ermittelt werden können:

x, y	Koordinaten von P
v_x, v_y	Einheitsvektor $\vec{v}$ von P entlang der Mittellinie
w_r, w_l	Abstand von P zum linken bzw. rechten Fahrbahnrand
κ	Krümmung in P

Der Vektor $\vec{v}$ kann um $90\,^\circ$ gedreht werden, welches den Vektor $\vec{n} = (-v_y, v_x)$ ergibt, der somit senkrecht zur Mittellinie steht. Mit dem Vektor $\vec{n}$ können die Punkte der Fahrbahnränder durch $P + w_r \cdot \vec{n}$ und $P - w_l \cdot \vec{n}$ berechnet werden. Ein Plot von 200 Punkten ist in Abbildung 11.9 zu sehen. Die Parameter w_r und w_l sind jeweils in den Kurvenobjekten `WidhtR` und `WidthL` als diskrete Lookup-Table hinterlegt. Die erste Spalte enthält den zugehörigen Parameterwert s, die zweite Spalte die Breite. Um auch von Parameterwerten, die nicht in der Tabelle gelistet sind, die Breite bestimmen zu können, werden die Breiten mit linearen Kurvensegmenten interpoliert. Die Kurvenelemente sind in `WidthR` und `WidthL` hinterlegt.

11.3.3. Hindernisse: Generierung oder Empfang aus dem Netzwerk

Um in Pfadplanungen Hindernisse mit berücksichtigen zu können, wird die Simulationsumgebung entsprechend erweitert. Zum Einen gibt es eine Funktion, mit der Eckpunkte von bis zu 16 Hindernissen in unterschiedlichen Positionen und Ausrichtungen generiert werden können. Auch ein optionaler Mindestabstand zwischen Hindernissen wird berücksichtigt. Zum Anderen besteht die Möglichkeit, den bereits vorhandenen Network-Receiver in Matlab einzubinden und per UDP Hindernisdaten direkt aus der Positionsbestimmung zu erhalten. Eine beispielhafte Platzierung der Hindernisse ist in Abbildung 11.10 zu sehen.

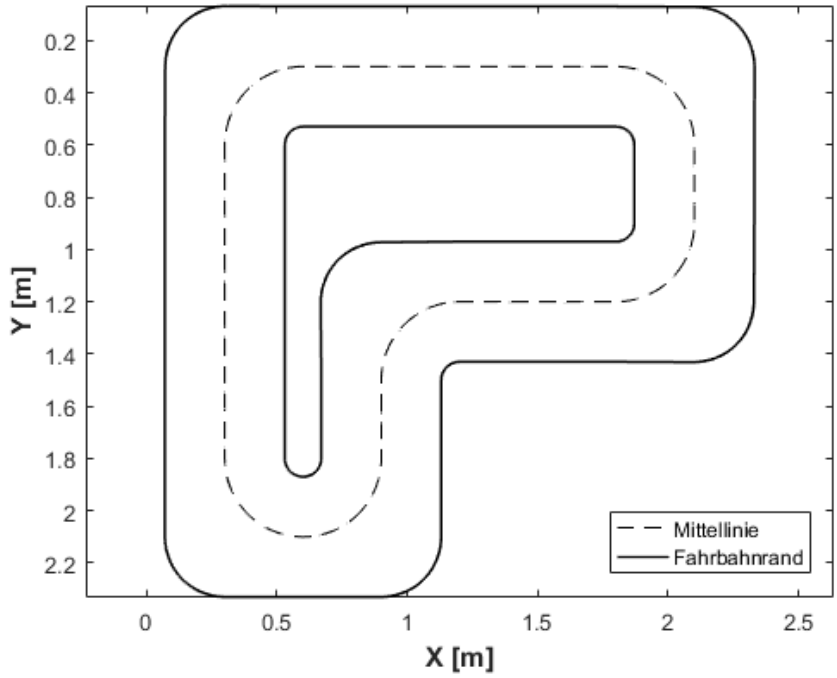


Abbildung 11.9.: Schematische Darstellung der Rennstrecke in der Matlab-Simulation.

11.3.4. Anpassung der Rennstrecke auf Hindernisse

Liegt ein Hindernis auf der Rennstrecke, so teilt es den Querschnitt der Fahrbahn in zwei zusammenhängende befahrbare Bereiche, die somit nicht konvex sind. Sowohl für eine modellprädikative Regelung, als auch für eine Umsetzung des Algorithmus zur krümmungsminimierten Trajektorie (siehe Abschnitt 4.1.2) sind konvexe Fahrbahnconstraints jedoch Voraussetzung. Daher wird der Fahrbahnrand für jedes Hindernis auf einer Seite so weit eingeschränkt, dass das Hindernis komplett abgedeckt ist und nicht mehr auf der Fahrbahn liegt. Somit kann jeder Fahrbahnquerschnitt als konvexer Constraint aufgefasst werden.
Zum Setzen der neuen Fahrbahnabstände gibt es die Methode $\texttt{set}(\texttt{s}_0, \texttt{s}_1, \texttt{w}_0, \texttt{w}_1)$. Diese setzt eine Fahrbahnbreite für den Kurvenparameter s_0 auf w_0 und für s_1 auf w_1. Zwischen s_0 und s_1 wird die Breite linear interpoliert. Um möglichst viel befahrbare Fläche zu erhalten, soll der Fahrbahnrand entlang der sichtbaren Hindernisseiten verlaufen. Dazu wird die Fahrbahnbreite jeweils zwischen zwei Hinderniseckpunkten gesetzt, indem die Mittellinienparameter s_0, s_1 und die Abstände w_0 und w_1 von Mittellinie zu den Eckpunkte bestimmt und der Methode `set` übergeben werden.
In Abbildung 11.11 ist dargestellt, wie der Fahrbahnrand eingegrenzt wurde. Des Weiteren ist zu sehen, dass zum derzeitigen Entwicklungsstand noch zwei Probleme offen sind, die im Folgenden beschrieben werden.

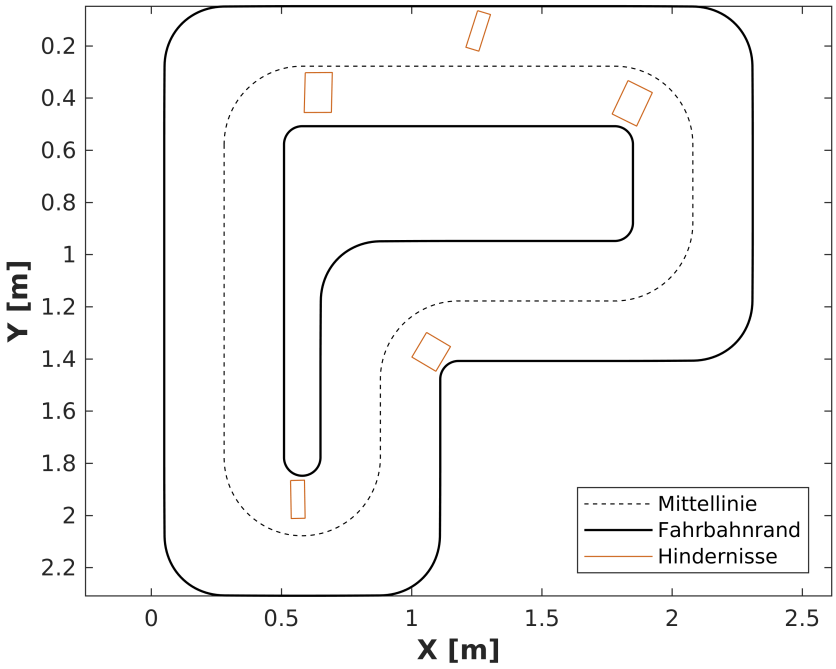


Abbildung 11.10.: Schematische Darstellung von generierten Hindernissen auf der Rennstrecke in der Matlab-Simulation.

1. Je nach Lage des Hindernis erfolgt der Abschluss mit dem Fahrbahnrand unterschiedlich. Die Unterscheidung dieser Fälle, die im folgenden kurz beschrieben werden, konnte bislang noch nicht implementiert werden.

   - Liegt das Hindernis parallel zu einem geraden Fahrbahnsegment, muss die Fahrbahnbreite nur entlang einer Hindernisseite abgeschlossen werden. Für die Anwendung von `set` werden die Eckpunkte einer parallelen Hindernisseite und deren Parameterwerte s_0 und s_1 benötigt. Zum Einstellen der Ränder wird `set` einmal aufgerufen. Dies entspricht Abbildung 11.12(a).

   - Falls das Hindernis nicht parallel zur geraden Fahrbahn liegt, so gibt es zwei Hindernisseiten, an denen die Fahrbahn eingeschränkt werden soll. Dann müssen drei zugehörigen Eckpunkte mit Parameterwerten identifiziert werden, und `set` wird zwei mal angewendet. Dies entspricht Abbildung 11.12(b).

   - Für den Fall, dass ein Hindernis im inneren Bereich einer Kurve liegt, kann es vorkommen, dass der Fahrbahnrand an drei Hindernisseiten entlang läuft. Dann muss `set` für jede der drei Seiten einmal ausgeführt

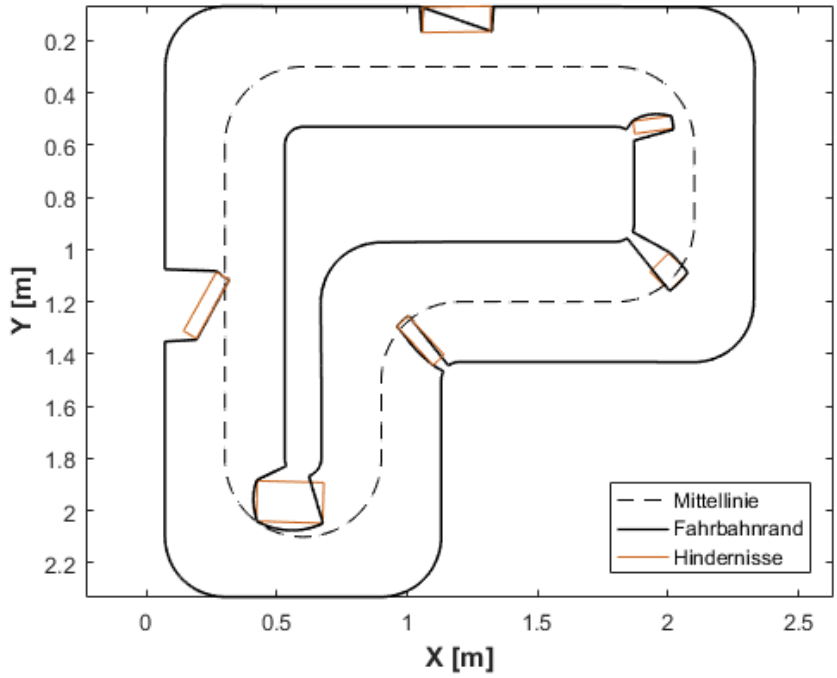


Abbildung 11.11.: Rennstrecke mit eingeschränktem Fahrbahnrand. Die Hindernisse wurden auf der nächstgelegenen Seite mit der Rennstrecke verbunden. Die Einschränkung der Fahrbahn erfolgte noch nicht fehlerfrei.

werden und 4 Eckpunkte samt Parameterwerten werden benötigt. Dies entspricht Abbildung 11.12(c).

- Für den Fall, dass ein Hindernis im äußeren Bereich einer Kurve liegt, kann der Fahrbahnrand an zwei Hinderniseckpunkten entlang laufen. Dann müssen 3 Eckpunkte samt Parameterwerten bekannt sein. Dies entspricht Abbildung 11.12(d).

2. Beim Aufruf von `set` werden Fahrbahnränder zwischen s_0 und s_1 linear interpoliert. Dieser lineare Verlauf ergibt in den Kurven allerdings einen kurvenförmigen Verlauf, da der Abstand senkrecht zum Kurvenverlauf aufaddiert wird. Falls sich ein Hindernis in einer Kurve befindet, ist dieser Effekt jedoch nicht erwünscht, da der Fahrbahnrand genau entlang von Hindernisseiten verlaufen soll, welche einen geraden Verlauf hat. Für eine mögliche Lösung könnten die Abstände nicht linear interpoliert, sondern zunächst in Punktkoordinaten umgewandelt werden. Diese Punkte werden dann durch eine Gerade g verbunden und für jedes $s \in [s0; s1]$ der Abstand zu g berechnet und als neue Breite gesetzt.

Um diese Probleme lösen zu können, wäre es möglich, die Fahrbahn nicht entlang der Hinderniseckpunkte, sondern entlang der nach s parametrisierten Mittellinie ein-

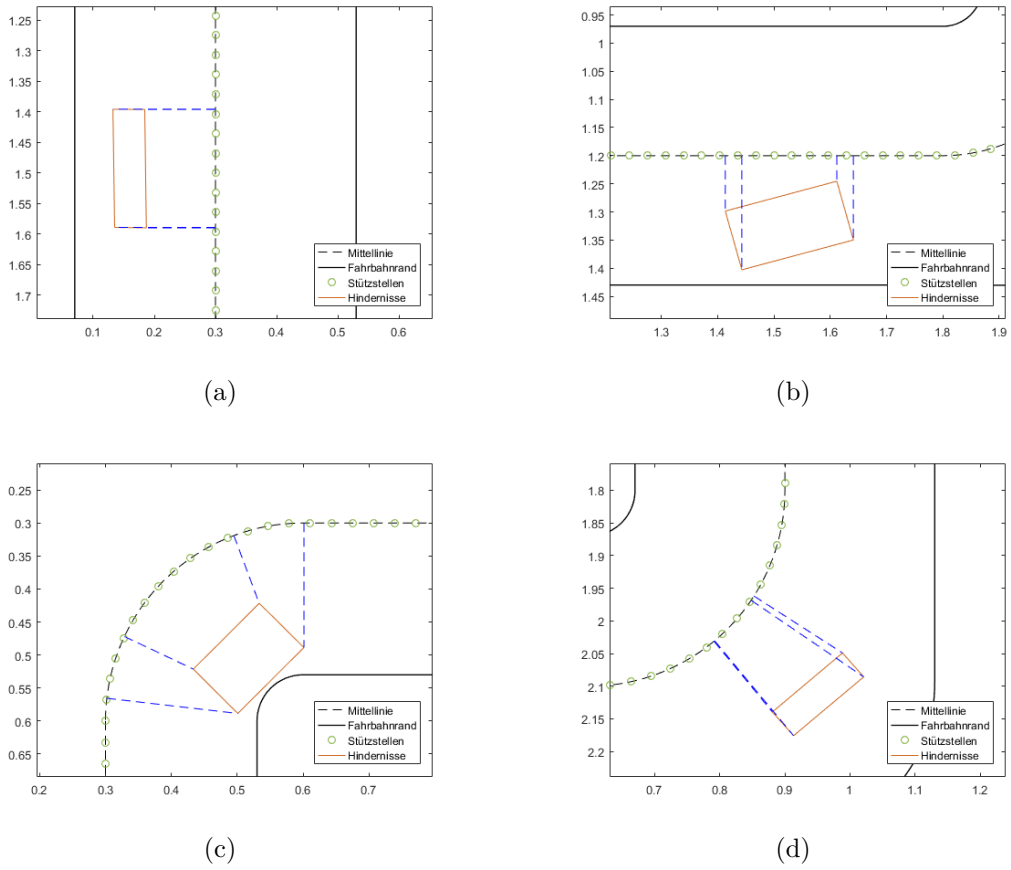


Abbildung 11.12.: Beschreibung vier möglicher Fälle beim Abschließen der Hindernisse mit der Wand.

zuschränken. Für eine diskrete Anzahl an Parameterwerten könnte die Rennstrecke durchlaufen und jeweils der nächstgelegene Punkt auf dem Hindernis gesucht werden. Diese Suche würde in Form eines linearen Programms erfolgen. Die Anzahl der Stützstellen müsste für die nötige Genauigkeit entsprechend erhöht werden.

Entscheidungshilfe durch A*

Weist ein Hindernis zu beiden Fahrbahnrändern ausreichend Freiraum um es zu umfahren, muss entschieden werden, auf welcher Seite das Fahrzeug fahren soll. Die Seite zu wählen, die am meisten Platz bietet, mag für einen Trajektorienverlauf nicht optimal sein, wenn stark gekrümmte Trajektorien zu fahren sind, da das Fahrzeug damit seine Geschwindigkeit verringern muss. Daher wird die Seitenauswahl einem Planungsalgorithmus überlassen. Die Projektgruppe hat sich für den in Kapitel 4.2 beschriebenen A*-Algorithmus entschieden. Er ist auf diskreten Graphenstrukturen anwendbar und kann Wege auf der Rennstrecke finden, so fern diese in einem Rastermaß (Grid) vorliegt. Da ein solches Grid in der Auflösung variabel ist, kann ein

Weg auch in einer groben Repräsentation der Rennstrecke gefunden werden, welche für eine Vorplanung von Trajektorien ausreichend ist. Weiterhin bietet A* durch seine Kostenfunktion die Möglichkeit, Kriterien für einen optimalen Pfad selbst zu setzten. Dies kann wahlweise der kürzeste Pfad sein, aber auch ein Pfad mit optimierter Krümmung wäre denkbar.
Die Projektgruppe *RCCARSng* nutzt für A* eine frei verfügbare Implementierung „A_Star_GUI“, die auch eine graphische Repräsentation von Gitternetz und Lösungspfad mitbringt. Zunächst wird mithilfe von A* ein kürzester Pfad bestimmt, falls dieser vorhanden ist. Hindernisse werden dabei auf der Seite des kürzesten Pfades umfahren. Für anders zu charakterisierende Pfade müsste die Kostenfunktion angepasst werden, was bisher noch nicht erfolgt ist. In einem ersten Schritt wird die Fahrbahn mit Hindernissen durch ein 94×94 großes Gitternetz diskretisiert. Da die Rennstrecke $2.37\,\mathrm{m} \times 2.37\,\mathrm{m}$ groß ist, ergibt sich ein Gitterabstand von 1.9 cm. Damit wird eine kleinstmögliche Hindernisseite von 5 cm durch mindestens 2 Punkte repräsentiert. Für jeden Gitterpunkt ist durch 0 bzw. 1 kodiert, ob er zum befahrbaren oder nicht befahrbaren Bereich (außerhalb vom Fahrbahnrand, innerhalb von Hindernissen) gehört ($0 \mathrel{\hat{=}}$ befahrbar, $1 \mathrel{\hat{=}}$ nicht befahrbar). Auf diesem Gitternetz kann anschließend A* angewendet werden, um einen Weg zu finden. Ein beispielhaftes Grid mit gefundenem Weg ist in Abb. 11.13 zu sehen.
Da A* auch überprüfen soll, ob es für die gegebene Hinderniskonstellation einen fahrbaren Weg gibt, ist es wichtig, dass in A* gefundene Wege nachgefahren werden können. Derzeit würde A* jedoch einen Weg finden, wenn Hindernisse dicht (geringer als die Fahrzeugbreite von 5 cm) beieinander oder am Fahrbahnrand stehen. Daher ist geplant die Hindernisse um 5 cm in jede Richtung zu vergrößern und den Fahrbahnrand um 5 cm in Richtung der Mittellinie einzuschränken. Aus dieser modifizierten Fahrbahn wird anschließend das Gitternetz erzeugt.
Der von A* gefundene Weg wird abschließend dahingehend untersucht, auf welcher Seite er an den Hindernissen vorbei läuft. Anschließend ist vorgesehen, den Fahrbahnrand für jedes Hindernis auf der entsprechenden Seite mithilfe der `set`-Methode einzuschränken, sodass konvexe Fahrbahnconstraints entstehen. Die Verknüpfung des A*-Ergebnisses und der `set`-Methode ist bislang jedoch noch nicht umgesetzt.

11.3.5. Pfadplanung durch iterative Krümmungsminimierung

In diesem Abschnitt ist die Implementierung der offline Trajektorienberechnung beschrieben. Die Grundlage bietet dabei der in Kapitel 4.1.2 beschriebene Algorithmus. Die Berechnung erfolgt durch zwei Funktionen. Die erste Funktion `KruemmungsOptSplines` berechnet für äquidistante Stützstellen eine optimierte Trajektorie. Die

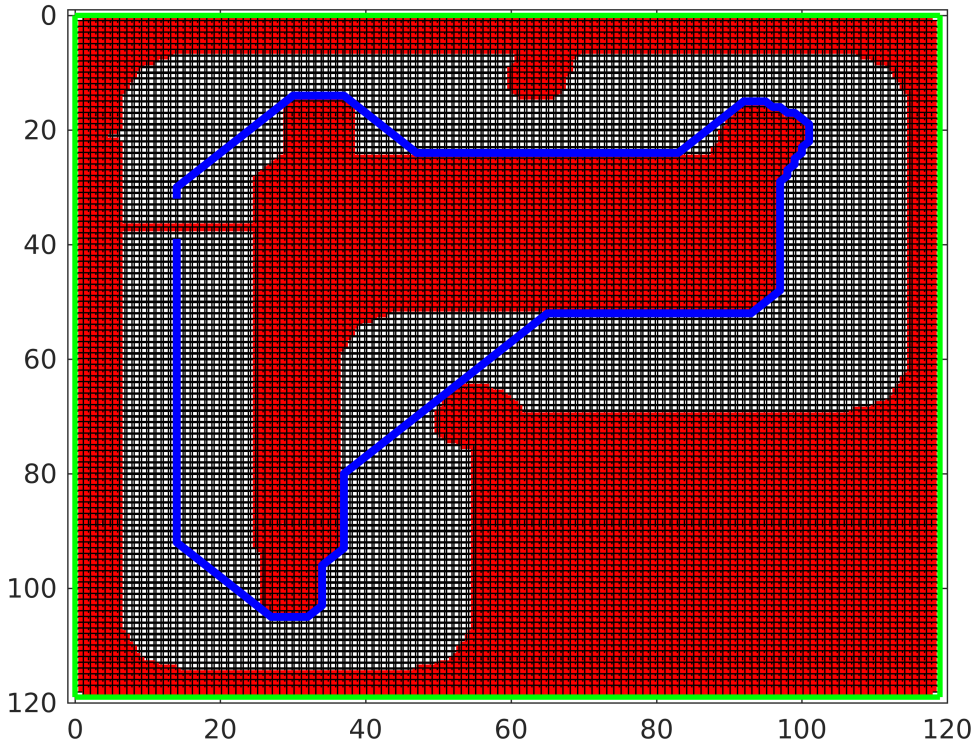


Abbildung 11.13.: Diskrete Rennstrecke in Form eines Gitternetzes mit durch A* gefundenem Weg. Rote Punkte kennzeichnen nicht befahrbaren Bereich.

zweite Funktion `PathplanningIterativSplines` ruft die erste in mehreren Iterationen aufgerufen, wobei zwischen den Iterationen Vorbereitungen, wie eine äquidistante Einteilung der Stützstellen, nötig sind. Beide Funktionen werden nun näher erläutert. Die Einbettung in den vollständigen Pfadplanungsalgorithmus ist in Abbildung 11.17 zu sehen.

`KruemmungsOptSplines` In der Funktion `KruemmungsOptSplines` werden übergebene Stützstellen auf der Rennstrecke verschoben und durch Splines verbunden, sodass eine optimierte Trajektorie entsteht. Diese sind mit $P_i = (p_{x_i}, p_{y_i}), i \in [1, n]$ benannt, wobei n die Anzahl der Stützstellen ist. Jedes P_i soll senkrecht zur Mittellinie verschoben werden können. Sei $L = (l_{x_i}, l_{y_i})$ der linke Begrenzungspunkt für P_i und d_i der Abstand von P zu L, dann kann mit des Normalenvektors $\vec{v}$ (sie-

he Abschnitt 11.3.2), der senkrecht zur Mittellinie steht, P_i wie folgt ausgedrückt werden:

$$\begin{aligned} P_i &= L + d_i \cdot \vec{n_i} \\ \Leftrightarrow \begin{pmatrix} p_{x_i} \\ p_{y_i} \end{pmatrix} &= \begin{pmatrix} l_{x_i} + d_i \cdot n_{x_i} \\ l_{y_i} + d_i \cdot ny_i \end{pmatrix} \end{aligned} \tag{11.1}$$

Die Splines sind durch Gleichung 4.1 beschrieben. Zusammen mit den Ableitungen der errechneten Splines ergibt sich

$$\begin{aligned} x_i(t) &= a_{i,x} + b_{i,x} \cdot t + c_{i,x} \cdot t^2 + d_{i,x} \cdot t^3 \\ y_i(t) &= a_{i,y} + b_{i,y} \cdot t + c_{i,y} \cdot t^2 + d_{i,y} \cdot t^3 \\ \dot{x}_i(t) &= b_{i,x} + 2c_{i,x} \cdot t + 3d_{i,x} \cdot t^2 \\ \dot{y}_i(t) &= b_{i,y} + 2c_{i,y} \cdot t + 3d_{i,y} \cdot t^2 \\ \ddot{x}_i(t) &= 2c_{i,x} + 6d_{i,x} \cdot t \\ \ddot{y}_i(t) &= 2c_{i,y} + 6d_{i,y} \cdot t \end{aligned} \tag{11.2}$$

Der Wert 0 für den Parameter t entspricht dem Punkt P_i, $t = 1$ entspricht dem nächsten Punkt P_{i+1}. Die Splines sollen eine geschlossene Trajektorie ergeben, daher ist für $i = n$ der folgende Index als 1 definiert. Es ergibt sich:

$$\begin{aligned} x_i(1) &= x_{i+1}(0) \\ y_i(1) &= y_{i+1}(0) \end{aligned} \tag{11.3}$$

Die Splines sollen krümmungsruckfrei verbunden werden, daher müssen die 1. und 2. Ableitungen an den Stützpunkten gleich sein:

$$x_i'(1) = x_{i+1}'(0) \quad \Leftrightarrow \quad b_{i,x} + 2c_{i,x} + 3d_{i,x} = b_{i+1,x} \tag{11.4}$$

$$y_i'(1) = y_{i+1}'(0) \quad \Leftrightarrow \quad b_{i,y} + 2c_{i,y} + 3d_{i,y} = b_{i+1,y} \tag{11.5}$$

$$\ddot{x}_i(1) = \ddot{x}_{i+1}(0) \quad \Leftrightarrow \quad 2c_{i,x} + 6d_{i,x} = 2c_{i+1,x} \tag{11.6}$$

$$\ddot{y}_i(1) = \ddot{y}_{i+1}(0) \quad \Leftrightarrow \quad 2c_{i,y} + 6d_{i,y} = 2c_{i+1,y} \tag{11.7}$$

Für jeden Spline $(x_i(t), y_i(t))$ mit $t \in [0, 1]$ soll gelten $(x_i(0)y_i(0)) = (p_{x_i}, p_{y_i})$, woraus folgt:

$$a_{i,x} = l_{x_i} + d_i \cdot n_{x_i} \tag{11.8}$$

$$a_{i,y} = l_{y_i} - d_o \cdot n_{y_i} \tag{11.9}$$

Zusammen mit Gleichung 11.3 ergibt sich:

$$\begin{aligned} l_{x_i} + d_i \cdot n_{x_i} + b_{i,x} + c_{i,x} + d_{i,x} &= l_{x_{i+1}} + d_{i+1} \cdot n_{x_{i+1}} \\ l_{y_i} + d_i \cdot n_{y_i} + b_{i,y} + c_{i,y} + d_{i,y} &= l_{y_{i+1}} + d_{i+1} \cdot n_{y_{i+1}} \end{aligned} \tag{11.10}$$

Die Gleichungen 11.10 und 11.3.5 beschreiben mit den freien Parametern d_i, $b_{i,x}$, $b_{i,y}$, $c_{i,x}$, $c_{i,y}$, $d_{i,x}$, $d_{i,y}$ den Lösungsraum für die Trajektorie. d_i ist dabei durch die Fahrbahnbreite eingeschränkt. Für die restlichen Parameter gibt es derzeit noch keine Einschränkung. Es ist aber denkbar, auch diese durch Fahrzeugparameter, wie fahrbare Kurvenradien, einzuschränken.
Innerhalb dieses Lösungsraumes soll gemäß Abschnitt 4.1.2 und Gleichung 4.5 nach der Summe der 2. Ableitung minimiert werden, um die Krümmung der Trajektorie klein zu halten. Diese kann durch Einsetzen der Ableitungen aus Gleichung 11.2 folgendermaßen ausgedrückt werden:

$$\sum_{i=1}^{n} \left[\left(\ddot{x}_i(0)\right)^2 + \left(\ddot{y}_i(0)\right)^2 \right] = \sum_{i=1}^{n} \left[\left(2c_{i,x}\right)^2 + \left(2c_{i,y}\right)^2 \right].$$

Zusammengefasst ergibt sich das folgende Minimierungsproblem:

$$\begin{aligned} &\underset{d_i, b_{i,x}, b_{i,y}, c_{i,x}, c_{i,y}, d_{i,x}, d_{i,y}}{\arg\min} \quad \sum_{i=1}^{n} \left[\left(2c_{i,x}\right)^2 + \left(2c_{i,y}\right)^2 \right] \\ &\text{s.t.} \\ d_i \cdot n_{x_i} + b_{i,x} + c_{i,x} + d_{i,x} - d_{i+1} \cdot n_{x_{i+1}} &= l_{x_{i+1}} - l_{x_i} \\ d_i \cdot n_{y_i} + b_{i,y} + c_{i,y} + d_{i,y} - d_{i+1} \cdot n_{y_{i+1}} &= l_{y_{i+1}} - l_{y_i} \\ b_{i,x} + 2c_{i,x} + 3d_{i,x} - b_{i+1,x} &= 0 \\ b_{i,y} + 2c_{i,y} + 3d_{i,y} - b_{i+1,y} &= 0 \\ 2c_{i,x} + 6d_{i,x} - 2c_{i+1,x} &= 0 \\ 2c_{i,y} + 6d_{i,y} - 2c_{i+1,y} &= 0 \end{aligned} \tag{11.11}$$

Das oben beschriebene Minimierungsproblem wird in Matlab mittels des Gurobi-Solvers [Gur] gelöst. In Abbildung 11.14 ist die Ausgabe dieses Gurobi-Solvers abgebildet.

`PathplanningIterativSplines` In Abbildung 11.14 ist anhand der Radien von Trajektorie und Mittellinie zu sehen, dass die Krümmung der berechneten Trajektorie nicht in allen Kurven geringer als die der Mittellinie ist (siehe besonders in der unteren Halbkurve der Abbildung). Dieser Effekt folgt aus der unzutreffen-

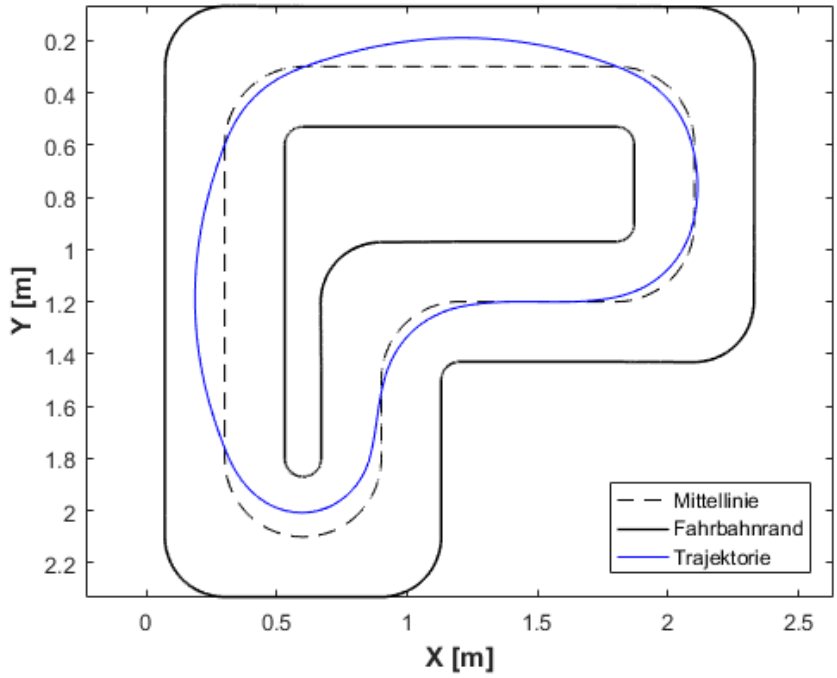


Abbildung 11.14.: Berechneter Pfad durch `KrümmungsOptSplines` nach einem Durchlauf des Solvers.

den Annahme, dass die Stützstellen der Splines äquidistant sind. Zwar sind die Stützstellen entlang der Mittellinie äquidistant verteilt, werden die Punkte auf den Stützstellen jedoch senkrecht zur Mittellinie verschoben, ist die Äquidistanz zwischen diesen Stützstellen nicht mehr gewährleistet. Um dieses Problem zu lösen, wird `KrümmungsOptSplines` in mehreren Iterationen ausgeführt, wobei nach jeder Iteration die Stützstellen so neu verteilt werden, dass die Trajektorie in äquidistante Abschnitte unterteilt ist. Diese Unterteilung erfolgt mit der frei verfügbaren Funktion `interparc` [D'E]. Die Durchführung der Iterationen ist in der Funktion `PathplanningIterativSplines` implementiert. Eine Trajektorie, die durch 10 Iterationen berechnet wurde ist in Abbildung 11.15 zu sehen.

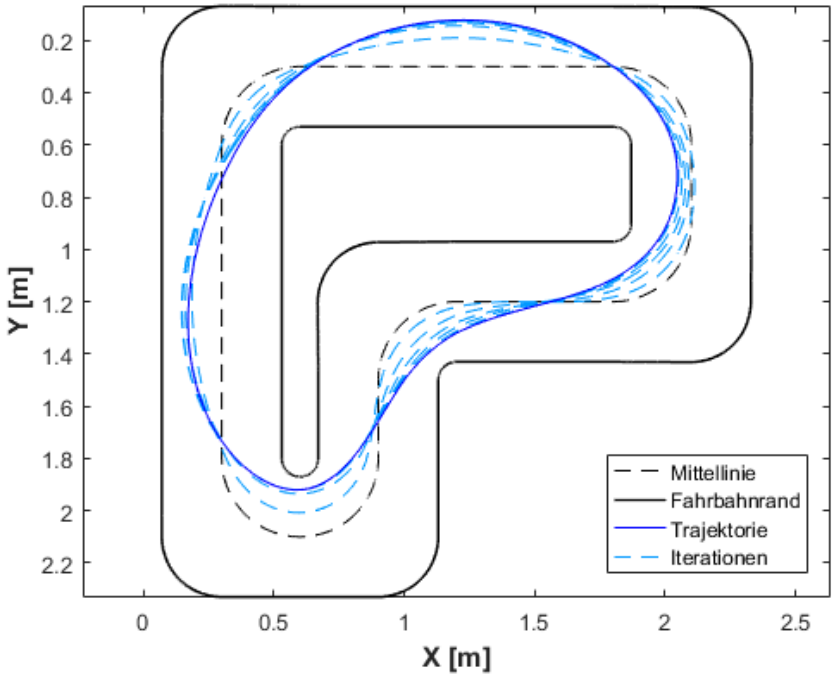


Abbildung 11.15.: Berechnete Trajektorie nach 10 Iterationen. Der Pfad nach jeder zweiten Iteration ist ebenfalls dargestellt.

Geschwindigkeitsprofil Um abschätzen zu können, mit welcher Geschwindigkeit ein Fahrzeug die Trajektorie maximal befahren kann, wurde zusätzlich ein Geschwindigkeitsprofil entwickelt. Dazu wird die Trajektorie, nach deren Bogenlänge durch s parametrisiert. Anschließend wird diese durch die Anfangspunkte der Splines in äquidistante Stützstellen s_i eingeteilt ($i \in [1, N]$, N ist die Anzahl der Stützstellen), zu denen mithilfe der Splines ebenfalls die Krümmung (κ_i) berechnet werden kann. Ziel ist ein optimales Geschwindigkeitsprofil (v_i, a_i), bestehend aus der Geschwindigkeit v_i und der Beschleunigung a_i. Dabei darf die Beschleunigung a_i nicht außerhalb der Grenzen $a_{\min}$ und $a_{\max}$ liegen und eine maximale Querbeschleunigung $q_{\max}$ darf nicht überschritten werden.

Es gilt:

$$\begin{aligned} v(t) &= a_i t + v_i && \text{Entwicklung der Geschwindigkeit} \\ s(t) &= \tfrac{1}{2} a_i t^2 + v_i t && \text{Entwicklung der Strecke.} \end{aligned}$$

Um eine Strecke $s_i = s(t)$ zurückzulegen,

$$s_i = \tfrac{1}{2} a_i t^2 + v_i t$$

wird folgende Umformung benötigt:

$$t_{1,2} = \pm\frac{\sqrt{v_i^2 + 2s_i a_i}}{a_i} - \frac{v_i}{a_i}.$$

Für $v_{i+1} = v(t_{1,2})$ ergibt sich demnach

$$v_{i+1} = \pm\sqrt{v_i^2 + 2s_i a_i}.$$

Es ist zweckmäßig, sich die Quadrate der v_i anzuschauen, denn es gilt folgender Zusammenhang der Geschwindigkeiten durch Beschleunigung:

$$v_{i+1}^2 = v_i^2 + 2s_i a_i. \tag{11.12}$$

Nun wird über v_i^2 und a_i unter folgenden Bedingungen optimiert:

$\kappa_i v_i^2 \leq q_{\max}$	Begrenzung der Querbeschleunigung
$a_{\min} \leq a_i \leq a_{\max}$	Beschleunigungsgrenzen

Der schnellste Weg ist dann

$$\sum_{i=1}^{N} v_i^2.$$

Um das lineare Programm zu formulieren, wird $q_i = v_i^2$ gesetzt und die Bedingungen entsprechend umgeschrieben. Anschließend wird das folgende lineare Programm gelöst:

$$\begin{aligned} \text{maximiere} \quad & \sum_{i=1}^{N} v_i^2, \\ \text{s.t.} \quad & v_{i+1}^2 = v_i^2 + 2s_i a_i \\ & \kappa_i v_i^2 \leq q_{\max} \\ & a_{\min} \leq a_i \leq a_{\max} \end{aligned}$$

Wird das Geschwindigkeitsprofil auf die Trajektorie aus Abbildung 11.15 angewandt ergibt sich ein Profil, wie in Abbildung 11.16 zu sehen ist.

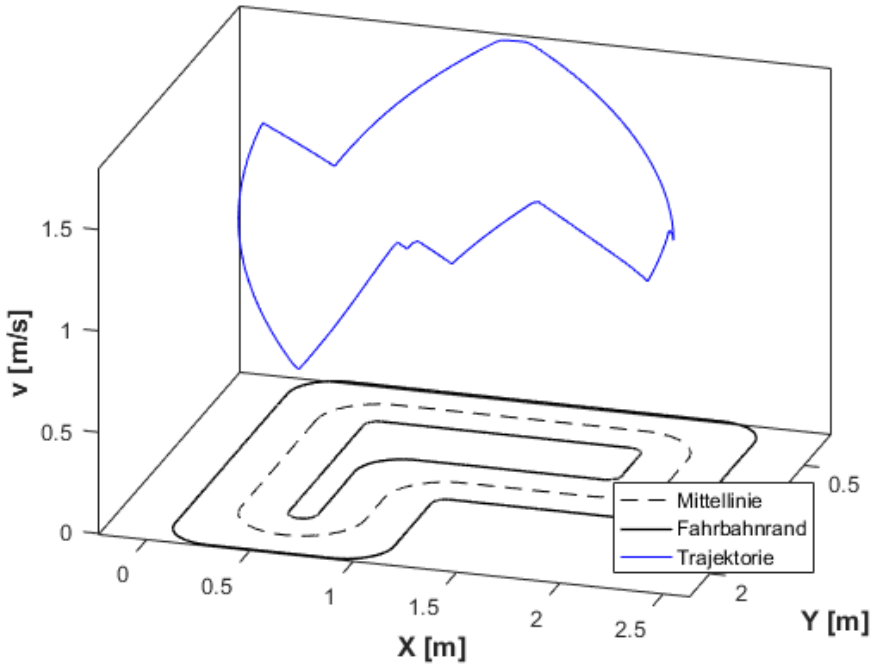


Abbildung 11.16.: Geschwindigkeitsprofil für die berechnete Trajektorie nach 10 Iterationen.

Integration in bisheriges Regelungssystem Durch das Zusammenfassen sämtlicher Planungsschritte ergibt sich ein Ablauf, wie er in Abbildung 11.17 zu sehen ist. Die berechnete Trajektorie wird als CSV exportiert und der Regelung aus der vorhergegangenen Projektgruppe übergeben. Da diese Regelung mit einer Repräsentation der Rennstrecke im CSV-Format arbeitet, konnte die Integration einfach erfolgen.

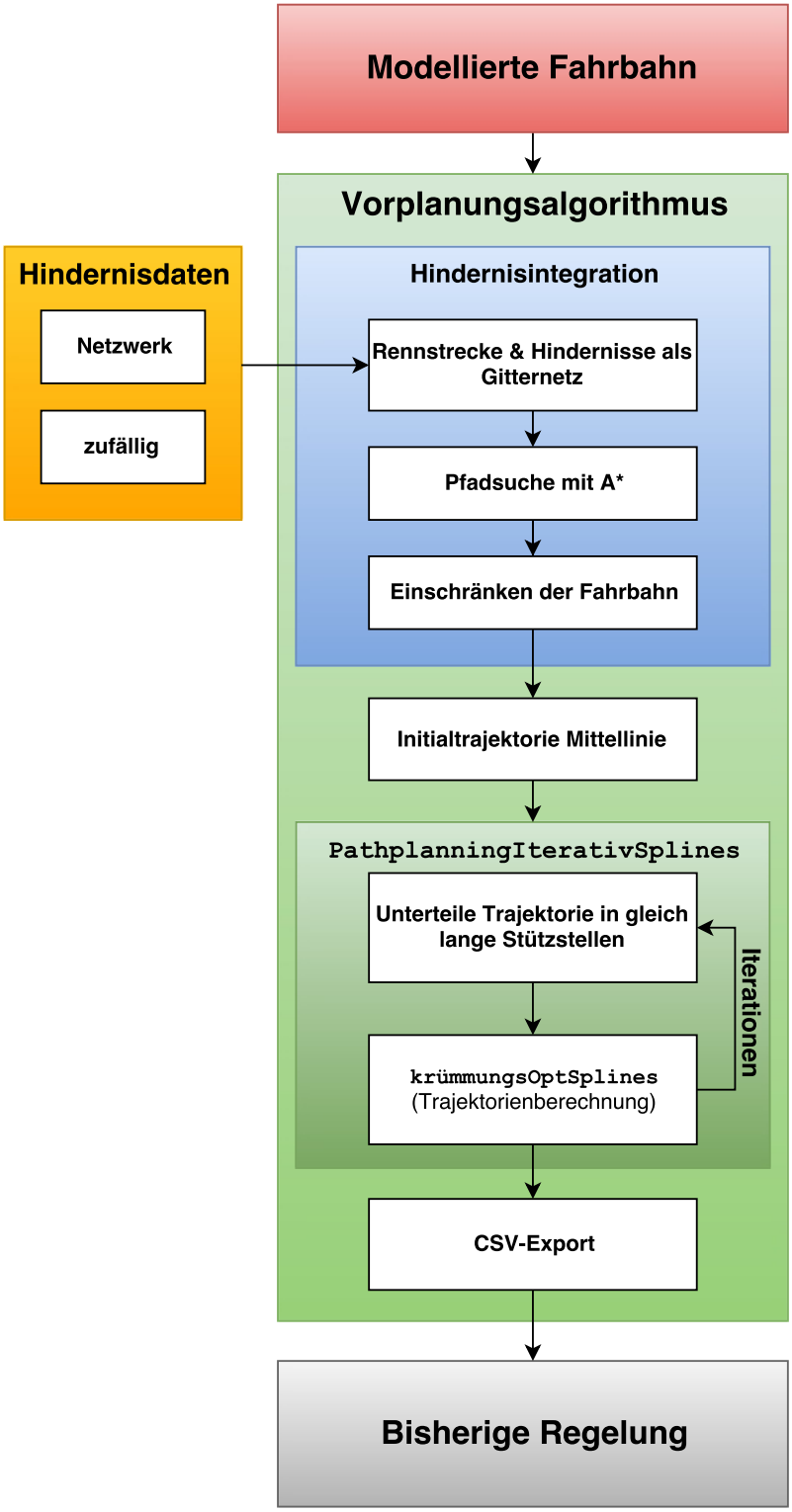


Abbildung 11.17.: Ablauf der Trajektorienplanung inklusive Hinderniserkennung.

11.3.6. Model Predictive Control (MPC)

Im Hinblick auf das vierte Zwischenziel wurde eine modellprädikative Regelung in Matlab simuliert. Grundlage für die Simulation bot die Masterarbeit von Tom Reske und der Semesterarbeit des Studenten Lukas Wunderli [Wun] von der ETH Zürich. Die MPC ist eine Methode zur Regelung. Der Unterschied zu einem herkömmlichen PID-Regler ist, dass durch eine MPC vorausberechnet werden kann, wie sich das Regelungssystem in der Zukunft verhalten wird. Dieses Verhalten kann durch mehrere Gleichungen beschrieben werden. Um eine Regelung durchzuführen, benötigt die MPC eine Referenztrajektorie und ein Fahrzeugmodell. Bei der Referenztrajektorie handelt es sich um die zuvor berechnete Trajektorie, wie in Abschnitt 11.3.5 beschrieben. Dadurch wird das System dynamischer und passt sich besser an die Gegebenheiten an.
Bei der MPC soll eine feste Folge von Kontrollgrößen ($\mathbf{u}_k$) gefunden werden, sodass die Summe aller Abweichungen von der Sollgröße ($\mathbf{x}_k$), die durch die Referenztrajektorie festgelegt ist, möglichst minimal ist. Dabei werden Fehlerquadrate der Fehlerzustände betrachtet, damit ein Aufheben entgegengesetzter Fehler nicht möglich ist, da die Fehler dadurch immer positiv sind.
Das System unterliegt bei der Berechnung bestimmten Bedingungen. So soll das Fahrzeug die Fahrbahn nicht verlassen und der Lenkwinkel ist begrenzt. Es gibt also nur einen erlaubten Bereich von Zuständen $\mathbf{X}$ und von Kontrollgrößen $\mathbf{U}$. Die mathematischen Grundlagen der MPC sind in Gleichung 11.13 beschrieben.

$$\begin{aligned} \underset{(\mathbf{u}_k)_{k=0,\dots,N}}{\arg\min} \quad & \sum_{k=0}^{N} \mathbf{x}_k^T Q \mathbf{x}_k \\ \text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}(t) \\ & \mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \\ & \mathbf{x}_k \in \mathbf{X} \\ & \mathbf{u}_k \in \mathbf{U} \end{aligned} \tag{11.13}$$

Es wird die feste Folge $k = 0, \dots, N$ betrachtet. Durch die positiv semidefinite Matrix Q erfolgt eine Gewichtung der Fehler. So können Fehler in der Position des Fahrzeugs stärker gewichtet werden als Fehler in der Geschwindigkeit. Durch die Gleichung $\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k$ wird der nachfolgende Zustand beschrieben, der anhand der angelegten Steuergröße und der momentanen Position erreicht wird. Durch $\mathbf{x}_k \in \mathbf{X}$ wird der befahrbare Bereich eingegrenzt. Werden die Hindernisse mit der Fahrbahnbegrenzung verbunden, wird diese Bedingung verändert.

Die Gleichung 11.13 kann erweitert werden, indem auch die Kosten für die Regelung hinzugenommen und durch $\sum_{k=0}^{N} \mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k$ optimiert werden. Um eine bessere Approximation von nicht linearen Systemen zu erreichen, kann die Gleichung 11.13 mit zeitlich verändernden Matrizen A_k und B_k versehen werden. Dies gilt gleichermaßen für die Kostenmatrizen Q, R und die zulässigen Mengen $\mathbf{X}$ und $\mathbf{U}$.
Verändert man die obige Gleichung mit den beschriebenen Maßnahmen erhält man folgende Gleichung:

$$\begin{aligned} \underset{(\mathbf{u}_k)_{k=0,\dots,N}}{\arg\min} \quad & \sum_{k=0}^{N} \mathbf{x}_k^T Q_k \mathbf{x}_k + \mathbf{u}_k^T R_k \mathbf{u}_k \\ \text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}(t) \\ & \mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k \\ & \mathbf{x}_k \in \mathbf{X}_k \\ & \mathbf{u}_k \in \mathbf{U}_k \end{aligned} \tag{11.14}$$

Typische Solver wie Gurobi setzen *konvexe* Contraints voraus. Deshalb werden aus den Constraints

$$\begin{aligned} \mathbf{x}_k &\in \mathbf{X}_k \\ \mathbf{u}_k &\in \mathbf{U}_k \end{aligned} \tag{11.15}$$

die Matrizen:

$$\begin{aligned} C_x \mathbf{x}_k &\leq \mathbf{c}_x \\ C_u \mathbf{u}_k &\leq \mathbf{c}_u. \end{aligned} \tag{11.16}$$

Bei der Verwendung von MPC als Controller werden durch die Gleichung 11.14 zukünftige Steuersignale berechnet. Es wird jedoch nur $\mathbf{u}_0$ als Kontrollsignal übermittelt. Es wird also, um ein Steuersignal zu berechnen, eine feste Anzahl (N) Schritte in die Zukunft geschaut. Dies ist ein Vorteil gegenüber PID, der erst nach einer Veränderung darauf reagiert. Durch den Blick in die Zukunft kann ein mögliches Hindernis frühzeitig erkannt werden.
Die Verwendung der MPC ist vor allem hinsichtlich des fünften Zwischenziels entscheidend. Um frühzeitig auf dynamische Hindernisse (Fahrzeuge) zu reagieren, soll die MPC die Trajektorie dynamisch auf die Gegebenheiten anpassen. Auch dynamische Fahrzeuge werden wie statische Hindernisse zur Berechnung einer Trajektorie einseitig mit der Streckenbegrenzung verbunden. In der Simulation wurde dazu ein Ansatz verfolgt, die Trajektorie dynamisch zu verschieben.

Dies wurde in einer Simulation bereits getestet. Bei der Simulation bleiben die verwendeten Hindernisse jedoch zunächst statisch, um zunächst die Ansätze zu testen. Dazu wurden eine Road (Fahrbahn), in der bereits Hindernisse mit der Wand verbunden wurden, und eine Trajektorie an die MPC übergeben. Die MPC testet daraufhin, ob die Trajektorie durch ein Hindernis führt. Ist dies der Fall, so entscheidet die MPC zu welcher Seite die Trajektorie verschoben werden soll. Diese Entscheidung wird mittels des Skalarproduktes durchgeführt. In Abbildung 11.18 ist die aufgezeichnete Fahrt des Fahrzeugs mit den Berechnungen der MPC mit und ohne dynamische Verschiebung der Trajektorie durch die MPC dargestellt. Dabei stellen die roten Abschnitte die Bereiche dar, die sich auf der rechten Seite der Mittellinie befinden, und die grünen Abschnitte die Bereiche dar, die sich auf der linken Seite der Mittellinie befinden. Durch diese Bereiche wird auch dargestellt, wo die eigentliche Trajektorie hin verschoben werden soll.

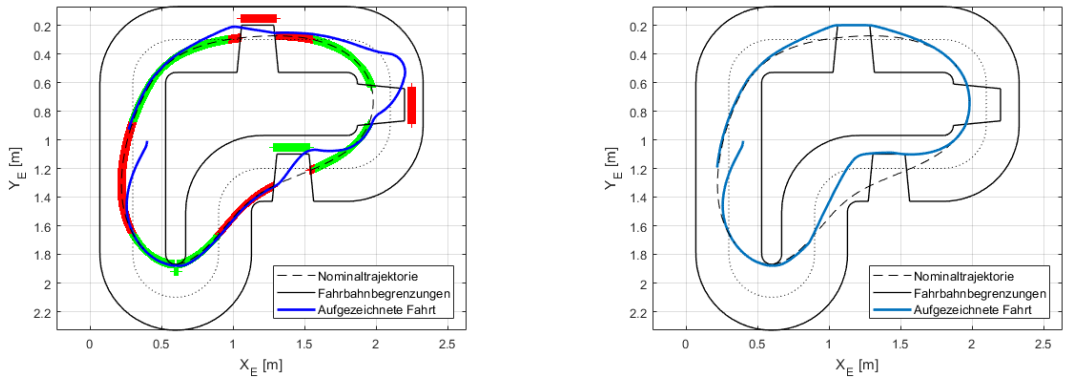


Abbildung 11.18.: MPC Aufzeichnung ohne dynamische Verschiebung und mit dynamischer Verschiebung

Es ist deutlich erkennbar, dass die Verschiebung der MPC versetzt zu den Hindernissen eintritt. Um diesen Fehler zu korrigieren, wird ein Ansatz verwendet, der den Fehler der MPC vergrößert, sobald die Trajektorie durch ein Hindernis führt. Durch die Fehlererhöhung soll die MPC schneller auf das Hindernis reagieren und Verzögerungen reduziert werden. Bei der Aufzeichnung ohne eine Korrektur der Trajektorie durch die MPC verhält sich das Fahrzeug hinsichtlich des ersten Hindernisses nach der 180 ° Kurve und letzten Hindernisses vor der 90 ° Kurve vor der Start-/Ziellinie besser als bei der Aufzeichnung mit Korrektur.

11.3.7. Hardwareentwicklung Control-Unit

Da für die Erfüllung des Szenarios „Unfallfreier Überholvorgang“ zwei Fahrzeuge zum Einsatz kommen sollen, muss die in Kapitel 5.3.1 beschriebene CarControl zweifach vorhanden sein. Da diese bisher nur einmal vorhanden war, wurde von der Projektgruppe *RCCARSng* eine zweite CarControl hergestellt (siehe Abbildung 11.19).

(a) Arbeitsstelle zur Herstellung der zweiten CarControl.

(b) Fertiggestellte zweite CarControl.

Abbildung 11.19.: Herstellung der zweiten CarConrol.

11.4. Subsystem 4: Rennstrecke

Die Rennstrecke wurde um frei platzierbare Hindernisse erweitert. Diese gehören zur Rennstrecke, da sie den befahrbaren Bereich auf dieser statisch einschränken. Hindernisse sind aus einem weichen Schaumstoff um, im Falle einer Kollision, Schäden an den Fahrzeugen zu vermeiden. Die Form der Hindernisse ist auf Quader beschränkt. Die Markierungen der Hindernisse sind auf die Funktionsweise der Positionsbestimmung angepasst. Da verschiedene Marker in der Positionsbestimmung nur anhand der Fläche unterschieden werden, wurde der erste Hindernisentwurf verworfen (siehe Abbildung 11.20) und ein zweiter Entwurf angefertigt. Die neuen Hindernisse besitzen einen 1 cm breiten Streifen Reflexionsfolie entlang der gesamten Grundfläche. Der Streifen muss an den äußeren Kanten der Fläche der Hindernisse angebracht sein, da es sonst zu Fehlern in der Erkennung kommen kann. Die Positionsbestimmung erkennt die Objekte anhand dieser Markierungen. In Abbildung 11.21 ist zusehen wie die Positionsbestimmung die Hindernisse erkennt.

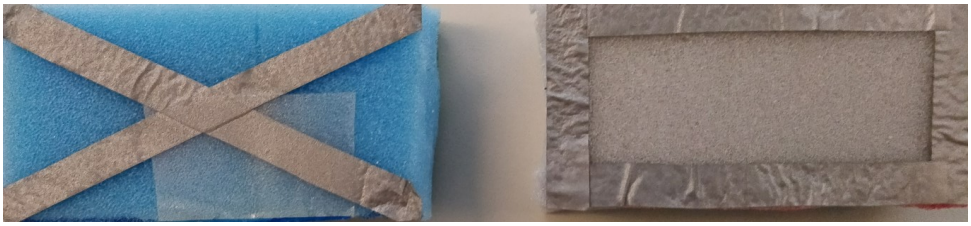


Abbildung 11.20.: Gegenüberstellung der Hindernisse. Links die alte Variante, Rechts die Neue.

Abbildung 11.21.: Gegenüberstellung der Hindernisse, so wie die Positionsbestimmung sie sieht. Links die alte Variante, Rechts die Neue.

11.5. Subsystem 5: Systemsteuerungssoftware

Die Systemsteuerungssoftware (SSSW) wurde so erweitert, dass eine Kommunikation mit der Positionsbestimmung und der Control-Unit auch mit neuen Netzwerkdaten weiterhin möglich ist. Zudem wurde die Bedienungsmaske dahingehend angepasst, dass nun mehrere Fahrzeuge über die Oberfläche verwaltet werden können. Dies ist in Abbildung 11.22 zu sehen. Da die Bedienung über die SSSW allerdings kein nötiger Bestandteil der Zwischenziele darstellt, wurde die Entwicklung pausiert und auf einen späteren Zeitpunkt der Entwicklungsphase verschoben.

11.6. Netzwerk

Die Kommunikation der einzelnen Rechner des Systems läuft im Wesentlichen noch so, wie es im Bericht [RCC16] im Abschnitt 4.5 beschrieben wurde. Die Rechner sind über eine Switch verbunden und es wird weiterhin UDP als Kommunikationsprotokoll genutzt. Allerdings musste das Netzwerkpaket so angepasst werden, dass es den Ansprüchen der Schnittstellenbeschreibung in Kapitel 9 genügt, indem nun Konfigurationsdaten und Hindernisdaten verschickt werden können. Ein Netzwerkpaket enthält nun die Information, wann es versendet wurde, wann es empfangen wurde, sowie einen Identifier, der angibt, was für Daten dieses Paket enthält und die eigentlichen Nutzdaten.

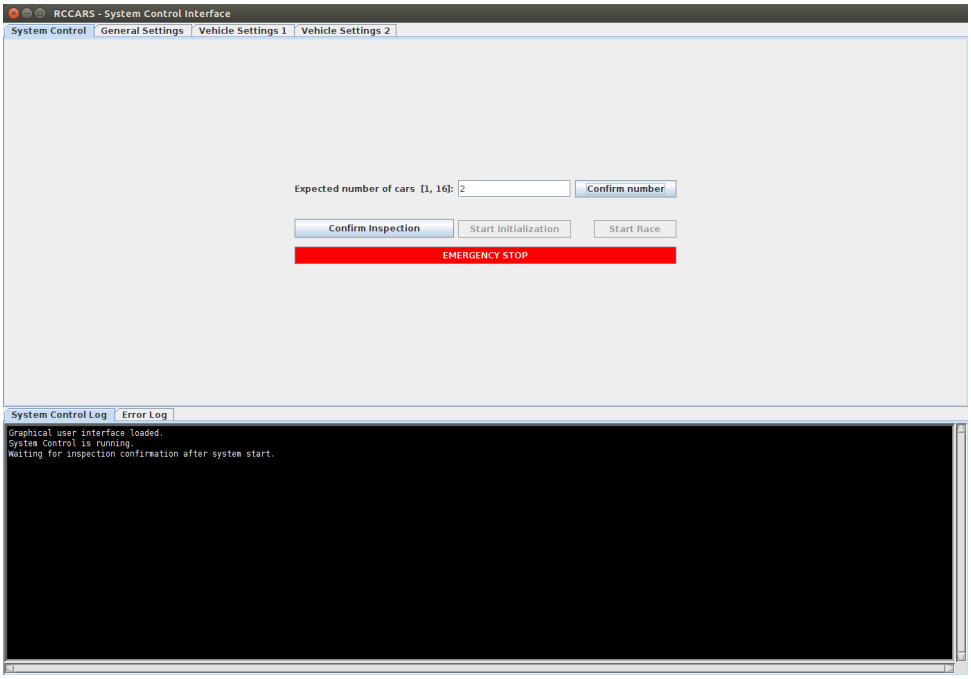


Abbildung 11.22.: Graphische Benutzeroberfläche der Systemsteuerungssoftware.

Fahrzeugdaten besitzen die ID 1 und enthalten Informationen über bis zu 16 Fahrzeuge. Hier kann ausgelesen werden, wie viele Fahrzeuge vorhanden sind, welche X und Y Koordinaten und welche Ausrichtung sie bezüglich der Rennstrecke haben.

Kontrolldaten besitzen die ID 2 und enthalten eine ID und einen Systemzustand. Diese Daten sollen ein Systemzustandswechsel eines Subsystems anzeigen. Die Zuordnung der IDs findet sich im Anhang in Tabelle A.1 und die verschiedenen möglichen Systemzustände in Tabelle A.2.

Fehlerdaten besitzen die ID 3 und enthalten einen Fehlercode und ein Flag für einen eventuellen Not-Aus Befehl. Die Fehlercodes finden sich im Anhang in Tabelle C.1.

Zeitdaten besitzen die ID 4 und enthalten die ID einer Control-Unit, sowie die Zeitdaten, die nötig sind, um die gesamte Dauer vom Senden der Fahrzeugdaten bis zum Senden der Steuerbefehle an die Fahrzeuge zu bestimmen.

Hindernisdaten besitzen die ID 5 und enthalten die Anzahl der gefundenen Hindernisse und die X und Y Koordinaten ihrer Eckpunkte.

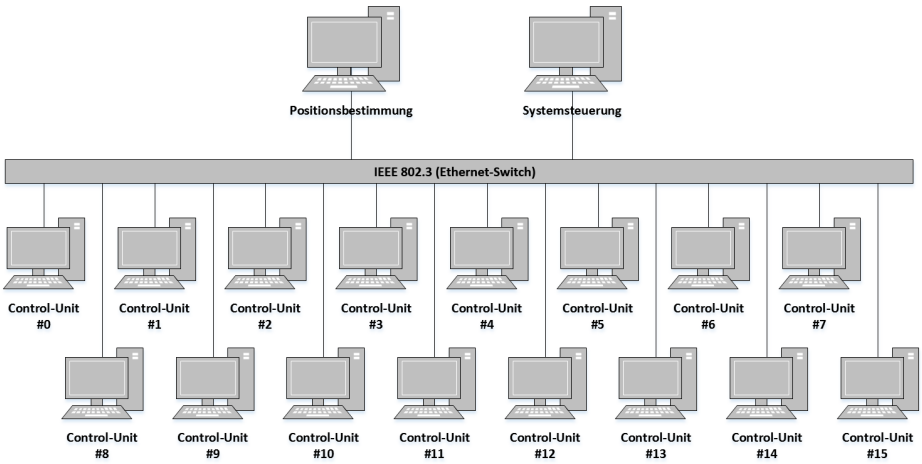


Abbildung 11.23.: Schematische Netzwerkkommunikation.

Konfigurationsdaten besitzen die ID 6 und enthalten die Zuweisung von Fahrzeugen an eine Control-Unit, das Fahrverhalten für die jeweiligen Fahrzeuge und die Mindestdurchschnittsgeschwindigkeiten für die Fahrzeuge.

Die Netzwerkpakete werden aktuell noch via Broadcast an alle Netzwerkteilnehmer (siehe Abbildung 11.23) versendet. Im Laufe der Projektgruppe soll dieses durch Multicast ersetzt werden, sodass gezielte Gruppen von Netzwerkteilnehmern angesprochen werden können.

12. Entwicklungsstand bis zum 06.04.2018

In diesem Kapitel sind Änderungen und Entwicklungen, die in der Zeit vom 15.11. 2017 bis zum 06.04.2018 erfolgten, beschrieben. Sofern nicht anders beschrieben, bleiben Systembeschreibungen aus den vorherigen Kapiteln weiterhin gültig. Die Beschreibung erfolgt getrennt für jedes Subsystem.

12.1. Subsystem 1: Fahrzeug

Für das Subsystem 1 wurden die in Tabelle 3.3 aufgelisteten Fahrzeugersatzteile bestellt und in die defekten Fahrzeuge eingebaut. Weitere Änderungen wurden nicht vorgenommen.

12.2. Subsystem 2: Positionsbestimmung

In diesem Abschnitt werden die Änderung an der Soft- und Hardware der Positionsbestimmung beschrieben.

12.2.1. Markertrennung

Die Software der Positionsbestimmung musste aufgrund der Änderungen am Fahrzeug (siehe Abschnitt 11.1) erweitert werden. Da diese Änderungen die Größe der Fahrzeugheckmarker verändert haben, reicht die Größe in Pixeln nicht mehr als Merkmal aus. Dies wurde während der Integrationstests des Systems festgestellt. Die Positionsbestimmung hat während des Betriebs verschiedene Markertypen falsch zugeordnet. Somit wurden Fahrzeuge, trotz idealer Rennstreckenausleuchtung, nicht erkannt. Die Trennung der Fahrzeugmarkierung anhand der Größe ist in Abbildung 12.1 zu sehen. In diesem Bild ist zu sehen, dass die ID-Marker und Rennstreckenmarker gut von den anderen Markertypen getrennt sind, die Fahrzeugfrontmarker sind allerdings schlecht von den Fahrzeugheckmarkern zu trennen. Somit müssen

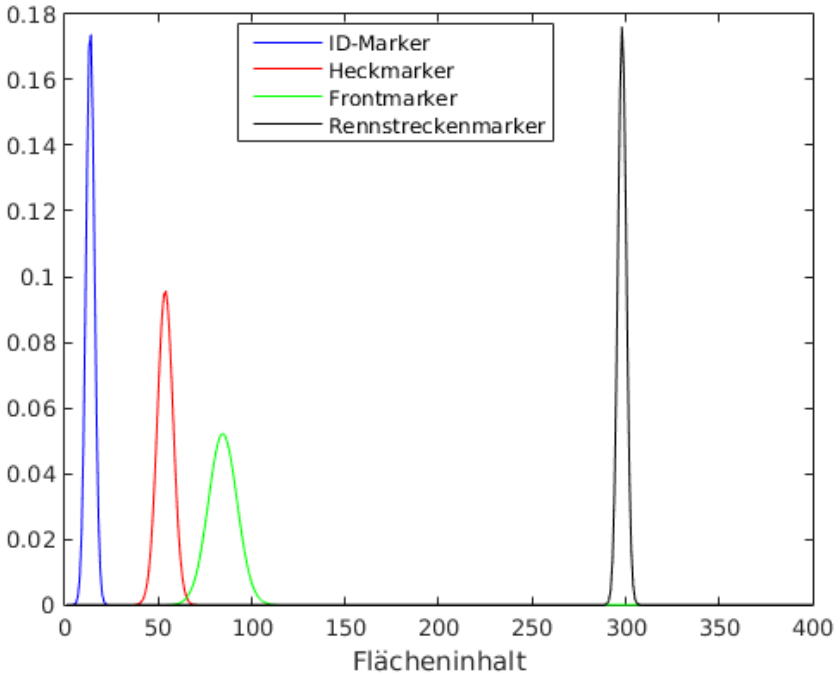


Abbildung 12.1.: Trennung der Marker anhand der Größe.

andere Merkmale evaluiert werden. Die Library OpenCV stellt diverse Erkennungsmerkmale zur Verfügung. Durch die OpenCV-Methode `moments` lassen sich Bildmomente berechnen. Bildmomente beschreiben Eigenschaften, wie die Fläche und den geometrischen Mittelpunkt$(\bar{x}, \bar{y})$, der gefundenen Objekte. Momente werden mittels der Formel 12.1 berechnet. Diese Momente sind abhängig von der Ausrichtung eines Objektes. Die Abhängigkeit bezüglich der Ausrichtung und Bewegung muss entfernt werden, da sich die Fahrzeuge während des Betriebs bewegen und um die eigene Achse drehen. Somit werden die zentralen Momente eines Objektes berechnet (siehe Formel 12.2). Aus diesen Eigenschaften lässt sich die Exzentrizität (siehe Formel 12.4) eines Objektes bestimmen. Die Exzentrizität beschreibt die Halbachsen der minimalen Ellipse um das Objekt.

$$M_{ij} = \sum_x \sum_y x^i y^j g(x, y), \tag{12.1}$$

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j g(x, y), \tag{12.2}$$

$$\mu'_{20} = \frac{\mu_{20}}{\mu_{00}} = \frac{M_{20}}{M_{00}} - \bar{x}^2, \quad \mu'_{02} = \frac{\mu_{02}}{\mu_{00}} = \frac{M_{02}}{M_{00}} - \bar{y}^2, \quad \mu'_{11} = \frac{\mu_{11}}{\mu_{00}} = \frac{M_{11}}{M_{00}} - \bar{x}\bar{y}, \tag{12.3}$$

$$\lambda_i = \frac{\mu'_{20} + \mu'_{02}}{2} \pm \frac{\sqrt{4\mu'^2_{11} + (\mu'_{20} - \mu'_{02})^2}}{2}. \tag{12.4}$$

Aufgrund dieser Eigenschaften konnte die Trennung verbessert werden. Abbildung 12.2 zeigt die Trennung nach Exzentrizität der Marker. Für jeden Markertyp wurde hier die der Erwartungswertvektor $\mu = \begin{pmatrix} \mu_{\lambda 1} \\ \mu_{\lambda 2} \end{pmatrix}$ aufgetragen und die Streuung um diesen. Eckmarker haben einen Erwartungswertvektor von $\mu_e = \begin{pmatrix} 26 \\ 23 \end{pmatrix}$, ID-Marker von $\mu_i = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$, Frontmarker von $\mu_f = \begin{pmatrix} 8 \\ 2 \end{pmatrix}$ und Heckmarker von $\mu_h = \begin{pmatrix} 22 \\ 8 \end{pmatrix}$.

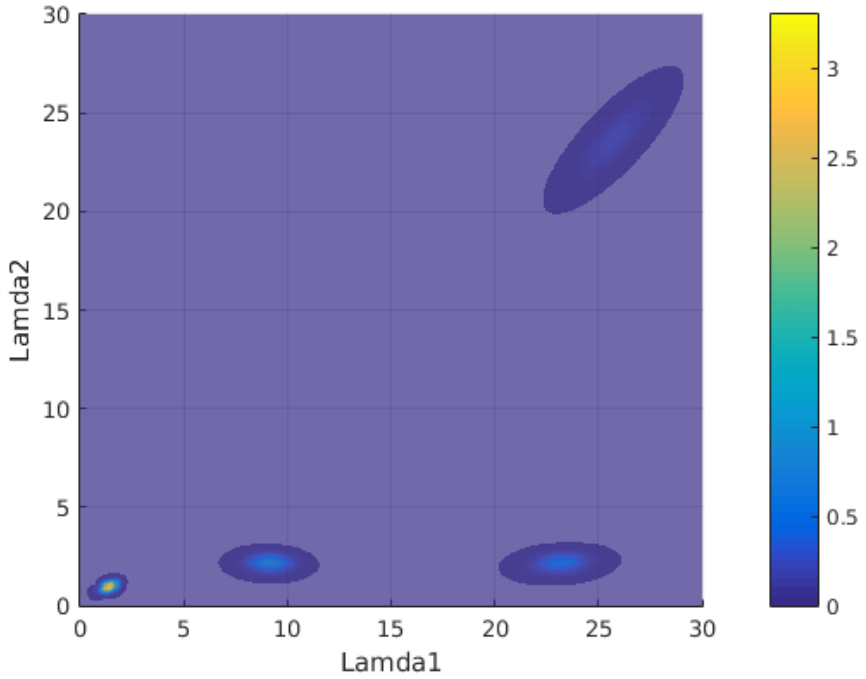
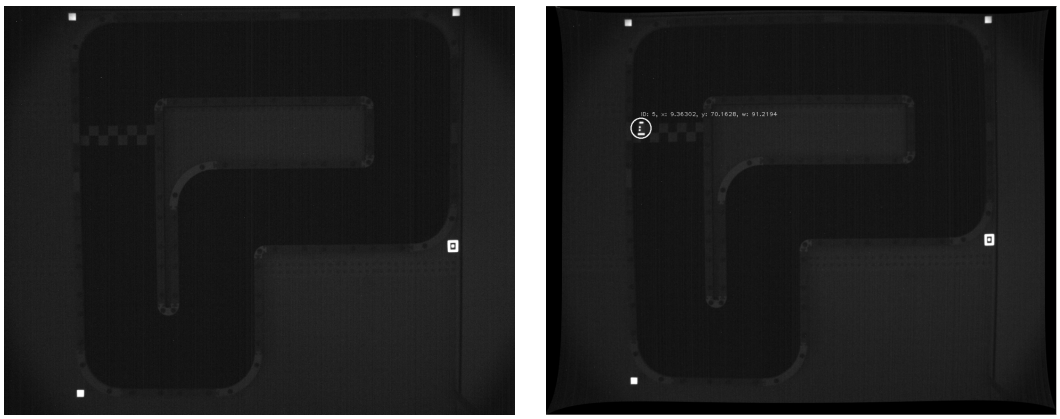


Abbildung 12.2.: Trennung nach mehreren Eigenschaften.

12.2.2. Markervarianzen

Neben den Problemen mit der Trennung von Markern kamen noch weitere Probleme hinzu. Diese lassen sich aus dem Kameraaufbau ableiten. Durch das verwendete Weitwinkelobjektiv wird das von der Kamera aufgenommene Bild verzerrt. Diese Verzerrung kann durch die in der Config-Datei (siehe Tabelle B.1) dargestellten Verzerrungskoeffizienten teilweise entfernt werden. Der Unterschied zwischen einem verzerrten und entzerrten Bild ist in Abbildung 12.3 zu sehen. Das Entzerren ist nicht perfekt, es bleiben Verzerrungsartefakte im entzerrten Bild vorhanden. Deshalb verändert sich die gemessene Größe und Form der Fahrzeugmarkierungen je

nachdem, wo sich das Fahrzeug auf der Rennstrecke befindet. Abbildung 12.4 stellt diese positionsabhängige Größe eines Fahrzeugfrontmarkers dar. Die Position ist hier in Bildkoordinaten und die Größe in Pixeln angegeben. Aus diesem Bild kann man die minimale und maximale Größe dieses Markers abgelesen werden. Minimal hat der Marker eine Größe von 65 Pixeln und maximal von 110 Pixeln. Um diese Veränderungen auszugleichen, werden hier die in Abschnitt 4.3 beschriebenen mehrdimensionalen Wahrscheinlichkeitsverteilungen angewendet.



(a) Verzerrtes Bild (b) entzerrtes Bild

Abbildung 12.3.: Gegenüberstellung eines verzerrten und entzerrten Bildes

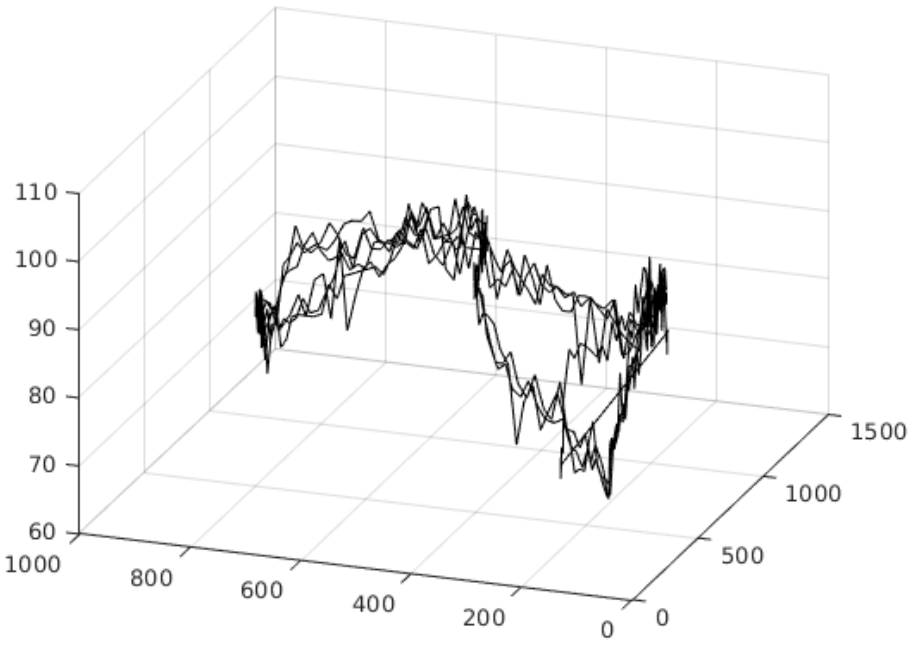


Abbildung 12.4.: Positionsabhängige Größenveränderung der Frontmarker.

Da sich die Formen der Objekte verändern, verändern sich auch die Exzentrizitäten dieser. So wurden mehrere Messungen durchgeführt und aus diesen ein Erwartungswertvektor und eine Kovarianzmatrix für jeden Markertyp berechnet. Diese sind ebenfalls in der Config-Datei einzusehen. Während des Rennbetriebes werden diese Erwartungswerte und die zugehörigen Varianzen, nachdem alle Marker erkannt und richtig zugeordnet wurden, aktualisiert. Die Zuordnung geschieht über die Dichtefunktion (siehe Formel 4.10). So wird für die Fahrzeugfront-, Fahrzeugheck- und ID-Marker die Dichte bezüglich aller Typen bestimmt und das Maximum dieser ausgewählt. Die Matrixmultiplikationen geschehen mittels der Armadillo-Library [SC16].

12.2.3. Rennstreckenrotation

Damit eine Rotation der Rennstrecke automatisch erkannt werden kann, wurde ein neuer Markertyp (siehe 12.4) zu der Rennstrecke hinzugefügt. Dieser hat wie die Hindernisse eine eindeutige Konturhierarchie. Somit dient dieser Marker als Referenzmarker, da dieser eindeutig identifizierbar ist und immer als letztes zur Sammlung der Eckmarker hinzugefügt wird. Anschließend wird der Schwerpunkt aller Eckmarker berechnet. Da die Reihenfolge der Eckmarker bisher von dem Erkennungsalgorithmus von OpenCV abhängig war, müssen diese noch sortiert werden. Daher werden Vektoren vom Schwerpunkt zum Eckpunkt gebildet. Die Sortierung geschieht anhand des Winkels, wobei der Vektor vom Schwerpunkt zum besonderen Eckmarker einen Winkel von $0\,^\circ$ hat. So kann eine Rotation erkannt werden.

12.2.4. Deckenhalterung

Ein wesentlicher Bestandteil der Hardware der Positionsbestimmung war das Gestell zur Befestigung der Kamera und IR-Strahler. Es bestand aus zwei Standbeinen und einer verbindenden Stange, an der Kamera und Strahler montiert sind.
Dieser Aufbau nahm relativ viel Platz im Projektraum in Anspruch. Außerdem sind die Einstellungsmöglichkeiten, wie zum Beispiel das Ausrichten der Kamera, begrenzt. Aus diesem Grund wurde eine neue Halterung in Zusammenarbeit mit der Uni Werkstatt in Wechloy entwickelt. Die Projektgruppe hat hierzu einen Entwurf an die Werkstatt heran getragen, der in Kooperation verfeinert und letzten Endes hergestellt wurde. Die wichtigsten Punkte hierbei waren das Wegfallen der Standbeine, da viel Raum verloren ging und die Rennstrecke ohne diese bis in die Raumecke gerückt werden kann. Außerdem beinhaltet die neue Halterung eine Höhenverstellung, um so bequem Kamera und Strahler ausrichten zu können.

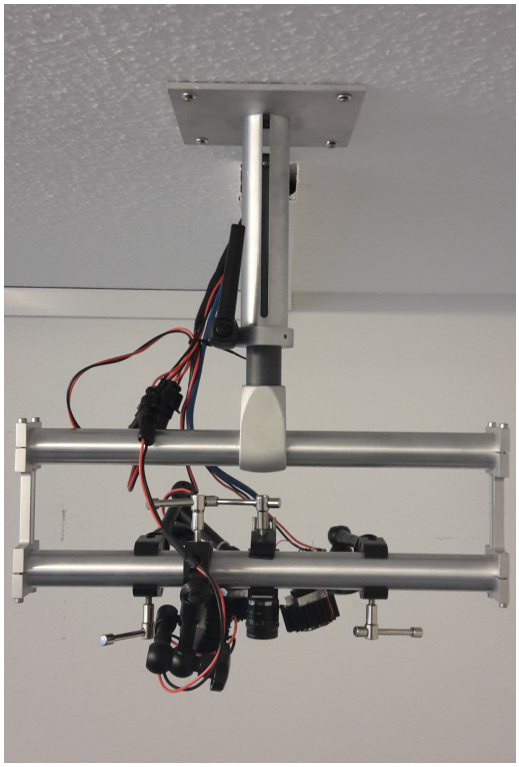
Die Abbildung 12.5 zeigt die entwickelte Deckenhalterung. Die Halterung ist an der Decke des Projektraums montiert, Kabel der Kamera und IR-Strahler verlaufen über Kabelkanäle bis zum Positionsbestimmungsrechner. Die Deckenhalterung verfügt über einen Teleskoparm, der es zulässt, die Höhe zwischen 2.05 m und 2.23 m Meter vom Boden zu variieren. Das alte Gestell war auf einer Höhe von 2.19 m Metern fest eingestellt. Durch die Höhenverstellbarkeit lässt sich die Kamera so ausrichten, dass die Rennstrecke einen möglichst großen Teil des Bildes einnimmt.
Ein weiteres Feature der Deckenhalterung ist ihre Mobilität. Der untere Teil lässt sich durch zwei Schellen lösen (siehe Abbildung 12.5(b)) und an das alte Gestell montieren. So ist es möglich, Kamera und IR-Strahler mit gleicher Ausrichtung an das Gestell zu schrauben, was sich zum Beispiel für Messebesuche als sehr praktisch erweisen kann, da keine weiteren Einstellungen an dem System notwendig sind.

12.3. Subsystem 3: Control-Unit

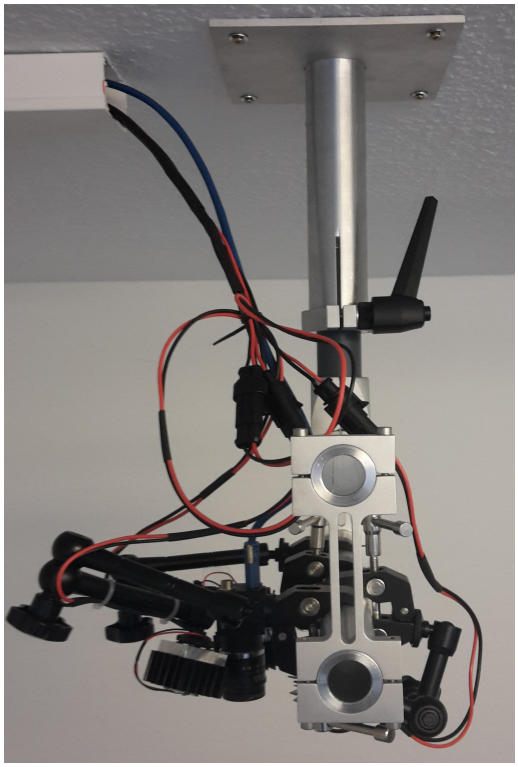
Die Control-Unit ist, wie in Abschnitt 9.1.3 beschrieben, in verschiedene Module unterteilt. In diesem Abschnitt wird ihre Funktionen und Implementierung beschrieben. Jedes Modul hat eine ID für die interne Kommunikation. Diese sind in Tabelle 12.1 aufgelistet.

Modul-nummer	**Modul-nummer Hex**	**Modul**
1	0x01	Vorplanung
2	0x02	Kollisionserkennung
3	0x03	Kommunikationsinterface
16	0x10	Konfigurationsverarbeitung
17	0x11	Statusmanagement
18	0x12	Plausibilitätskontrolle
19	0x13	Watchdog
20	0x14	Fehlermanagement
21	0x15	Realzeitüberprüfung
22	0x16	Renndatenberechnung
23	0x17	SerialControl
32-47	0x20 - 0x2F	Regelungsmodule

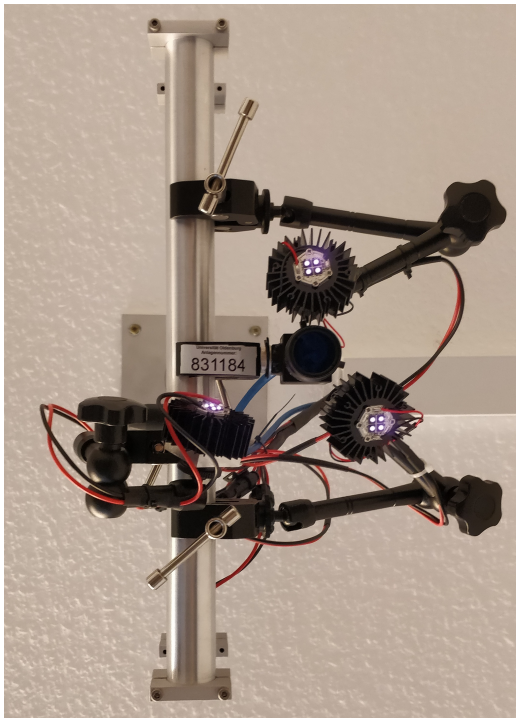
Tabelle 12.1.: Kodierung der Control-Unit Modulnummern.



(a) Ansicht von vorne.



(b) Ansicht von der Seite.



(c) Ansicht von unten.

(d) Gesamtansicht mit Rennstrecke.

Abbildung 12.5.: Deckenhalterung mit Kamera und Infrarotstrahlern in verschiedenen Perspektiven.

12.3.1. Vorplanung

In diesem Abschnitt wird die Implementierung der Vorplanung beschrieben. Dazu erfolgt zunächst eine Beschreibung des Programmablaufs. In den nachfolgenden Abschnitten wird auf die Modifikation des Geschwindigkeitsprofils hinsichtlich Durchschnittsgeschwindigkeit, Trajektorienberechnung mit Hindernissen, Berechnung von Überholtrajektorien und Behandlung von Fehlern, die von der Vorplanung geworfen werden, näher eingegangen, die im Wesentlichen in der Zeit bis zum 06.04. für die Vorplanung entwickelt wurden.

Ablauf der Vorplanung

Die Vorplanung ist nach Modulbeschreibung (siehe Abschnitt 9.1.3) für die Berechnung der Planungsdaten (Trajektorien) zuständig. Hierbei entnimmt sie dem zugewiesenen Fahrverhalten, welche Trajektorien berechnet werden müssen. Ist Fahrverhalten 0 ausgewählt, was keinem Fahrverhalten entspricht, so findet keine Berechnung statt und die Vorplanung wird beendet. Nur für Fahrverhalten größer 0 werden Berechnungen durchgeführt.
In der Trajektorienberechnung kommt die Matlab-Software, die bereits in den Abschnitten 11.3.1 - 11.3.5 beschrieben wurde, zum Einsatz. Da der Hauptteil der Control-Unit in C++ implementiert ist, wird eine von Matlab bereitgestellte Schnittstelle zu C++ verwendet. Diese stellt eine Matlab-Engine zur Verfügung, mit der Matlab-Code ausgewertet und ausgeführt werden kann. Somit kann die unter Matlab entwickelte Software weiterhin verwendet werden.

Zur Berechnung von Planungsdaten werden statische Umgebungsdaten und Vorplanungsparameter (siehe Abschnitt 9.2.3) benötigt, die aus der Konfigurationsverarbeitung entnommen und als Matrix an die Matlab-Engine weitergeleitet werden. Aus den statischen Umgebungsdaten wird eine Rennstreckenklasse inklusive Hindernisse initialisiert, welche Informationen über die Koordinaten der Mittellinie der Rennstrecke, Fahrbahnrand und Hindernissen zur Verfügung stellt. Außerdem ist eine Methode zur Visualisierung der Rennstrecke verfügbar. Die Vorplanungsparameter werden in einer Parameterklasse gespeichert.

Sind statische Umgebungsdaten und Parameter verarbeitet, startet die Vorplanung mit der Berechnung der Trajektorien. Hierbei wird anhand des Fahrverhaltens entschieden, welche Trajektorien relevant sind und berechnet werden müssen. Für Fahrverhalten 1, „passives Fahren“, erfolgt die Berechnung einer krümmungsoptimierten Trajektorie gemäß Abschnitt 11.3.5 und eine Anpassung des Geschwindigkeitspro-

fils hinsichtlich einer gewünschten Durchschnittsgeschwindigkeit $vAverage$. Die Geschwindigkeitsanpassung ist im nachfolgenden Abschnitt 12.3.1 beschrieben. Falls Fahrverhalten 2, „Überholen“, oder 3, „Überholt werden“, eingestellt ist, werden aus der krümmungsoptimierten Trajektorie zwei Trajektorien für ein Überholmanöver berechnet. Eine dieser Trajektorien ist für die standardmäßige Fahrt vorgesehen, die andere für das überholende Fahrzeug im Überholmanöver.
Nach Abschluss der Berechnungen speichert die Vorplanung alle vorhandenen Trajektorien separat als Planungsdaten ab und wird beendet. Der Ablauf der Vorplanung ist in Abbildung 12.6 schematisch dargestellt.

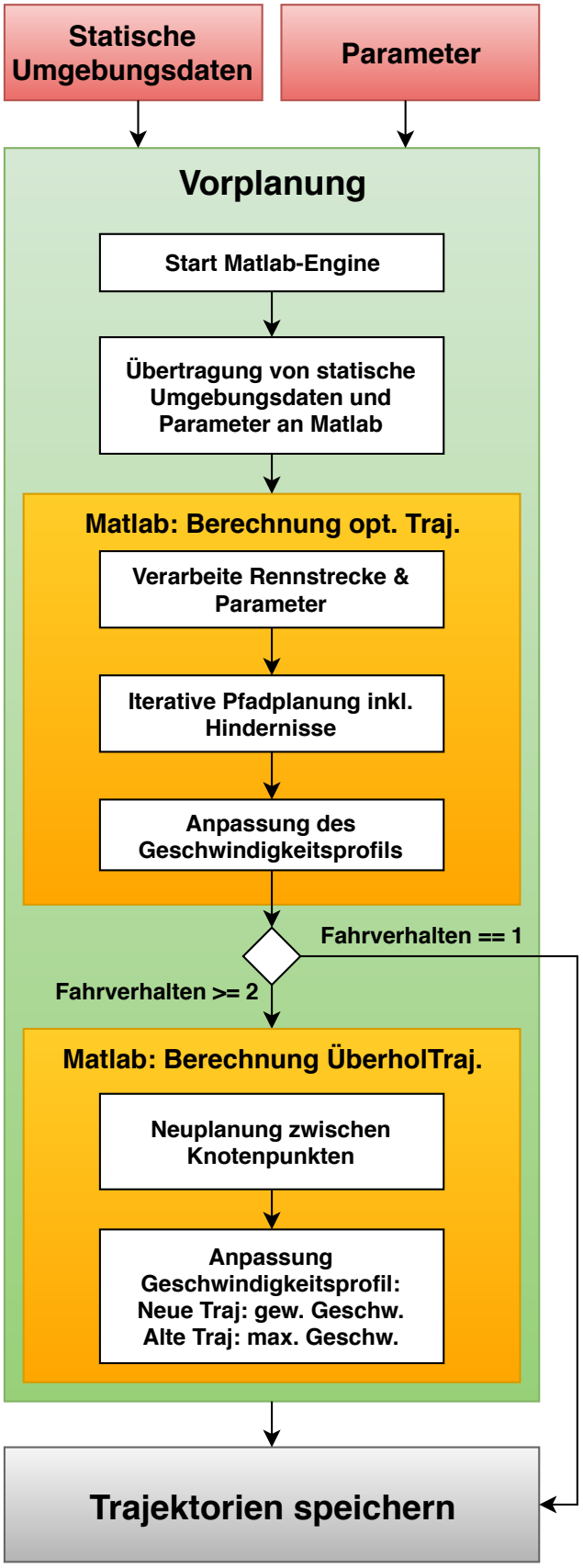


Abbildung 12.6.: Ablaufdiagramm der Vorplanung.

Modifikation des Geschwindigkeitsprofils hinsichtlich Durchschnittsgeschwindigkeit

Damit die gewünschte Durchschnittsgeschwindigkeit $vAverage$ eingehalten werden kann, muss die Berechnung der Geschwindigkeit, wie sie in Abschnitt 11.3.5 beschrieben ist, um die Möglichkeit einer nachträglichen Anpassung erweitert werden. Das Geschwindigkeitsprofil einer Trajektorie wird wie in Abschnitt 11.3.5 durch ein Optimierungsproblem bestimmt, dass unter Berücksichtigung der möglichen Beschleunigung die Geschwindigkeit maximiert. Es gibt für jede Stützstelle i der Trajektorie die Geschwindigkeit v_i und die Beschleunigung a_i zurück. Der Ausdruck

$$\bar{v}_{max} = \frac{1}{n} \sum_{i=1}^{n} v_i \tag{12.5}$$

gibt die Durchschnittsgeschwindigkeit dieses Profils an. Da v_i im Optimierungsproblem maximiert wird, ist $\bar{v}_{max}$ die maximal mögliche Durchschnittsgeschwindigkeit. Zur Anpassung der Geschwindigkeit wird die maximal mögliche Geschwindigkeit skaliert. Dazu wird $t = \frac{vAverage}{\bar{v}_{max}}$ bestimmt und mit jeder Geschwindigkeit v_i multipliziert, sodass sich die gewünschte Durchschnittsgeschwindigkeit ergibt. Für die Beschleunigung a_i gilt nach Gleichung 11.12:

$$\begin{aligned} v_{i+1}^2 &= v_i^2 + 2 s_i a_i \\ \Rightarrow a_i &= \frac{v_{i+1}^2 - v_i^2}{2 s_i}. \end{aligned}$$

Für die neue Beschleunigung wird v_i durch $v_i \cdot t$ ersetzt. Es ergibt sich:

$$a_i Adapted = \frac{t^2 \cdot (v_{i+1}^2 - v_i^2)}{2 s_i} = t^2 \cdot a_i. \tag{12.6}$$

Daher wird jeder Beschleunigungswert mit t^2 multipliziert. Ein beispielhaft angepasstes Geschwindigkeitsprofil ist in Abbildung 12.7 zu sehen.
Die obige Anpassung ist nur für Durchschnittsgeschwindigkeiten kleiner als $\bar{v}_{max}$ sinnvoll. Falls $vAverage$ größer als $\bar{v}_{max}$ ist, gibt Matlab eine entsprechende Fehlermeldung aus.

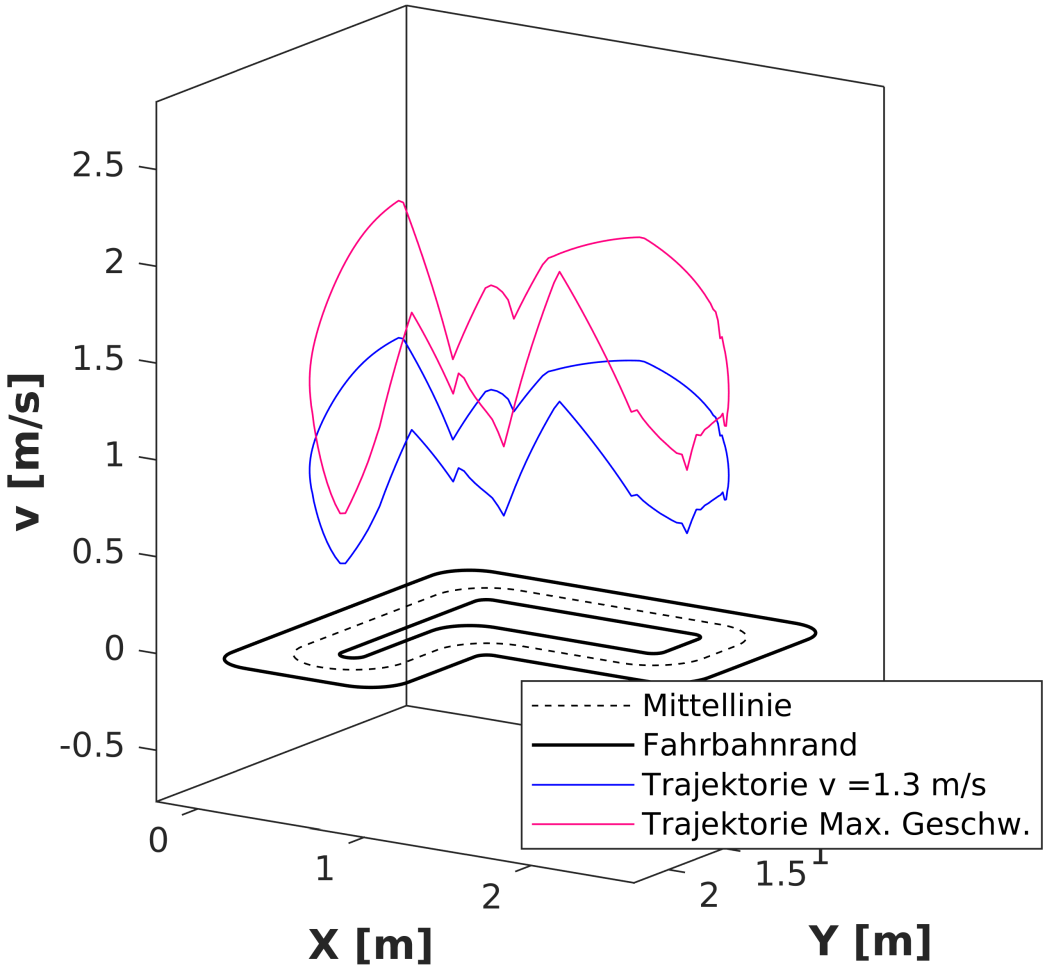


Abbildung 12.7.: Vergleich des angepassten Geschwindigkeitsprofils mit der ursprünglich maximal möglichen Geschwindigkeit.

Trajektorienberechnung mit Hindernissen

In Abschnitt 11.3.4 wurde eine Methode vorgestellt, die Trajektorienplanung mit Hindernissen realisieren kann. Sie sieht vor, den Fahrbahnrand so zu modifizieren, dass Hindernisse vom Fahrbahnrand eingeschlossen sind und nur auf einer Seite befahrbarer Bereich vorhanden ist. Diese Methode verwendet die Projektgruppe *RC-CARSng* nicht mehr, da es zum Einen Probleme in der Implementierung gab, zum Anderen die Rennstrecke veränderte. Da der ursprüngliche Fahrbahnrand nicht mehr vorhanden war, wäre es nicht mehr möglich gewesen Wege auf beiden Seiten des Hindernisses zu planen.
Zur Hindernisintegration wird stattdessen eine neue Methode verwendet, die für jeden Punkt auf der Rennstrecke den befahrbaren Bereich nach innen und außen angibt. Die Berechnung ist wie folgt:

Sei $P = (p_x, p_y)$ ein Punkt, für den der befahrbare Bereich berechnet werden soll und $\vec{v}$ der zugehörige Normalenvektor auf der Mittellinie in Richtung des äußeren Fahrbahnrandes. Ferner beschreibt $\vec{n_i}$ den Normalenvektor einer Hindernisseite und b_i den Abstand der Hindernisseite zum Ursprung. Wird das Skalarprodukt von $\vec{n_i}$ mit einem beliebigen Ortsvektor gebildet und mit b_i verglichen, so lässt sich ermitteln, auf welcher Seite der Hindernisseite sich der Punkt des Ortsvektors befindet. Werte größer b_i entsprechen einer Lage auf der inneren Seite des Hindernisses, Werte kleiner b_i einer Lage auf der äußeren Seite. Sind die Werte gleich, liegt der Punkt auf der Seite.
Um den Abstand λ_i von P zu einer Hindernisseite zu bestimmen, wird mit $P + \lambda_i \cdot \vec{v}$ der Punkt beschrieben, der auf der Gerade der Hindernisseite liegt. Da der Punkt auf der Geraden liegen soll, gilt folgende Gleichung:

$$\vec{n_i} \cdot (P + \lambda_i \cdot \vec{v}) = b_i. \tag{12.7}$$

Es ergibt sich folgender Ausdruck für λ_i:

$$\lambda_i = \frac{b_i - \vec{n_i} \cdot P}{\vec{n_i} \cdot \vec{v}}. \tag{12.8}$$

Mit dieser Gleichung ist der Abstand zu einer beliebigen Hindernisseite beschrieben. Da v immer in Richtung Fahrbahnaußenrand zeigt, gibt ein positives λ_i an, dass sich die Hindernisseite in Richtung Fahrbahnaußenrand befindet, ein negatives λ_i entspricht einer Lage in Richtung Fahrbahninnenrand. Da ein Hindernis mehrere Seiten aufweist, und dementsprechend mehrere λ_i vorhanden sind, sind folgende zwei Fragen zu klären:

a) ob die Hindernisseite überhaupt zu einem relevanten Hindernis gehört,
b) welches der λ_i der Abstand zum Hindernis ist.

Um a) zu überprüfen, wird jedes b_i als Eintrag eines Vektors b, die Richtungsvektoren v_i als Zeile einer Matrix A aufgefasst. Mithilfe der folgenden Gleichung wird ausgedrückt, ob entlang von v überhaupt ein Punkt $P + \lambda \cdot v$ gefunden werden kann, der im Hindernis liegt:

$$A \cdot (P + \lambda \cdot v) \geq b. \tag{12.9}$$

Nur, wenn diese Gleichung lösbar ist und λ innerhalb der Fahrbahngrenzen liegt, ist das Hindernis zu berücksichtigen.

Für die Beantwortung von b) wird der Lösungsraum gültiger λ in Gleichung 12.9 näher betrachtet. Die untere und oberste Grenze sind die gesuchten Abstände von P zu den Hindernisseiten.

Zur Berücksichtigung von Hindernissen wird die Methode zur Berechnung von Abständen zu Hindernissen in `PathplanningIterativSplines` (siehe Abschnitt 11.3.5) in jedem Iterationsschritt angewendet. Für jede Stützstelle wird der Abstand zu Hindernissen und dem Fahrbahnrand berechnet und so das Intervall für den befahrbaren Bereich bestimmt. Dieses Intervall wird dem Krümmungsoptimierungsproblem als Constraint übergeben. Eine beispielhafte Trajektorie ist in Abbildung 12.8 zu sehen. Damit ist Anforderung SW2.1.9 erfüllt.

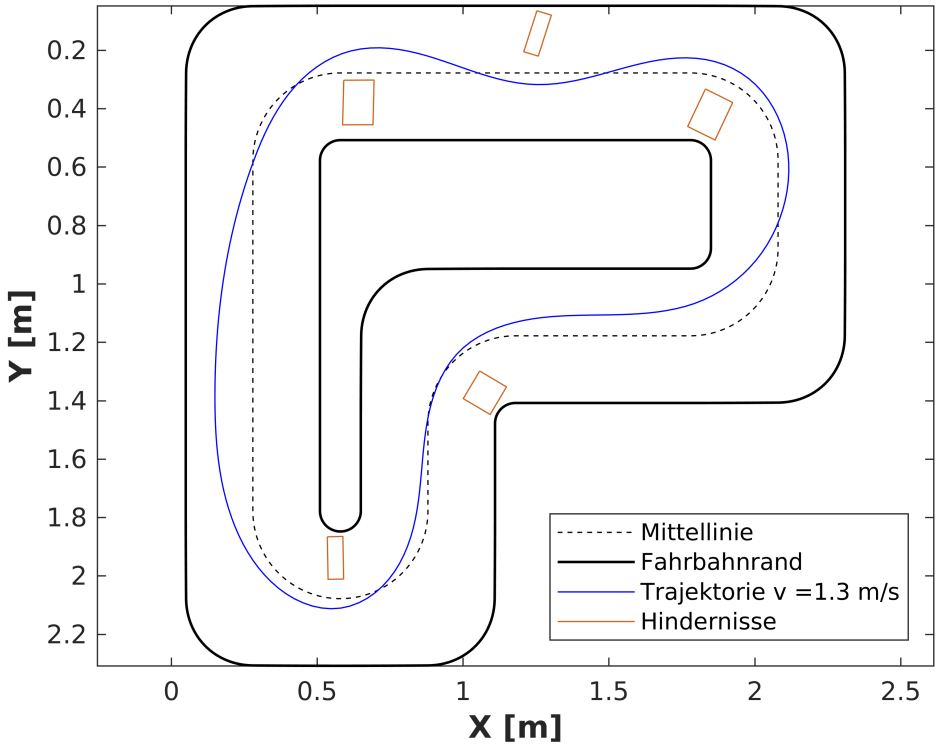
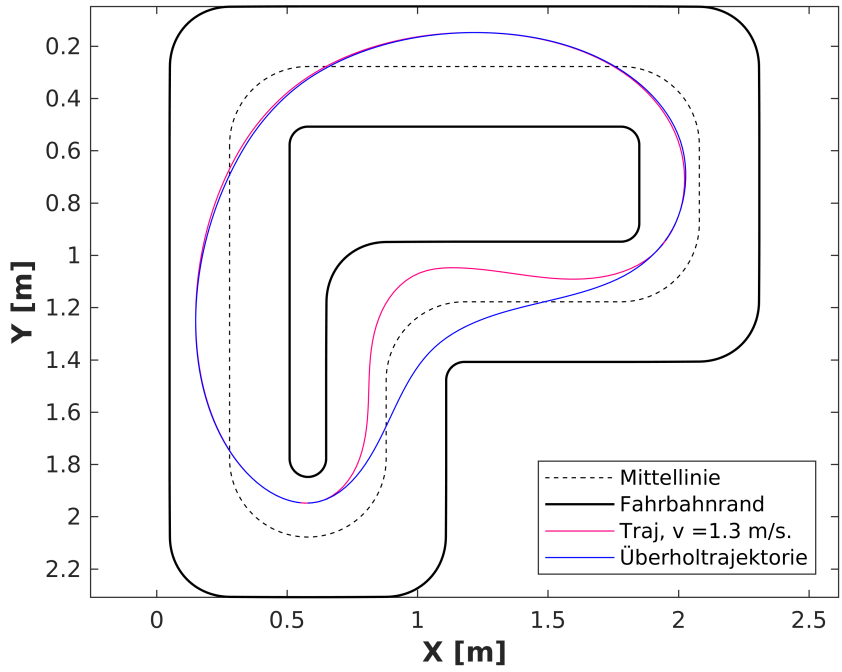


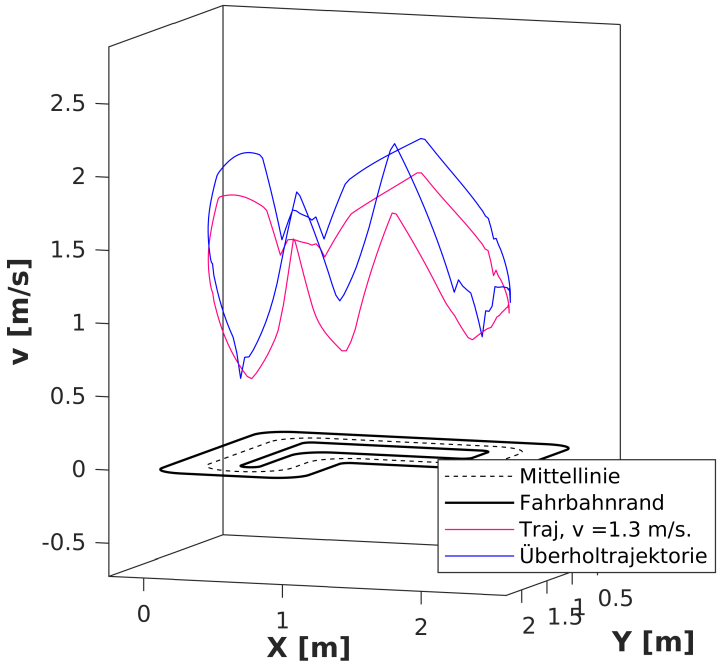
Abbildung 12.8.: Geplante Trajektorie um Hindernisse

Berechnung von Überholtrajektorien

Im Überholszenario fahren die Fahrzeuge im Normalfall eine modifizierte Trajektorie, auf der ein Überholmanöver möglich ist. (Weitere Informationen zur Überholstrategie sind in Abschnitt 12.3.13 nachzulesen). Diese Trajektorie entsteht durch Umplanen der sonst verwendeten krümmungsoptimierten Trajektorie zwischen zwei Knotenpunkten auf der Rennstrecke. Für die beiden Knotenpunkte wird der nächstgelegene Punkt auf der Trajektorie berechnet und deren Position gespeichert. Die alten Positionen werden als Constraint in der Umplanung integriert, sodass die neue Trajektorie an den Knotenpunkten der alten entspricht.
Zwischen den Knotenpunkten soll die neue Trajektorie einen längeren Weg aufweisen. Hierzu platziert der Algorithmus in der Mitte zwischen den Knotenpunkten ein virtuelles Hindernis. Dieses liegt zwischen der alten Trajektorie und der Fahrbahnseite, die einen größeren Abstand zur Trajektorie aufweist. Um dieses Hindernis wird eine neue längere Trajektorie geplant.
Weiterhin wird das Geschwindigkeitsprofil auf die gewünschte Mindestdurchschnittsgeschwindigkeit angepasst, sodass jedes Fahrzeug mit seiner eigenen Geschwindigkeit fahren kann, was Überholvorgänge erst möglich macht. Da ein Fahrzeug das Andere auf der krümmungsoptimierten Trajektorie überholen soll, muss es zwischen den Knotenpunkten seine Geschwindigkeit erhöhen. Daher wird das Geschwindigkeitsprofil auf der krümmungsoptimierten Trajektorie neu bestimmt, wobei keine nachträgliche Anpassung auf die Durchschnittsgeschwindigkeit erfolgt. Damit die Geschwindigkeit an den Knotenpunkten jedoch überein stimmt, wird diese im Optimierungsproblem zur Geschwindigkeitsberechnung an den Knotenstellen als Constraint eingeschränkt. Da die Trajektorie neu geplant ist, kann Anforderung SW2.1.10 erfüllt werden. Die Trajektorien für die Fahrverhalten „Überholen“ und „Überholt werden“ sind in Abbildung 12.9 - 12.10 beispielhaft dargestellt.

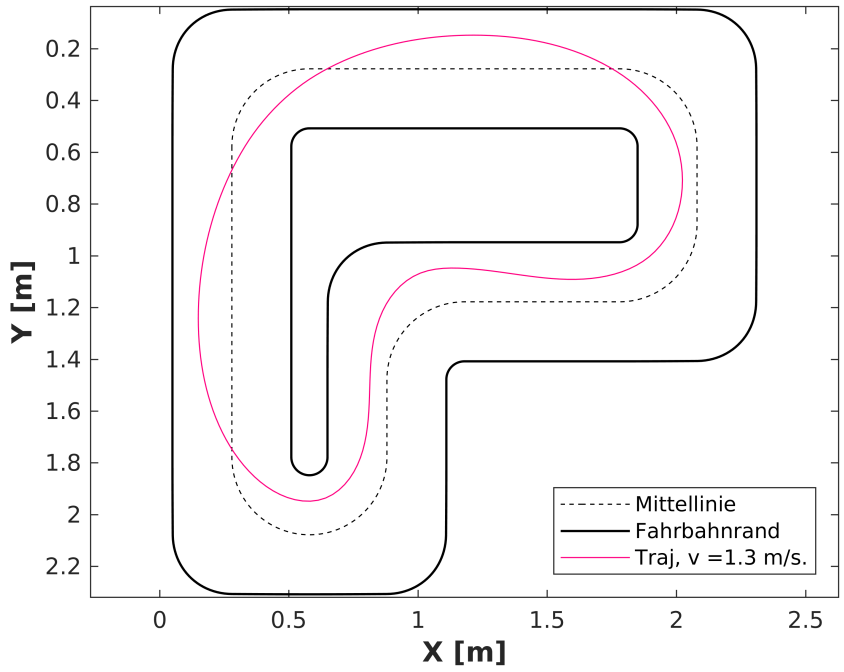


(a) Trajektorien für das Fahrverhalten „Überholen“.

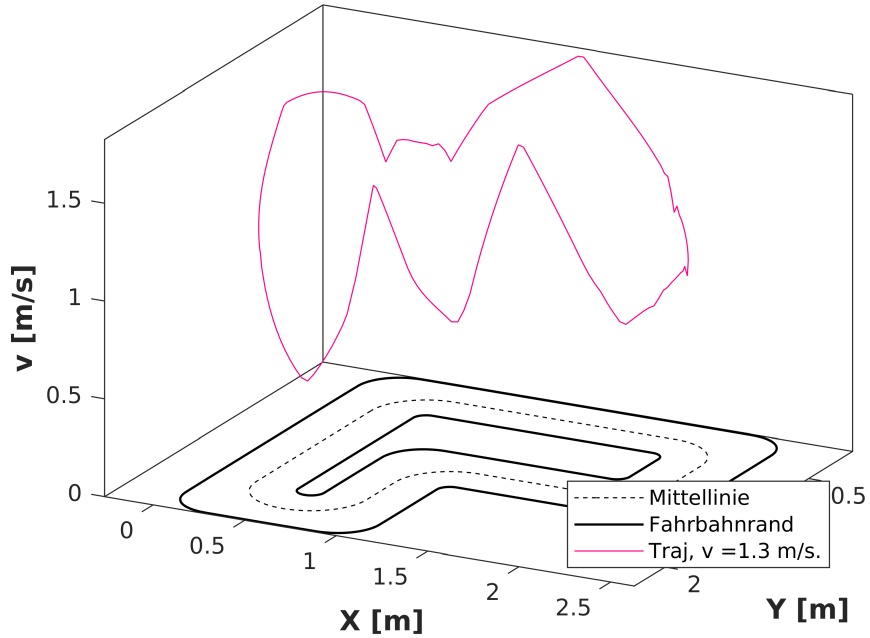


(b) Geschwindigkeitsprofil für das Fahrverhalten „Überholen“.

Abbildung 12.9.: Darstellung der Trajektorien mit ihrem Geschwindigkeitsprofil für das Fahrverhalten „Überholen“.



(a) Berechnete Trajektorien für das Fahrverhalten „Überholt werden“



(b) Geschwindigkeitsprofil der berechneten Trajektorie.

Abbildung 12.10.: Darstellung der Trajektorie mit ihrem Geschwindigkeitsprofil für das Fahrverhalten „Überholt werden“.

Verarbeitung von Matlab-Fehlern

Für den Fall, dass während der Ausführung von Matlab-Code Fehler auftreten, müssen diese an das Fehlermanagement weitergeleitet werden. Dazu wird in der Matlabsoftware gegebenenfalls ein Matlabfehler geworfen, welcher neben einem Informationstext auch Fehlergruppe und einen Fehlercode enthält. Letztere entsprechen den internen Fehlercodes der Control-Unit (siehe Abschnitt 12.3.8) und sind der Fehlercodetabelle (siehe Anhang C.2) zu entnehmen. Der Matlabfehler wird in einem try-catch Block abgefangen, der Fehlergruppe und -code ausliest und in einen Vektor schreibt. Der Vektor wird im Hauptprogramm der Vorplanung aus der Matlab-Engine gelesen, sodass eine Weiterleitung von Fehlergruppe und Fehlercode an das Fehlermanagement erfolgen kann. Unbekannte Matlabfehler, die keine eigenen Gruppen- und Codezuweisung besitzen, sind unter einer gesonderten Fehlergruppe zusammengefasst. Um den Fehler dennoch analysieren zu können, speichert die Matlab-Engine den Stacktrace samt Fehlermeldungen im Loggingpath, der durch Vorplanungsparameter einstellbar ist.

12.3.2. Kollisionserkennung

Zur Erkennung von Kollisionen ist ein Kollisionserkennungsmodul realisiert. Die Kollisionserkennung ist Teil des Sicherheitskonzepts, welche jegliche Arten von Kollisionen auf der Rennstrecke erkennen muss (siehe Sicherheitsanforderung SA9) Es wird zwischen drei verschiedenen Arten von Kollisionen unterschieden: Der Kollision zwischen einem Fahrzeug und einem Hindernis, der Kollision zwischen zwei Fahrzeugen und der Kollision zwischen einem Fahrzeug und dem Fahrbahnrand. Die Erkennung aller drei Arten von Kollisionen ist im Modellierungstool SCADE entwickelt. Das SCADE-Modell für die Kollisionserkennung ist in zwei RootNodes eingeteilt, eine zur Bandenkollisionserkennung und eine zur Fahrzeug- und Hinderniskollisionserkennung. Die RootNodes bilden die Schnittstelle vom SCADE-Modell zur restlichen C++-Implementierung der Control-Unit, welche als einen Wrapper-Code realisiert ist. Der Wrapper-Code stellt notwendige Daten zur Kollisionserkennung bereit, leitet sie an das SCADE-Modell weiter und startet diese. Zusätzlich liest sie die Ausgaben der Modelle aus und übernimmt gegebenenfalls die Versendung von Fehlercodes.
Es folgen Abbildungen aus dem SCADE-Modell, deren Funktionalität im Anschluss erläutert wird.

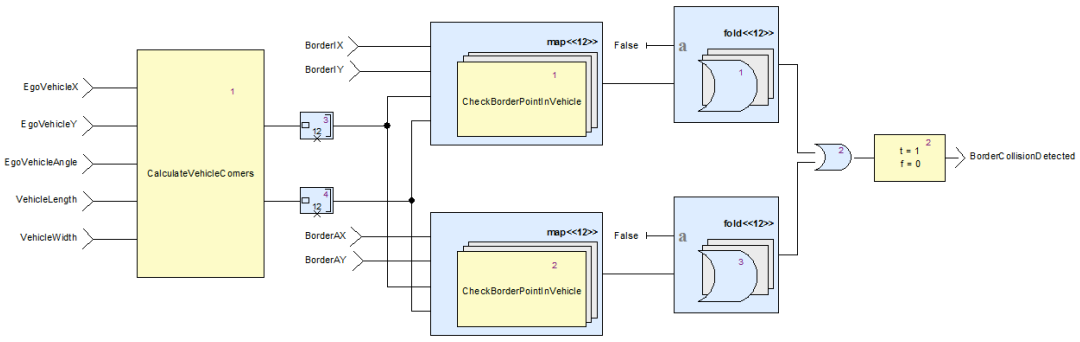


Abbildung 12.11.: SCADE Modell - Bandenkollisionserkennung.

Bandenkollisionserkennung

In Abbildung 12.11 ist das entwickelte Bandenkollisionsmodell dargestellt. Der SCADE-Operator `CalculateVehicleCorners` wird zunächst mit fünf Inputvariablen aufgerufen. Diese beinhalten die X- und Y-Koordinate des zu steuernden Fahrzeugs, dessen Ausrichtungswinkel, Breite und Länge. Aus diesen Daten werden die genauen Koordinaten der vier Eckpunkte des Fahrzeugs berechnet und anschließend als zwei Arrays ausgegeben.

Im nächsten Schritt wird überprüft, ob einer von zwölf Bandenpunkten innerhalb des zuvor berechneten Fahrzeug liegt. Die Bandenpunkte werden durch eine Funktion außerhalb von SCADE berechnet, welche in C++ implementiert ist und die statischen Umgebungsdaten ausliest. Auf ganzer Länge und Breite des Fahrzeugs sind gleichmäßig Kontrollpunkte definiert. Die C++-Funktion liefert für diese Punkte die korrespondierenden Bandenpunkte, die auf der inneren und äußeren Bande liegen und im rechten Winkel zur Mittellinie angeordnet sind. Im Modell werden diese Punkte als vier Arrays eingegeben (BorderIX, BorderIY, BorderAX, BorderAY).
Der Operator `CheckBorderPointInVehicle` wird in einem Aufruf hierzu zwölf mal ausgeführt, dabei wird kontrolliert ob jeweils einer der ermittelten Bandenpunkte im zuvor berechnetem Fahrzeug liegt. Dazu müssen die Fahrzeugpositionsdaten verzwölffacht werden. Zum Abschluss wird kontrolliert, ob alle Überprüfungen negativ ausgefallen sind, was in diesem Fall bedeutet, dass keine Kollision mit der Bande vorliegt. Im Falle, dass ein Punkt im Fahrzeug festgestellt wurde, wird dieses durch eine 1 an den Wrapper-Code des SCADE Modells weitergegeben ansonsten wird für jede Überprüfung eine 0 ausgegeben.

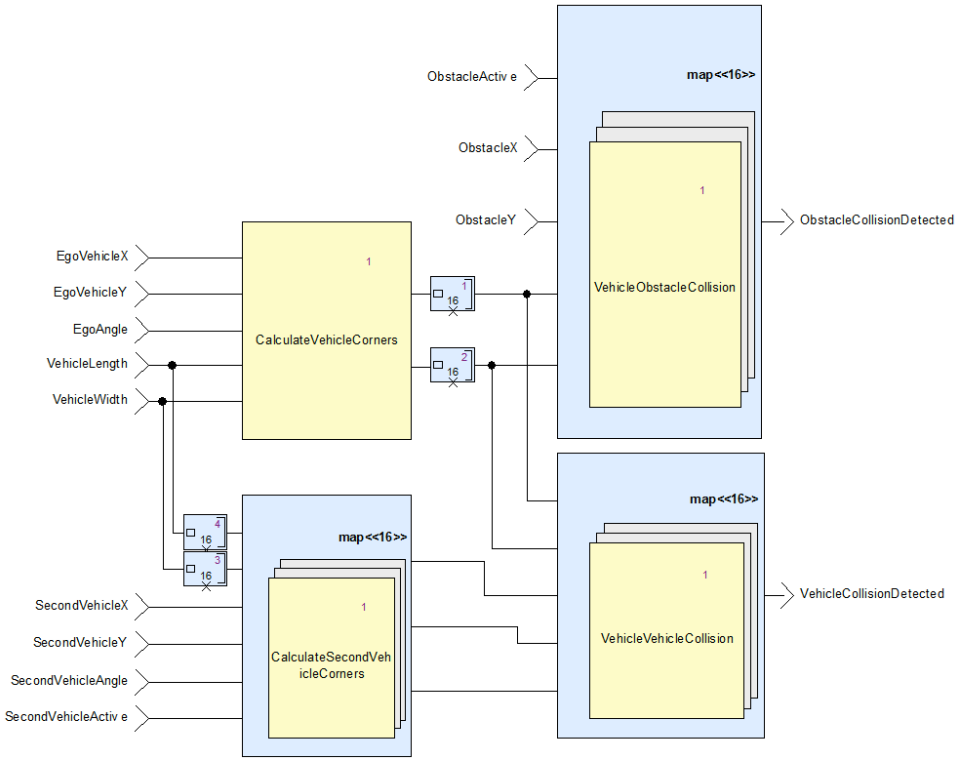


Abbildung 12.12.: SCADE Modell - Fahrzeug- und Hinderniskollisionserkennung.

Hinderniskollisionserkennung

Abbildung 12.12 zeigt das SCADE-Modell zur Erkennung von Fahrzeug- Fahrzeugkollisionen und Kollisionen mit einem Hindernis. Zunächst werden wieder die vier Eckpunkte des eigenen Fahrzeugs mittels `CalculateVehicleCorners` ermittelt. Da maximal 16 Hindernisse und maximal 16 Fahrzeuge auf der Rennstrecke vorhanden sein dürfen, werden die Eckpunkte des Fahrzeugs versechzehnfacht. In den `VehicleObstacleCollision`-Operator gehen die Eckpunkte des Fahrzeugs, sowie die Eckpunkte aller Hindernisse. Zusätzlich wird durch das ObstacleActive-Flag angegeben, ob sich an der Indexstelle der Obstacle-Arrays ein Hindernis befindet oder nicht. Falls nicht, wird dieser Schritt übersprungen und das nächste Hindernis wird überprüft. Zur Überprüfung, ob eine Hinderniskollision stattgefunden hat, werden alle Hinderniseckpunkte betrachtet. Wenn einer der Hindernispunkte sich innerhalb des im Modell berechneten Fahrzeugs befindet oder vice versa, meldet die Hinderniskollision dies an den Wrapper-Code.

Fahrzeugkollisionserkennung

Der Teil der Kollisionserkennung, der eine Kollision zwischen zwei Fahrzeugen feststellt, entspricht der Funktionalität der Hinderniskollisionserkennung, nur dass zuerst die Eckpunkte aller anderen Fahrzeuge berechnet werden müssen. Dazu empfängt der `CalculateSecondVehicleCorners`-Operator die Mittelpunkte, Ausrichtungswinkel, Länge, Breite und ein Active-Flag aller Fahrzeuge. Das Active-Flag des eigenen Fahrzeugs wird dabei vom Wrapper-Code deaktiviert, um keine Kollision mit sich selbst zu erkennen. Die Eckpunkte werden berechnet und als Array an den `VehicleVehicleCollision`-Operator übergeben. Dieser überprüft, ob einer der Eckpunkte anderer Fahrzeuge im betrachteten Fahrzeug liegt und umgekehrt.

12.3.3. Kommunikationsinterface

Das Kommunikationsinterface ist die Schnittstelle der Control-Unit mit den anderen Subsystemen. Dieses Modul stellt Funktionen zum Empfangen und Verarbeiten der Netzwerkpakete zur Verfügung. Neben diesen Funktionen stellt dieses Modul Methoden zur Überprüfung, ob ein Paket eines bestimmten Typs bereits eingetroffen ist, bereit. Die Netzwerkpakete und die Implementierung der Kommunikation ist in Abschnitt 12.6 beschrieben. Das Kommunikationsinterface ist in C++ als Thread implementiert. Somit wird gewährleistet, das ankommende Pakete in regelmäßigen Abständen verarbeitet werden. Dieser Thread wird alle 20 µs aufgerufen.

12.3.4. Konfigurationsverarbeitung

In der Konfigurationsverarbeitung werden die Konfigurationseinstellungen für die Control-Unit eingelesen und statische Umgebungsdaten für die Vorplanungsebene erstellt.
Die Konfigurationseinstellungen setzen sich aus einer Textdatei, welches Lokal auf dem Rechner liegt, auf dem die Control-Unit ausgeführt wird, und aus Konfigurationsdaten, die von der Systemsteuerungssoftware verschickt werden, zusammen. In der Textdatei befinden sich Kommentarzeilen, welche mit einem „#“ beginnen, und Einstellungen, die nach folgendem Schema aufgebaut sind:

$$Bezeichner = Wert.$$

Dabei ist *Bezeichner* ein String, über den im Code auf den *Wert* zugegriffen werden kann. Eine Beschreibung dieser Konfigurationsdatei befindet sich im Anhang (siehe Abschnitt B.2).

Die Konfigurationsdaten der Systemsteuerungssoftware beinhalten Informationen über die Zuweisung eines Fahrzeugs zu einer Control-Unit, die Mindestdurchschnittsgeschwindigkeit und das Fahrverhalten der Fahrzeuge. Die Konfigurationsverarbeitung erhält die Konfigurationsdaten über das Kommunikationsinterface (siehe Abschnitt 12.3.3) von der Systemsteuerungssoftware und speichert diese.
Die statischen Umgebungsdaten beschreiben den Aufbau der Rennstrecke. Hier wird ausgehend von einem Startpunkt nahe der Start-/Ziellinie der Verlauf der Rennstrecke beschrieben. Hierzu werden Informationen über Geraden- und Kurvensegmente zeilenweise in ein Array geschrieben.
Abschließend wird die Position der Eckpunkte der Hindernisse in die letzten Zeilen des Arrays geschrieben und dieses Array als statische Umgebungsdaten dem System zur Verfügung gestellt.

12.3.5. Statusmanagement

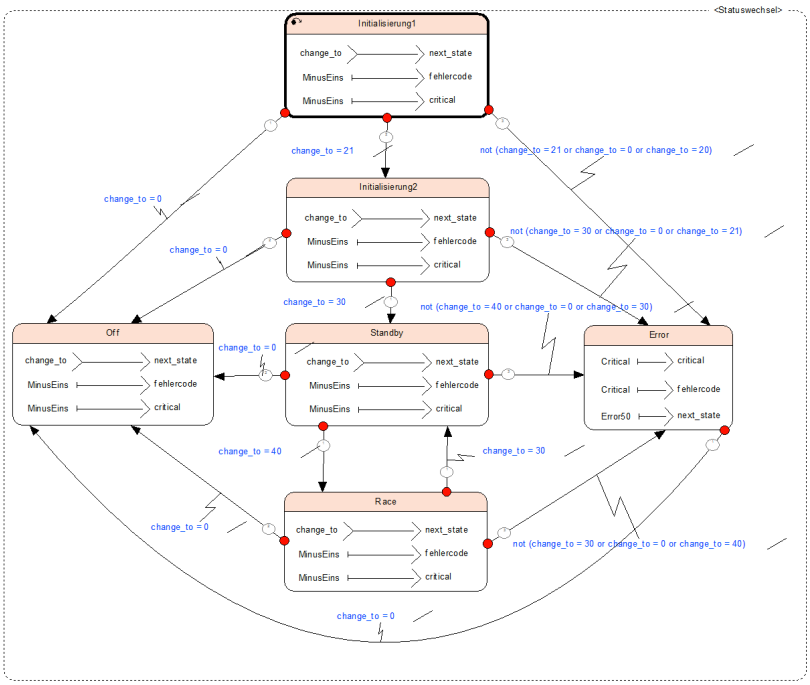


Abbildung 12.13.: Zustandsautomat Statusmanagement.

Das Statusmanagement realisiert das Verhalten des Zustandsautomaten mittels eines SCADE-Modells. Die Statuswechselbefehle werden von der Systemsteuerungssoftware via eines Kontrolldatenpakets versendet. Entspricht dieser Statuswechselbefehl nicht dem Zustandsautomaten, so geht das System in einen Fehlerzustand über und versendet einen sicherheitskritischen Fehlercode.

Das Statusmanagement ist als Thread realisiert und fragt in regelmäßigen Abständen nach dem aktuellsten Kontrolldatenpaket. Eine Beschreibung der einzelnen Zustände befindet sich in Abschnitt 9.1.3.
Abbildung 12.13 zeigt den modellierten Zustandsautomaten. Der Aufbau und die Funktionalität ist mit dem Zustandsautomaten in 9.3 identisch. Initial befindet sich der Automat im Zustand „Initialisierung 1“. Mit dem `change_to`-Befehl wechselt der Automat in „Initialisierung 2“, „Standby“, „Race“ und von „Race“ wieder zurück in „Standby“. In jedem Zustand ist es möglich, in den „Aus“-Zustand zu wechseln. Ist der Zustandswechsel erfolgreich, sendet die Control-Unit mit dem `next_state`-Befehl den neuen Zustand an alle anderen relevanten Komponenten und bestätigt somit auch der Systemsteuerungssoftware den Zustandswechsel. Empfängt das Statusmanagement zu einem Zeitpunkt einen illegalen `change_to`-Befehl, so wird ein Wechsel in den „Error“-Zustand eingeleitet und ein sicherheitskritischer Fehlercode gesendet.

12.3.6. Plausibilitätskontrolle

Die Plausibilitätskontrolle überprüft Konfigurationsdaten, statische Umgebungsdaten und Fahrzeugdaten gemäß der Schnittstellenbeschreibung in Abschnitt 9.2.3, sowie Konfigurationsparameter. Da nicht alle Daten mehrfach erzeugt werden, unterscheidet die Plausibilitätskontrolle zwischen einmaliger und regelmäßiger Prüfung. Während sie Fahrzeugdaten im Rennzustand jede Millisekunde auf Schnittstellenkonformität in einer Methode als Thread regelmäßig überprüft, werden die folgenden Daten durch einen einmaligen Methodenaufruf kontrolliert:

Konfigurationsparameter Die eingelesenen Parameter werden überprüft, nachdem die Konfigurationsverarbeitung die Konfigurationsdaten eingelesen hat. Es wird überprüft, ob Zeichenketten nicht leer, Gleitkommazahlen kein NaN und Ganzzahlen ungleich 0 sind.

Konfigurationsdaten Sobald im zweiten Initialisierungszustand Konfigurationsdaten empfangen wurden, überprüft die Methode der Plausibilitätskontrolle die zugewiesenen Konfigurationsdaten auf Schnittstellenkonformität.

Statische Umgebungsdaten Nachdem im zweiten Initialisierungszustand statische Umgebungsdaten erzeugt wurden, erfolgt eine Prüfung durch die Plausibilitätskontrolle. Zunächst wird in der Infozeile geprüft, ob diese gültige aufsteigende Zeilennummern verweisen, anschließend werden die Zeilen zu Rennstrecken- und

Hindernisdaten auf Schnittstellenkonformität geprüft. Da hier auch die empfangenen Hindernisdaten eingehen und hinsichtlich der Schnittstelle kontrolliert werden, müssen diese nicht separat überprüft werden.

12.3.7. Watchdogs

Die Watchdogs sind in der Control-Unit für sicherheitskritische Komponenten vorhanden. Somit wird sichergestellt, dass ein Ausfallen von Modulen erkannt wird (siehe Sicherheitsanforderung SA7). Nicht jedes Modul hat einen expliziten Watchdog, da die Funktionalität eines Watchdogs schon von anderen Komponenten erfüllt wird. Die folgenden Module verfügen über einen expliziten Watchdog:

- Vorplanung
- Kollisionserkennung
- Statusmanagement
- Plausibilitätskontrolle
- Fehlermanagement
- Realzeitüberprüfung
- Systemsteuerungssoftware

Diese sind in SCADE implementiert. Die Rootnode dieses Modells wird in einem Thread alle 1000 µs aufgerufen. Jede Komponente sendet in regelmäßigen Abständen einen Heartbeat an das SCADE-Modell. Ein beispielhafter Watchdog ist in Abbildung 12.14 zusehen.
In diesem Modell wird in jedem Aufruf 1000 µs von den individuellen Zeitbegrenzungen der Module abgezogen. Sobald ein Heartbeat und dadurch das Zeichen, dass dieses Modul noch am Arbeiten ist, eintrifft, wird die interne Zeitvariable dieser Komponente wieder zurückgesetzt. Der Watchdog kann für einzelne Komponenten an oder ausgeschaltet werden. Dies ermöglicht ein besseres Testen.
Das Kommunikationsinterface und die Renndatenüberprüfung verfügt über einen impliziten Watchdog in der Realzeitüberprüfung (siehe Abschnitt 12.3.9). Die CarControl (siehe Abschnitt 12.3.12) verfügt über einen Softwarewatchdog auf dem Entwicklungsboard. Dieser ist auch der implizite Watchdog der SerialControl und des Regelungsmoduls. Der CarControl-Watchdog wird immer dann zurückgesetzt, wenn ein berechneter Regelungswert an das Fahrzeug gesendet wird.
Abbildung 12.14 zeigt den Watchdog für das Fehlermanagement. Die einzelnen Watchdogs für die unterschiedlichen Komponenten unterscheiden sich nicht in der Funktionalität sondern nur in dem ErrorCode, der an den Wrapper-Code des SCADE-

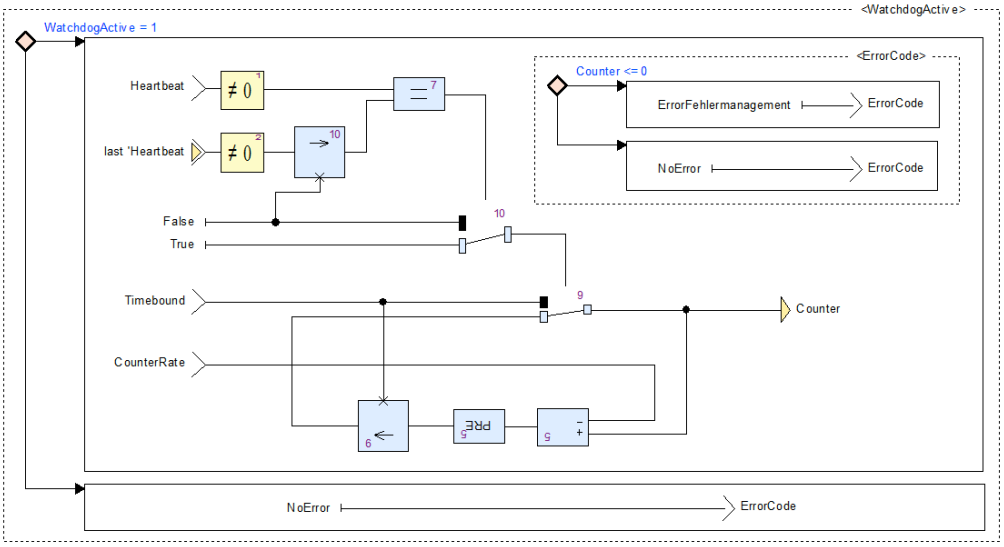


Abbildung 12.14.: SCADE Modell - Beispielhafte Darstellung des Watchdogs für das Fehlermanagement.

Modells weiter gegeben wird. Zunächst empfängt jeder Watchdog einen Heartbeat, der 1 oder 0 beinhaltet, einen Timebound in µs und eine CounterRate, die der Aufrufrate der Watchdog-RootNode entspricht. Ist der Watchdog aktiv, so subtrahiert er bei jedem Aufruf die CounterRate vom Timebound. Wechselt der Heartbeat seinen Wert und ist dementsprechend nicht gleich dem letzten Heartbeat, wird der Counter wieder auf den Ursprung des Timebound zurück gesetzt und der Watchdog beginnt erneut herunter zu zählen. Diese Funktionalität entspricht der eines Time-Out-Watchdogs. Sollte die lokale Variable Counter, in der der aktuelle Timebound abgespeichert wird, kleiner oder gleich Null sein, meldet der Watchdog einen Fehler an den Wrapper-Code des SCADE-Modells, der den Fehler mit der entsprechenden Modulnummer an das Fehlermanagement weiterleitet.

12.3.8. Fehlermanagement

Das Fehlermanagement sammelt die Fehler von allen Komponenten und leitet diese an das Kommunikationsinterface weiter. Um Lese- und Schreibzugriffe voneinander zu trennen, wurden hier Ringpuffer verwendet. Jede Komponente der Control-Unit verfügt über zwei Puffer, einen für kritische Fehler, welcher 5 Speicherplätze groß ist, und einen für nichtkritische Fehler, der 10 Plätze groß ist. Zudem werden externe Fehlercodes (siehe Anhang C.1) für die interne Verarbeitung umgewandelt. Damit

andere Subsysteme alle auftretenden Fehler mitbekommen, werden auch interne Fehlercodes in externe umgewandelt und als solche über die Netzwerkschnittstelle versendet (siehe Sicherheitsanforderung SA14). Sobald ein kritischer Fehler im internen oder externen Speicher liegt, wird über die SerialControl (siehe Abschnitt 12.3.11) ein Not-Halt-Befehl ausgelöst. Dieser Not-Halt-Befehl wird über eine Variable an die SerialControl übergeben. Das Fehlermanagement überprüft in regelmäßigen Abständen, ob ein aktueller Fehler in einem der Modulfehlerpuffer liegt. Diese Überprüfung geschieht in einer festen Reihenfolge. Zuerst wird überprüft, ob ein externer Fehler anliegt, anschließend werden die Module in folgender Reihenfolge abgearbeitet:

1. Realzeitüberprüfung
2. Watchdogs
3. Kollisionserkennung
4. SerialControl
5. Plausibilitätskontrolle
6. Statusmanagement
7. Konfigurationsverarbeitung
8. Renndatenberechnung
9. Vorplanung
10. Regelungsmodul

Die Regelmäßigkeit wird über einen Thread gelöst, welcher alle 500 µs aufgerufen wird. Zudem werden so die Sicherheitsanforderungen SA7 und SA13 abgedeckt.

12.3.9. Realzeitüberprüfung

In der Realzeitüberprüfung werden die Realzeitconstraints an die Control-Unit überprüft. Zum Einen wird überprüft, ob alle 20 ms ein neues Bild von der Positionsbestimmung empfangen wurde. Dazu wird in regelmäßigen Abständen das neueste Paket vom Kommunikationsinterface abgefragt. Diese regelmäßige Überprüfung sorgt dafür, dass Sicherheitsanforderung SA2 erfüllt wird. Das Intervall von 20 ms erlaubt den Verlust eines Bildes, da die Positionsbestimmung pro Bild nicht mehr als 10 ms zur Berechnung braucht. Diese Toleranz sorgt für die Erfüllung der Sicherheitsanforderung SA13 bzgl. der Control-Unit.

Zum Anderen überprüft die Realzeitüberprüfung, ob das Regelungsmodul alle 10 ms neue Regelungswerte berechnet. Die Regelungswerte werden mit einem Timestamp in einen Puffer geschrieben. Anhand dieses Puffers kann die Realzeitüberprüfung die Einhaltung dieses Zeitconstraints überprüfen. Diese Überprüfung stellt sicher, dass die Sicherheitsanforderung SA8 erfüllt wird.
Als letzten Punkt berechnet die Realzeitüberprüfung die Zeit vom Empfang eines Bildes der Kamera bis zum Aussenden der Regelungswerte an das Fahrzeug über die CarControl. Zu dieser Berechnung wird der Timestamp des Fahrzeugdatenpakets, der den Aufnahmezeitpunkt des Bildes markiert, und der Zeitpunkt, an dem die SerialControl die Regelungswerte an das Fahrzeug verschickt, mit herangezogen.

12.3.10. Renndatenberechnung

Eine Renndatenberechnung wurde zu diesem Zeitpunkt nicht realisiert. Ursprünglich sollte diese Auskunft über die aktuelle Geschwindigkeit, die Rundendurchschnittsgeschwindigkeit und die Rundenanzahl geben. Aus zweierlei Gründen hat die Projektgruppe sich gegen die Implementierung dieses Moduls entschieden. Zum einen wird innerhalb der Control-Unit weder die Rundendurchschnittsgeschwindigkeit, noch die Rundenanzahl benötigt. Zum Anderen wurden höhere Prioritäten auf andere Module gesetzt.
Die aktuelle Geschwindigkeit des Fahrzeugs wird nur für die MPC (siehe Abschnitt 12.3.13) benötigt, diese berechnet die Geschwindigkeit selbst. Die Rundendurchschnittsgeschwindigkeit und Rundenanzahl werden im Interface des Systemsteuerungssoftware angezeigt. Um diese Daten von der Control-Unit zur Systemsteuerungssoftware zu bekommen, wäre eine Anpassung des Netzwerkpakets notwendig gewesen. Diese Änderung hat die Projektgruppe als nicht notwendig betrachtet. Stattdessen werden die Berechnungen in der Systemsteuerungssoftware ausgeführt (siehe 12.5).
Für zukünftige Renndatenberechnungen in SCADE wurde allerdings schon eine Schnittstelle und RootNode erstellt. Diese kann, sofern gewünscht, mit Funktionalität gefüllt werden.

12.3.11. Serial Control

Die SerialControl ist für die Kommunikation mit der CarControl zuständig. Hier werden die Regelungsdaten zu den Stellgrößen umgerechnet, welche über eine serielle Schnittstelle an die CarControl übertragen werden. Die Regelungswerte werden, um auch hier gleichzeitige Lese- und Schreibzugriffe zu verhindern, aus einem Rege-

lungsdatenpuffer ausgelesen. Die Regelungsdaten bestehen aus dem Lenkwinkel δ, der Beschleunigung a und der Geschwindigkeit v. Die Umrechnung von Beschleunigung zum Dutycycle D erfolgt mittels der in Abschnitt 10.4 evaluierten Fahrzeugparameter, welche in folgende Formel eingesetzt werden.

$$D = \frac{a + cr2 \cdot v^2 + cr0 + c1 \cdot c2 \cdot v^2 \cdot \delta^2}{cm1 - cm2 \cdot v}. \tag{12.10}$$

Anschließend wird der Dutycycle mittels eines linearen Mappings auf die Throttlewerte gemappt. Dieses Mapping hängt zudem davon ab, in welchem Zustand (Vorwärtsfahrt oder Bremsen) sich der Motor befindet. Anschließend wird der gewünschte Lenkwinkel, ebenfalls über ein lineares Mapping, in die entsprechende Steering-Stellgröße übersetzt. Die Ermittlung dieser Mapping-Funktion wird in Abschnitt 10.1 beschrieben.
Nach der Umrechnung findet hier noch eine Begrenzung der berechneten Werte statt, sodass nur valide Daten über die serielle Schnittstelle an die CarControl gesendet werden (siehe Sicherheitsanforderung SA10).
Neben dem Versenden der Stellgrößen ist durch dieses Modul die Steuerung der CarControl möglich. Das Modul SerialControl empfängt das Not-Halt-Signal von dem Fehlermanagement (siehe Abschnitt 12.3.8) und leitet dieses direkt an die CarControl weiter. Zudem reagiert die SerialControl auf einen Ausfall der CarControl. Dies beinhaltet das Drücken des NotAus- und Reset-Buttons auf der CarControl. Das Senden der Steuerbefehle und das Empfangen von Feedbacknachrichten wird in je einem Thread durchgeführt. Sobald die Verbindung zur CarControl gestört ist wird von diesem Modul ein sicherheitskritischer Fehler ausgelöst.

12.3.12. CarControl

Die Software der CarControl wurde an wenigen Stellen angepasst. So wurde ein Watchdog auf das Entwicklungsboard gebracht, welcher überprüft, ob alle 200 ms ein Regelungswert von der SerialControl empfangen wurde.

Zudem wurde die Kommunikation mit der SerialControl verbessert. So wurde das Versenden von Antworten auf Statusbefehle angepasst. Wenn die CarControl einen von diesen Befehlen bekommt, antwortet diese nun nur noch einmal statt fünfmal. Somit kann eine genaue Zuordnung von Befehl zu Antwort vorgenommen werden. Über diese Zuordnung lässt sich überprüfen, ob ein Befehl empfangen und ausgeführt wurde.

12.3.13. Regelungsmodul

Das Regelungsmodul beinhaltet insgesamt drei verschiedene Einzelmodule. Sie werden neben den anderen Modulen der Control-Unit als Thread aufgerufen. Die Einzelmodule sind auf die entsprechenden Fahrverhalten, „Passives Fahren“, „Überholen“ und „Überholt werden“, abgestimmt. In diesem Abschnitt wird zunächst ein Regelungsablauf erläutert, um dann die einzelnen Fahrverhalten in ihrer Umsetzung vorzustellen. Im Laufe der Entwicklung ist aufgefallen, dass eine erhöhte Latenz zwischen den Bildern und dem Senden der Steuersignale auftritt. Auf die empirische Untersuchung dieser Latenz und der Evaluation weiterer Parameter für die Regelung wird im letzten Abschnitt eingegangen.

Regelungsablauf mit Model Predictive Control

In der Regelung wird die Model Predictive Control (MPC), wie sie in Kapitel 11.3.6 beschrieben ist, verwendet. Die MPC wurde dazu zunächst mithilfe der Bibliothek „Armadillo“ [SC16] von Matlab in C++ übersetzt. Der Ablauf der Regelung ist wie folgt:
Zu Beginn wird die Pose des Fahrzeugs ausgelesen. Durch diese Position wird der nächstgelegene Punkt auf der Trajektorie ermittelt. Anhand des Punktes auf der Trajektorie und der tatsächlichen Pose des Fahrzeugs wird der Fehler bestimmt, mit dem das Fahrzeug von der Referenztrajektorie abweicht. Dieser Fehler wird daraufhin normalisiert. Die folgenden Berechnungen der MPC zielen darauf ab, unter Berücksichtigung von Fahrzeugconstraints, wie maximaler und minimaler Lenkwinkel, Beschleunigung, und Fahrbahnconstraints, diesen Fehler zu minimieren. Das Problem wird dabei zunächst aufgebaut und dann durch den Solver „gurobi“ [Gur] gelöst. Sollten die Constraints nicht einzuhalten sein, wird versucht, das Problem durch Relaxion zu lösen. In diesem Fall versucht „gurobi“ das Problem ohne die Fahrbahnconstraints zu lösen.
Durch Gewichtung der Fehler in Bezug auf die x-/y-Koordinate, die Ausrichtung ψ und die Geschwindigkeit v des Fahrzeuges kann bestimmt werden, wie stark die MPC von den Werten abweichen darf. Dabei beschreiben höhere Werte eine erhöhte Genauigkeit. Durch Kosten für Lenkung und Beschleunigung können die Regelungswerte weiter angepasst werden. So wird durch eine Erhöhung der Kosten für die Lenkung ein zu schnelles und starkes Lenken des Fahrzeuges verhindert. Die Gewichte wurden durch empirische Tests ermittelt und sind im Config-File hinterlegt. Sie werden dort als w_v,w_x, w_y, w_ψ für die Gewichtungen und ru_1 und ru_2 für die Kosten angegeben. Im Referenzpaper zur MPC [Wun] sind die Gewichte als q_w, q_v, q_x, q_y angegeben.

Sollte das Regelungsmodul eine Geschwindigkeit mit dem Wert 0 m/s feststellen, so wird diese Geschwindigkeit auf einen minimalen Wert von 0.001 gesetzt. Dies hat den Grund, da in der Normalisierung des Fehlers sonst durch 0 geteilt werden und dies zu nicht verwendbaren Werten führen würde.

Es wurde festgestellt, dass das System eine gewisse Latenz besitzt. Als Latenz wird die Zeit bezeichnet, die vom Aussenden bis zur Umsetzung der Regelungsgröße vergeht. Das bedeutet, dass die Regelungswerte nicht zu der Position des Fahrzeugs passen. Der Wert für die Latenz ist im Config-File hinterlegt. Durch empirisches Testen wurde ein Latenzwert von 4 festgelegt. Das bedeutet jedoch, dass das System bis zum Aussenden der Regelungswerte bereits 4 Bilder empfangen hat, was etwa 26.8 ms Versatz entspricht. Dieser Versatz entspricht etwa der Verzögerung des Gesamtsystems von der Bildaufnahme bis zum Empfang des Signal am Fahrzeug. Durch eine Prognosefunktion für das Fahrzeug wird dies allerdings behoben. Diese errechnet anhand der Pose des Fahrzeugs, der Geschwindigkeit, der Trajektorie und eines Prognosevektors, die Position des Fahrzeugs, an dem es sich befindet, wenn die Regelungsgrößen umgesetzt sind. Der Prognosevektor besitzt zwei Spalten und eine Anzahl Reihen, die der Größe der Latenz entspricht. In ihm sind die Beschleunigung und der Lenkwinkel für die Zustände, in denen sich das Fahrzeug bis zur Umsetzung der Regelungsgrößen befindet. Durch die Funktionen

$$u_1 = \frac{v \cdot \delta}{L} - v_n \cdot k \tag{12.11}$$

und

$$u_2 = \frac{a - a_n}{v_n} \tag{12.12}$$

werden die Steuerungsgrößen berechnet. v beschreibt dabei die momentane Geschwindigkeit, L den Radstand, δ den Lenkwinkel, k die Krümmung, a die momentane Beschleunigung, a_n die Nominalbeschleunigung und v_n die Nominalgeschwindigkeit, die das Fahrzeug an diesem Punkt besitzen soll.
Durch die Formel

$$x_{k+1} = A_k \cdot x_k + B_k \cdot u_k \tag{12.13}$$

wird der nächste Zustand des Fahrzeugs berechnet, in dem es sich mit diesen Steuergrößen befinden würde. x_{k+1} beschreibt den nächsten Zustand des Fahrzeugs durch x, y, ψ und v. Bei A_k und B_k handelt es sich um feste Matrizen. x_k beschreibt den momentanen Zustand und u_k beinhaltet die Steuergrößen u_1 und u_2. Diese Berechnung wird der Größe des Prognosevektors entsprechend oft durchgeführt. Durch eine Denormalisierung wird dann die tatsächliche prognostizierte Position des Fahrzeugs

bestimmt und in die MPC übergeben. Dadurch werden Steuersignale für zukünftige Positionen berechnet. Diese Signale werden an das Fahrzeug gesendet und erreichen dies, wenn das Fahrzeug sich tatsächlich an dieser Position befindet.
Die MPC besitzt eine Schrittweite T von 0.0067 s. Dieser Wert wurde gewählt, da die Kamera mit einer Frequenz von 150 Hz Bilder aufnimmt. Dies bedeutet, dass ein Bild alle $\frac{1}{150}$ s aufgenommen wird. Damit die MPC auf die Bilderfassungsrate getaktet ist, wurde die Schrittweite auf $\frac{1}{150}$ gesetzt, was 0.0067 s entspricht. Um die Realzeitanforderungen bestmöglich einzuhalten, wurde die Anzahl der Schritte N innerhalb der MPC auf 18 gesetzt. Dieser Wert wurde empirisch getestet. Die MPC besitzt somit einen Gesamthorizont von 0.121 s pro Aufruf.

Fahrverhalten 1: Passives Fahren

Für die Umsetzung des Fahrverhaltens „passives Fahren“ wird das Geschwindigkeitsprofil der Nominaltrajektorie in jedem Aufruf des Regelungsmoduls modifiziert. Die Geschwindigkeits- und Beschleunigungswerte werden ausgehend von dem vorausfahrenden Fahrzeug in Richtung des nachfolgenden Fahrzeugs durch ein Bremsprofil ersetzt. Die Zielgeschwindigkeit des Bremsprofils entspricht dabei der gegenwärtigen Geschwindigkeit des vorausfahrenden Fahrzeugs. Die Stelle, an der das Fahrzeug zu bremsen beginnt, ist so gewählt, dass das Fahrzeug mit der maximal möglichen Bremsbeschleunigung spät möglichst abgebremst werden kann. Der Algorithmus zu Berechnung ist wie folgt:

Aus den aktuellen Positionsdaten des voraus fahrenden Fahrzeugs wird der nächstgelegene Punkt auf der Trajektorie ermittelt. In die zugehörige Stützstelle der Trajektorie (siehe „Planungsdaten“ in Abschnitt 9.2.3) wird seine Geschwindigkeit v_{soll} mit Beschleunigung $0\,\mathrm{m/s^2}$ eingetragen, sodass das folgende Fahrzeug dort die Geschwindigkeit v_{soll} aufweist und weder bremst noch beschleunigt. Für einen Sicherheitsabstand wird v_{soll} ebenfalls in mehrere vorherige Stützstellen mit Beschleunigung $0\,\mathrm{m/s^2}$ eingetragen. Die Anzahl der Einträge ist in den Konfigurationsparametern einstellbar. Vor Beginn dieser geschwindigkeitskonstanten Stützstellen beginnt das eigentliche Bremsprofil. Mithilfe der maximalen Bremsbeschleunigung a_{min} und der Segmentlänge der Trajektorie s wird durch die Gleichung $v_{soll}^2 = v_{max}^2 + a_{min} \cdot s$ die Geschwindigkeit v_{max} berechnet, von der das Fahrzeug von der vorherigen Stützstelle der Trajektorie auf v_{soll} abbremsen kann. Ist v_{max} kleiner als die ursprüngliche Geschwindigkeit v_i an der vorherigen Stützstelle im Geschwindigkeitsprofil, so wird v_{max} mit der Beschleunigung a_{min} an diese Stützstelle geschrieben. Durch

$v_{max}^2 = v'^2_{max} + a_{min} \cdot s$ berechnen sich iterativ die maximal möglichen Geschwindigkeiten für die weiteren vorherigen Stützstellen. Für jede Stützstelle wird verglichen, ob v'_{max} kleiner als v_i ist und im positiven Fall mit a_{min} in das Geschwindigkeitsprofil geschrieben. Erst, wenn v'_{max} größer v_i ist, also von v_i direkt abgebremst werden kann, terminiert die Berechnung.

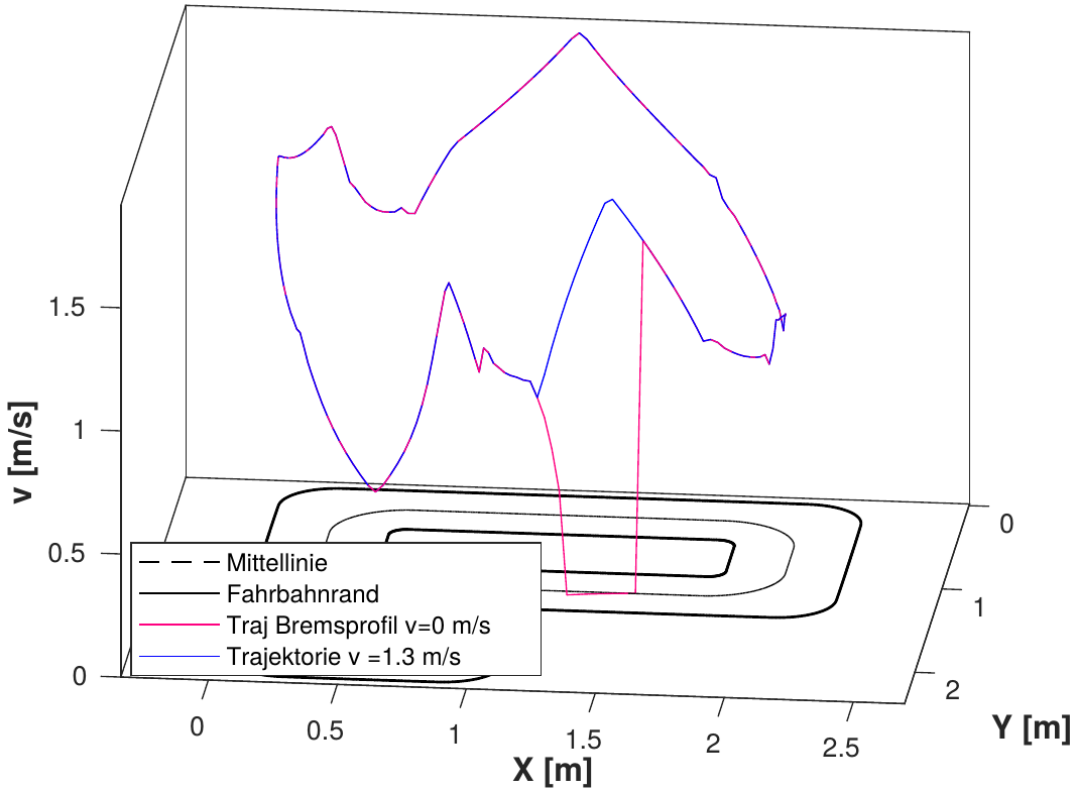


Abbildung 12.15.: Vergleich einer Trajektorie mit Bremsprofil und einer Trajektorie mit originalem Geschwindigkeitsprofil.

Die Berechnung des Geschwindigkeitsprofils ist nur für eine Position und Geschwindigkeit des voraus fahrenden Fahrzeugs gültig. Daher wird es im Regelungsmodul jeweils mit aktuellen Positions- und Geschwindigkeitsdaten aufgerufen. Stünde das folgende Fahrzeug bei Rennstart im Bereich des Bremsprofils, so würde es schlagartig die maximale Bremsbeschleunigung einstellen und gegebenenfalls unkontrolliert rückwärts beschleunigen. Um diesen Effekt auszuschließen, wird das Bremsprofil nur in die Trajektorie geschrieben, falls die Geschwindigkeit des eigenen Fahrzeugs größer 0 m/s ist.

Fahrverhalten 2: Überholen

Das Fahrverhalten „Überholen“ wird aktuell nur zwischen zwei vorgegebenen Knotenpunkten realisiert, die im Config-File eingetragen sind. An diesen festgelegten Punkten spalten sich zwei Trajektorien bzw. werden sie wieder zusammengeführt. Die Trajektorien werden in der Vorplanung generiert. Im Rennbetrieb fahren beide Fahrzeuge zunächst auf einer der beiden Trajektorien, jeweils mit ihren zugewiesenen Geschwindigkeiten. Die andere Trajektorie wird von einem Fahrzeug für das Überholen verwendet. Sie ist zwischen den Knotenpunkten hinsichtlich Geschwindigkeitsprofil und Streckenverlauf für einen Überholvorgang optimiert.
Um einen Überholvorgang einzuleiten, müssen mehrere Bedingungen erfüllt sein. Das überholende Fahrzeug muss sich weniger als fünf Fahrzeuglängen hinter dem zu überholenden Fahrzeug befinden und gleichzeitig den ersten Knotenpunkt überschritten haben. Trifft dies zu, wechselt das Fahrzeug die Trajektorie und verwendet kein Bremsprofil zum Folgen des Fahrzeugs. Sobald das Fahrzeug am anderen Fahrzeug vorbeigefahren ist und die Distanz zwischen den Fahrzeugen größer wird, wechselt es wieder auf die Originaltrajektorie. Solange sich das Fahrzeug mit dem Fahrverhalten „Überholen“ nicht im Bereich der Knotenpunkte befindet, fährt es nach dem Fahrverhalten „passives Fahren“ und verwendet das Bremsprofil. In Abbildung 12.9 sind die beiden Trajektorien mit ihrem Geschwindigkeitsprofil dargestellt.

Fahrverhalten 3: Überholt werden

Das Fahrverhalten „Überholt werden“ gleicht dem Fahrverhalten „passives Fahren“. Der einzige Unterschied ist, dass wie beim Fahrverhalten „Überholen“ zwei Trajektorien durch die Vorplanung berechnet werden. Es wird jedoch nur die langsamere Trajektorie verwendet, sodass das andere Fahrzeug die selbe Trajektorie fahren kann, bis dieses einen Überholvorgang einleiten kann. In Abbildung 12.10 ist die Trajektorie für das Fahrverhalten „Überholt werden“ dargestellt.

Evaluation von Regelungsparametern und Laufzeit

In diesem Abschnitt wird zunächst auf die empirischen Ermittlung von Parametern für die Regelung eingegangen. Dies betrifft die Gewichte für die MPC, den Kostenfaktor für Lenkung und Beschleunigung, die Schrittanzahl der MPC und die Latenz bezüglich dem Berechnen und Versenden der Regelungsdaten. Anschließend erfolgt eine Beschreibung einer Laufzeitevaluation.

Empirische Werte Die Gewichte der MPC wurden durch mehre Tests festgelegt. Da die Position des Fahrzeugs in Fahrtrichtung als wichtigster Parameter angesehen wurde, ist die Gewichtung hier besonders hoch gesetzt. Die Fahrtrichtung wird durch den Parameter w_x beschrieben und wurde auf 1.0 gesetzt. Durch eine hohe Gewichtung auf w_x wird erreicht, dass das Fahrzeug sich möglichst auf Höhe der Nominaltrajektorie in Fahrtrichtung befindet. Um in der Fahrtrichtung eine hohe Genauigkeit zu erreichen, sollte das Gewicht auf die Geschwindigkeit geringer gehalten werden. So ist es dem Fahrzeug möglich Fehler in der Geschwindigkeit hinzunehmen um eine hohe Genauigkeit in der Fahrtrichtung zu erhalten. Die Gewichtung für den Versatz von der Nominaltrajektorie orthogonal zur nominalen Fahrtrichtung wird durch den Parameter w_y beschrieben. Dieser Wert ist auf 0.7 gesetzt. Um hier eine gute Genauigkeit zu erreichen, sollte die Gewichtung auf den Ausrichtungswinkel geringer gehalten werden. Die Gewichtung für die Geschwindigkeit wird durch w_v beschrieben und wurde auf 0.2 gesetzt. Die Gewichtung für den Ausrichtungswinkel ist durch w_ψ beschrieben und wurde auf 0.01 gesetzt. Alle Gewichte stehen in Relation zueinander. Eine feste Obergrenze ist dadurch nicht gegeben.
Für Werte der Kosten ru_1, welche die Kosten der Lenkung beschreiben, und ru_2, welche die Kosten für die Beschleunigung beschreiben, wurden ebenfalls empirische Werte bestimmt. Wird der Wert ru_1 auf 0 gesetzt, bedeutet dies, dass Lenken keine Kosten verursacht. Es hat sich herausgestellt, dass das Fahrzeug beim Geradeausfahren einen dauerhaften maximalen beziehungsweise minimalen Lenkeinschlag im Wechsel vollzieht. Dadurch schlingert das Fahrzeug hin und her und übersteuert. Dies führt zu einem ungleichförmigen Fahrtverlauf. Eine minimale Erhöhung des Wertes auf einen Wert von 0.001 hat das Problem behoben. Die Kosten für die Beschleunigung wurde auf 0 gesetzt. Dadurch verursacht Beschleunigen keine Kosten. Eine Erhöhung dieses Wertes führte teilweise zum Stillstand des Fahrzeugs. Auch bei einer minimalen Erhöhung wurden entweder keine oder eine zu starke Veränderung festgestellt, sodass ein reibungsloser Ablauf nicht mehr stattfinden konnte.
Bei einer Latenz von 0 hat sich gezeigt, dass das Fahrzeug vor allem in Kurven nicht der Trajektorie folgt, sondern einen zeitlichen Versatz zu dieser besitzt. Daraus lässt sich schließen, dass die Regelungswerte zeitverzögert am Fahrzeug umgesetzt werden und so ein zu spätes Einlenken auftritt. Durch die Erhöhung dieses Latenzwertes hat sich die Fahrdynamik verbessert, sodass das Fahrzeug in Kurven der Trajektorie deutlich besser folgen kann. Ein Wert von 4 wurde gewählt, da eine Erhöhung keine Verbesserung erzielte, eine Verringerung jedoch einen Versatz hervorruft. Ab einem Wert von 20 wird die Fahrdynamik wieder deutlich schlechter.

Bei der Schrittanzahl N wurde zu Beginn ein Wert von 40 gewählt. Durch den Wert dauerte die Berechnung der Steuersignale jedoch zu lange, sodass das Fahrzeug nicht rechtzeitig mit Regelungswerten versorgt wurde. Die Folge waren Kollisionen mit der Fahrbahnbegrenzung. Durch ein Herabsetzen der Schrittanzahl konnte eine rechtzeitige Berechnung von Regelungswerten gewährleistet werden. Ein N von 20 wurde hier zunächst als Wert festgelegt. Da die Anforderung Sys4.8 eingehalten werden muss, darf eine Berechnung jedoch nicht länger als 10 ms benötigen. Dies ist erst durch einen Wert von $N = 18$ gegeben. Im nachfolgenden Paragraphen wird auf die Laufzeit näher eingegangen.

Laufzeit Um die Laufzeit des Regelungsmoduls festzustellen, wurde eine Zeitmessung für jeden Durchlauf des Regelungsmoduls eingebaut. Die Messung läuft dabei über 60 s und wurde dabei für jedes Fahrverhalten aufgezeichnet.
Für das Fahrverhalten „passives Fahren“ wurden zwei unterschiedliche Messungen durchgeführt. Zum Einen für ein einziges Fahrzeug auf der Rennstrecke ohne Update des Geschwindigkeitsprofils, zum Anderen für zwei Fahrzeuge auf der Rennstrecke mit Update des Geschwindigkeitsprofils.
In Abbildung 12.16 ist die Laufzeit ohne Update des Geschwindigkeitsprofil dargestellt. Es ist sowohl die jeweilige Laufzeit pro Durchlauf des gesamten Regelungsmodul über 60 s, als auch die jeweilige Laufzeit, die Gurobi in dem jeweiligen Durchlauf benötigt, zu entnehmen. Zusätzlich ist der Mittelwert angegeben. Dieser beträgt hier 0.0071 s.

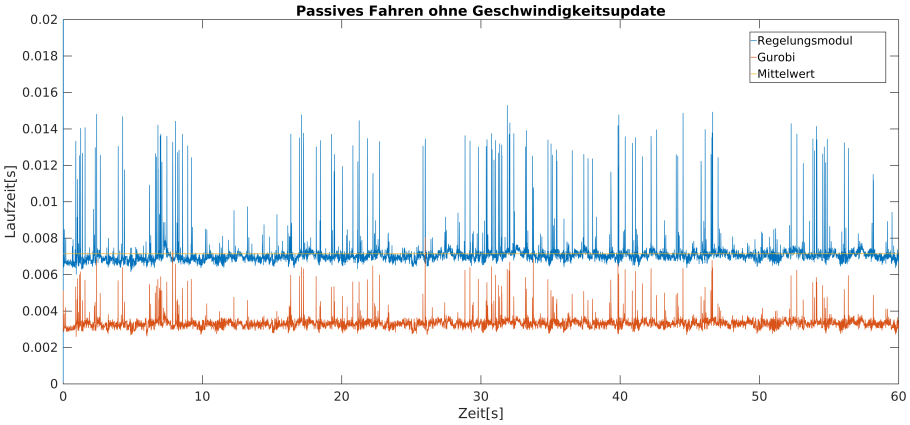


Abbildung 12.16.: Laufzeitmessung eines Fahrzeuges mit Fahrverhalten „passives Fahren“.

In Abbildung 12.17 ist die Laufzeit mit Update des Geschwindigkeitsprofils dargestellt. Es ist sowohl die jeweilige Laufzeit pro Durchlauf des gesamten Regelungsmodul über 60 s, als auch die jeweilige Laufzeit, die Gurobi in dem jeweiligen Durchlauf benötigt, zu entnehmen. Zusätzlich ist der Mittelwert angegeben. Dieser beträgt hier 0.0074 s. Das Ansteigen der Laufzeit ist auf das Updaten des Geschwindigkeitsprofils zurückzuführen.

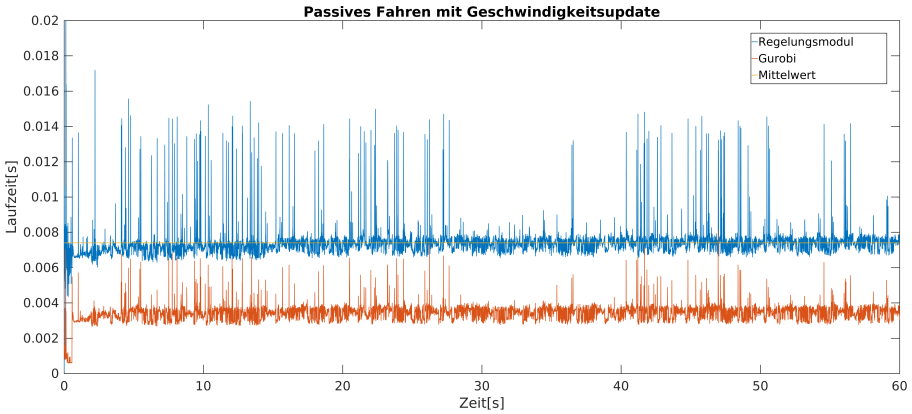


Abbildung 12.17.: Laufzeitmessung eines Fahrzeuges mit Fahrverhalten „passives Fahren“.

In Abbildung 12.18 ist die Laufzeit für das Fahrverhalten „Überholen“ dargestellt. Es ist sowohl die jeweilige Laufzeit pro Durchlauf des gesamten Regelungsmodul über 60 s, als auch die jeweilige Laufzeit, die Gurobi in dem jeweiligen Durchlauf benötigt, zu entnehmen. Zusätzlich ist der Mittelwert angegeben. Dieser beträgt hier 0.0075 s. Das Ansteigen der Laufzeit ist auf das Updaten des Geschwindigkeitsprofils und den Wechsel der Trajektorie zurückzuführen.

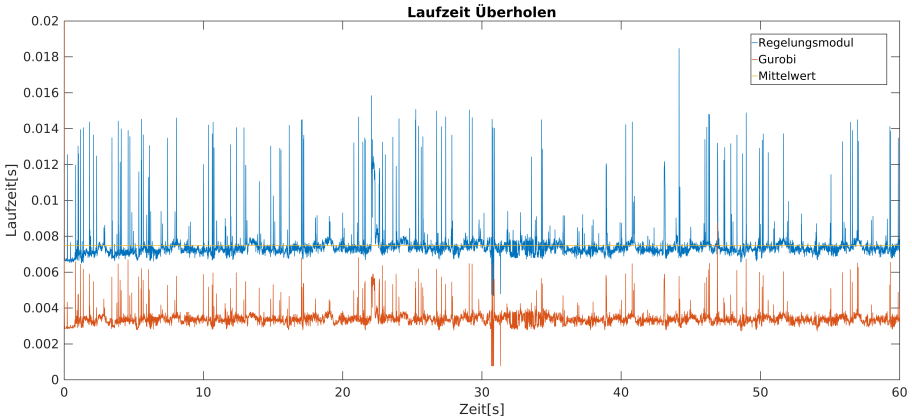


Abbildung 12.18.: Laufzeitmessung eines Fahrzeuges mit Fahrverhalten „Überholen“.

In allen Laufzeitmessungen konnte die Anforderung Sys4.8 erfüllt werden, da die Laufzeit 10 ms im Durchschnitt nicht überschreitet. Die Ausreißer konnten während der Projektgruppe nicht begründet werden. Vermutungen bezüglich dieser Ausreißer sind Bildverlust, Relaxierung durch die MPC, eine veränderte Taktrate der Kamera und eine hohe Prozessorauslastung. Auffällig ist hier, dass Gurobi immer zur selben Zeit einen Anstieg in der Zeit besitzt. Es ist deutlich zu sehen, dass „gurobi“ in allen Laufzeitmessungen ungefähr die Hälfte der benötigten Laufzeit in Anspruch nimmt.

12.4. Subsystem 4: Rennstrecke

Zur Erkennung der Rennstreckenrotation ist eine neue Markierung (siehe Abbildung 12.19) der Rennstrecke hinzugefügt. Diese ist an einer im Config-File ausgezeichneten Position der Rennstrecke anzubringen. Um eine feste Anbringung zu ermöglichen, wurden Halterungen für die Eckmarker, welche direkt an der Rennstrecke angebracht werden können, hergestellt. Eine Gegenüberstellung der alten Markerhalterung und der neuen ist in Abbildung 12.4 zu dargestellt.

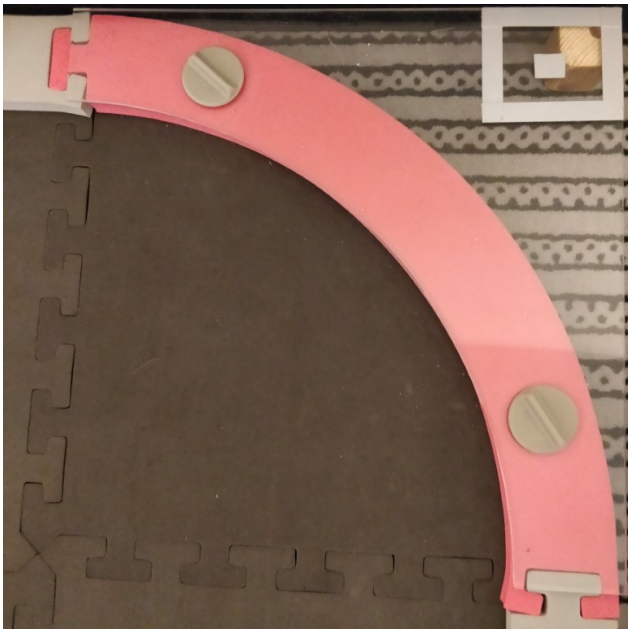
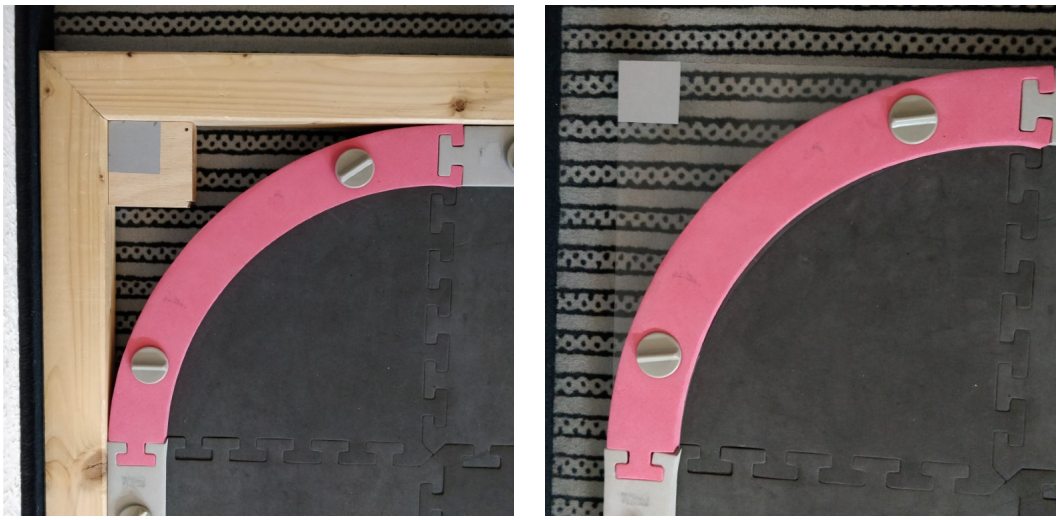


Abbildung 12.19.: Neuer Eckmarker.



(a) Alte Halterung (b) Neue Halterung

Abbildung 12.20.: Gegenüberstellung der Markerbefestigungen.

12.5. Subsystem 5: Systemsteuerungssoftware

Die Systemsteuerungssoftware ist für die Steuerung und Überwachung der anderen Komponenten im System verantwortlich. Sie kann über ein Benutzerinterface von einem Administrator des Systems bedient werden. Die Systemsteuerungssoftware ist zum größten Teil in Java geschrieben wurden und hat eine Schnittstelle über die Bibliothek JNA zu den C Funktionen der Netzwerkklasse `NetworkSharedLib` (siehe Abschnitt 11.6).
Zunächst musste die SSW darauf angepasst werden, mit mehreren Fahrzeugen, also zusätzlichen Control-Units kommunizieren zu können. Dazu wurde die Benutzeroberfläche so angepasst, dass je nach angegebener Anzahl von Fahrzeugen dynamisch Eingabemöglichkeiten für Konfigurationsdaten erscheinen. Die Konfigurationsdaten, die durch den Administrator für jede Control-Unit festgelegt werden müssen, bestehen daraus, welche Mindestdurchschnittsgeschwindigkeit erreicht werden soll, welches Fahrverhalten benutzt wird und welche Fahrzeug ID von der jeweiligen Control-Unit angesteuert werden muss. Um die Eingabe zu vereinfachen, entsprechen die möglichen Tabs „Vehicle Settings“ 1-16 den Control-Units mit der ID 301-316. Ein Beispiel, wie eine solche Eingabe in der Software aussieht, ist in Abbildung 12.21 zu sehen.

Im nächsten Schritt wurde die Systemsteuerungssoftware auf den neuen Zustandsablauf der Control-Units angepasst (siehe Abbildung 9.3). Dazu wurden benötigte fehlende Zustände eingepflegt und der Ablauf auf die neue Struktur angepasst.

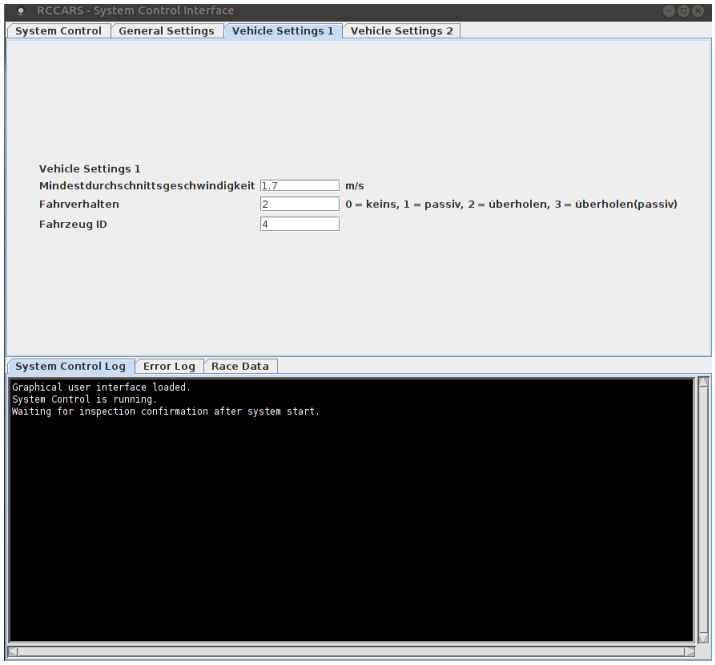


Abbildung 12.21.: In dieser Abbildung ist die Systemsteuerungssoftware mit 2 Eingabefenster für Konfigurationsdaten zu sehen.

Sobald alle Eingaben in der Systemsteuerungssoftware getätigt wurden und eine Überprüfung abgeschlossen wurde, kann die Initialisierung des Systems über einen Button gestartet werden. Die angepasste Initialisierung stellt innerhalb der SSW nun sicher, dass alle erwarteten Control-Units und die Positionsbestimmung sich im System anmelden, ihre Konfigurationsdaten zugesendet bekommen und erfolgreich nach der Vorplanung in einen Standby-Modus übergehen. Anschließend kann, wie gewohnt, der Rennzustand über die Systemsteuerungssoftware gestartet werden. Die Projektgruppe *RCCARSng* hat sich zudem dazu entschlossen, nicht wieder von dem erreichten Standby-Modus zurück in einen Initialisierungsmodus wechseln zu können. In dem neuen Konzept inklusive der Vorplanung der Control-Units ist dieser Vorgang nicht vorgesehen und wurde deshalb auch aus der SSS entfernt.
Im Rahmen der Netzwerkstrukturanpassungen, die im folgenden Abschnitt 12.6 beschrieben werden, ist auch die Java Schnittstellen der Systemsteuerungssoftware entsprechend angepasst. Hierbei wurde eine eigens für Java angepasste Union-Klasse geschrieben und die Pakete entsprechend der in C geschriebenen Netzwerkstruktur angepasst.

Weiterhin wurde zur Visualisierung der Renndaten während des Rennbetriebs ein weiterer Tab der SSW hinzugefügt. Dieser zeigt die Durchschnittsgeschwindigkeit, Rundenanzahl und Rundenzeit der individuellen Fahrzeuge an. Die Renndatenberechnung findet dabei innerhalb einer eigenen Klasse aufgrund der empfangenen Fahrzeugdaten der Positionsbestimmung statt.

12.6. Netzwerk

Im Laufe der Projektgruppenarbeit wurden zwei größere Änderungen an der Netzwerkstruktur vorgenommen.
Zum Einen wurde die Größe des Netzwerkpakets reduziert, indem die 6 verschiedenen Pakettypen (Fahrzeugdaten, Kontrolldaten, Fehlerdaten, Zeitdaten, Hindernisdaten, Konfigurationsdaten), die das System erlaubt, als Union zusammengefasst wurden. In dem alten Netzwerkpaket war es so vorgesehen, dass Information über alle 6 Pakettypen immer mitgesendet wurde, obwohl nur ein Pakettyp benötigt wurde. Diese Optimierung reduzierte die Größe des Pakets von 1840 Byte auf 1072 Byte. Besonders hervorzuheben ist hierbei, dass die Größe des Netzwerkpakets nun unter der Grenze der maximalen Nutzlast eines Datenframes des Ethernet-Standards von 1500 Bytes liegt und somit die Anzahl der Pakete, die versendet werden, effektiv halbiert werden konnte, da Pakete, die größer als 1500 Bytes sind, gesplittet werden. Der neue Aufbau des Netzwerkpakets ist in Abbildung 12.22 zu sehen.

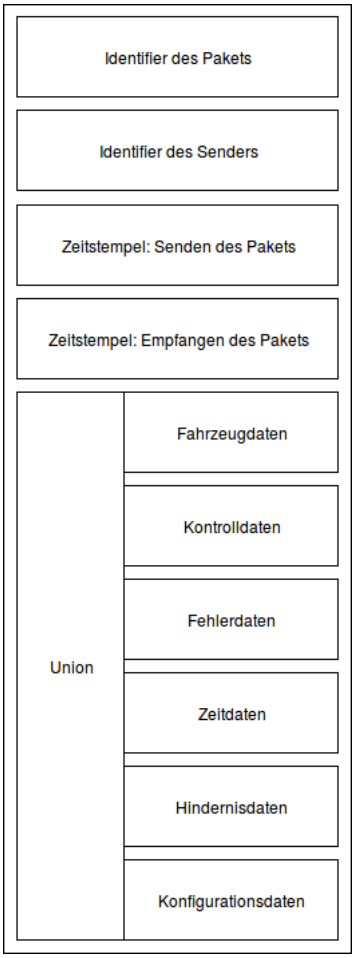


Abbildung 12.22.: In dieser Abbildung ist der Aufbau des Netzwerkpakets zu sehen. Der Identifier des Pakets gibt hierbei an, welcher Datentyp in dem Union gespeichert wurde.

Zum Anderen wurde das Empfangen von Kontrolldaten verbessert. Da in dem System Zustandswechselbefehle sowie die Bestätigung, das ein Zustandswechsel durchgeführt worden ist, via Kontrolldaten erfolgt und alle Netzwerkteilnehmer diese Daten aufgrund des Broadcasting empfangen, konnte es passieren, dass ein Teilnehmer ein für sich bestimmtes Paket nicht rechtzeitig auslesen konnte.
Dieses Problem wurde dadurch behoben, dass nun konfigurierbar ist, welche Kontrolldaten gespeichert werden sollen und welche nicht.

13. Teststatus

In diesem Kapitel wird der Teststatus des Systems vorgestellt. Hierbei wird subsystemweise vorgegangen und erklärt, was getestet wurde und welche Anforderungen (siehe Kapitel 7) erfüllt werden konnten. Die Betrachtung der Subsysteme erfolgt nach Hardware und Software getrennt, da auch die Anforderungen entsprechend definiert wurden. Eine detaillierte Übersicht über die konkreten Testfälle findet sich im Testdokument der Projektgruppe *RCCARSng*.

13.1. Fahrzeuge

Die Fahrzeuge wurden ohne Veränderung aus dem Setup der Gruppe *RCCARS* übernommen. Sie wurden durch Integrationstest auf Funktionsfähigkeit überprüft. Somit wurden sämtliche Anforderungen, die an die Fahrzeuge gestellt wurden, erfüllt (siehe Abbildung 13.1). Die Evaluation der Fahrzeuge findet sich in [RCC16];

HW1				JA
	HW1.1			JA
	HW1.2			JA
	HW1.3			JA
		HW1.3.1		JA
		HW1.3.2		JA
	HW1.4			JA
		HW1.4.1		JA
		HW1.4.2		JA
		HW1.4.3		JA
		HW1.4.4		JA
		HW1.4.5		JA
		HW1.4.6		JA

Abbildung 13.1.: Hier zu sehen ist, dass die Anforderungen HW1 mit ihren Unteranforderungen komplett erfüllt werden konnte. Eine grüne Markierung steht hierbei für erfüllte Anforderungen, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

13.2. Positionsbestimmung

Die Positionsbestimmung ist ein wichtiger Bestandteil, der funktionalen und zeitlichen Constraints unterliegt. So musste durch Tests sichergestellt werden können, dass die Erkennung von Fahrzeug- und Hindernisdaten (wie beispielsweise die Anzahl und ID's der Fahrzeuge oder die Position von Hindernissen) korrekt funktioniert. Hier wurden bestehende CUTE-Tests [ESR] angepasst und erweitert.
Zuerst wird jedoch die Hardwareseite der Positionsbestimmung betrachtet. Dies schließt Komponenten wie die Kamera und Infrarotstrahler, einen Rechner, das Gestell und notwendige Kabel mit ein.

Hardwaretests Die Positionsbestimmung wurde hardwareseitig um verschiedene Komponenten erweitert. Zum einen wurden Halter für Eckmarker der Rennstrecke entworfen, die eine stabile Positionierung der Reflexionsmarker ermöglichen.
Weiterhin wurde ein höhenverstellbares Gestell entworfen, welches fest an der Decke montiert wurde, sodass das System nun weniger störanfällig ist.
Die bestehende Hardware der Projektgruppe *RCCARS* und die beiden neuen vorgestellten Komponenten wurden durch Integrationstest auf ihre Funktionstüchtigkeit überprüft und in das System übernommen. Für eine detaillierte Evaluation der bestehenden Hardware siehe Kapitel 7 im Endbericht der Projektgruppe *RCCARS* [RCC16]. Insgesamt konnten alle Anforderungen, die das Subsystem Positionsbestimmung betreffen, erfüllt werden. In Abbildung 13.2 ist der Stand der aktuellen Anforderungen aufgelistet.

Softwaretests Die Software der Positionsbestimmung musste an vielen Stellen angepasst und erweitert werden, damit jetzt auch mehrere Fahrzeuge und Hindernisse erkannt werden konnten. Somit mussten auch bestehende Tests angepasst, und neue Tests erstellt werden. An dieser Stelle ist die Zeit, die die Bilderkennung benötigt, um das System realzeitfähig zu halten, besonders wichtig. Aktuell benötigt die Positionsbestimmung $\approx$7 ms für eine solche Bilderkennung und befindet sich damit unter der Grenze von 10 ms. Diese Zeit ist maximal erlaubt, um eine sichere Steuerung der Fahrzeuge zu gewährleisten. In Abbildung 13.3 findet sich der Stand der Software-Anforderungen der Positionsbestimmung.

HW2				JA
	HW2.1			JA
	HW2.2			JA
		HW2.2.1		JA
		HW2.2.2		JA
		HW2.2.3		JA
		HW2.2.4		JA
		HW2.2.5		JA
	HW2.3			JA
		HW2.3.1		JA
			HW2.3.1.1	JA
			HW2.3.1.2	JA
			HW2.3.1.3	JA
			HW2.3.1.4	JA
			HW2.3.1.5	JA
		HW2.3.2		JA
		HW2.3.3		JA
			HW2.3.3.1	JA
			HW2.3.3.2	JA
			HW2.3.3.3	JA
	HW2.4			JA
		HW2.4.1		JA
		HW2.4.2		JA
		HW2.4.3		JA
		HW2.4.4		JA
		HW2.4.5		JA
		HW2.4.6		JA
		HW2.4.7		JA

Abbildung 13.2.: Teststatus für die Hardware-Anforderungen der Positionsbestimmung. Eine grüne Markierung steht hierbei für erfüllte, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

SW1				JA
	SW1.1			JA
	SW1.2			JA
		SW1.2.1		JA
		SW1.2.2		JA
		SW1.2.3		JA
		SW1.2.4		JA
		SW1.2.5		JA
	SW1.3			JA
		SW1.3.1		JA
		SW1.3.2		JA
	SW1.4			JA
	SW1.5			JA
	SW1.6			JA
	SW1.7			JA
	SW1.8			JA
		SW1.8.1		JA
		SW1.8.2		JA

Abbildung 13.3.: Teststatus für die Software-Anforderungen der Positionsbestimmung. Eine grüne Markierung steht hierbei für erfüllte, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

13.3. Control-Unit

Auch die Control-Unitunterliegt strengen Beschränkungen, da hier die Steuerbefehle für die Fahrzeuge berechnet und gesendet werden.

Hardware Damit mehre Fahrzeuge gesteuert werden können, war es notwendig, dass neue Rechner beschafft und eine neue CarControl erstellt wurden. Dies waren die einzigen hardwareseitigen Erweiterungen der Control-Unit. Diese Komponenten wurden entsprechend des Testprozesses (siehe Abschnitt 3.8) erfolgreich getestet. So wurden alle Hardwareanforderungen an die Control-Unit erfüllt, da bestehende Komponenten bereits in [RCC16] getestet wurden (siehe Abbildung 13.4).

HW3				JA
	HW3.1			JA
	HW3.2			JA
		HW3.2.1		JA
		HW3.2.2		JA
		HW3.2.3		JA
		HW3.2.4		JA
		HW3.2.5		JA
	HW3.3			JA
		HW3.3.1		JA
		HW3.3.2		JA
		HW3.3.3		JA
		HW3.3.4		JA
		HW3.3.5		JA
		HW3.3.6		JA
		HW3.3.7		JA
		HW3.3.8		JA

Abbildung 13.4.: Die Anforderungen an die Hardware der Control-Unit konnten komplett erfüllt werden. Sämtliche notwendige Hardware ist im System vorhanden und wurde getestet. Eine grüne Markierung steht hierbei für erfüllte Anforderungen, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

Software Die Software der Control-Unit ist für das sichere Fahren der Fahrzeuge auf der Rennstrecke verantwortlich. Aus Zeitgründen war es nicht mehr möglich, die Software gemäß unseres Testprozesses (siehe Abschnitt 3.8) zu testen. Es wurden an dieser Stelle nur Integrationstests durchgeführt, um sicherzustellen, dass die einzelnen Softwarekomponenten korrekt zusammenarbeiten. Da es in der Control-Unit die Möglichkeit gibt, einige Module gezielt auszuschalten, konnten so einzelne Module isoliert betrachtet werden und somit gezielter getestet werden. Zudem wurde das System während der Entwicklung und der Parameterevaluation regelmäßig in

Betrieb genommen, sodass Anforderungen an die Control-Unit an dieser Stelle lediglich durch Integrationstest erfüllt werden konnten (siehe Abbildung 13.5).
Die Anforderung SW2.3 betrifft die Dauer der Regelungsdatenberechnung und konnte nicht erfüllt werden, da im Rennbetrieb die Dauer dieser Berechnung den Wert von 10 ms überschreiten kann.
Die Anforderung SW2.1.7 betrifft die Einhaltung der Mindestdurchschnittsgeschwindigkeit. Diese Anforderung kann nicht eingehalten werden, wenn die Fahrzeuge durch das passive Fahren abbremsen müssen, wenn ein vorausfahrendes Fahrzeug langsam fährt.
Die Anforderungen SW2.1.12.1 bis SW2.1.12.4 betreffen die Kollisionserkennung. Im aktuellen Zustand ist die Kollisionserkennung nicht funktionstüchtig und kann demzufolge keine Kollision eines Fahrzeugs erkennen.
Im aktuellen Setting existieren noch Software-Fehler, die einen Systemabsturz der Control-Unit verursachen. Dieser Fehler wird durch einen Watchdog auf der Car-Control abgefangen, wodurch das Fahrzeug in einen sicheren Fehlerzustand versetzt wird.

SW2				NEIN
	SW2.1			NEIN
		SW2.1.1		JA
		SW2.1.2		JA
		SW2.1.3		JA
		SW2.1.4		JA
		SW2.1.5		JA
		SW2.1.6		JA
		SW2.1.7		NEIN
		SW2.1.8		JA
		SW2.1.9		JA
			SW2.1.9.1	JA
			SW2.1.9.2	JA
		SW2.1.10		JA
			SW2.1.10.1	JA
			SW2.1.10.2	JA
			SW2.1.10.3	JA
			SW2.1.10.4	JA
		SW2.1.11		JA
		SW2.1.12		NEIN
			SW2.1.12.1	NEIN
			SW2.1.12.2	NEIN
			SW2.1.12.3	NEIN
			SW2.1.12.4	NEIN
			SW2.1.12.5	JA
			SW2.1.12.6	JA
			SW2.1.12.7	JA
			SW2.1.12.8	JA
			SW2.1.12.9	JA
			SW2.1.12.10	JA
			SW2.1.12.11	JA
		SW2.1.13		JA
	SW2.2			JA
		SW2.2.1		JA
		SW2.2.2		JA
		SW2.2.3		JA
		SW2.2.4		JA
	SW2.3			NEIN

Abbildung 13.5.: In diesem Bild ist der aktuelle Teststatus für die Software-Anforderungen der Control-Unit zu sehen. Zu beachten ist hierbei, dass aus Zeitgründen nur Integrationstests durchgeführt werden konnten, welche an dieser Stelle aber trotzdem eine Erfüllung von Anforderungen gewährleisten können. Eine grüne Markierung steht hierbei für erfüllte Anforderungen, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

13.4. Rennstrecke

Das Subsystem Rennstrecke musste um Hindernisse erweitert werden. Die eigentliche Rennstrecke wurde unverändert übernommen und durch die Projektgruppe *RCCARS* getestet. Somit sind die Anforderungen an die Rennstrecke erfüllt. Die Hindernisse sind aus einer Schaumstoffmatte der Dicke 2.9 cm gefertigt. Länge und Breite der Hindernisse sind jeweils $i \cdot 5\,\text{cm}, i \in \{1, 2, 3, 4\}$. Somit sind die Hardwareanforderungen HW4.4.1 und HW4.4.2 erfüllt. (siehe Abbildung 13.6).

HW4				JA
	HW4.1			JA
	HW4.2			JA
	HW4.3			JA
	HW4.4			JA
		HW4.4.1		JA
			HW4.4.1.1	JA
			HW4.4.1.2	JA
			HW4.4.1.3	JA
			HW4.4.1.4	JA
		HW4.4.2		JA
		HW4.4.3		JA
		HW4.4.4		JA
		HW4.4.5		JA
		HW4.4.6		JA
		HW4.4.7		JA

Abbildung 13.6.: Hier zu sehen ist, dass alle Anforderungen an das Subsystem Rennstrecke erfüllt werden konnten. Eine grüne Markierung steht hierbei für erfüllte Anforderungen, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

13.5. Systemsteuerungssoftware

Die Systemsteuerungssoftware sorgt für die Kontrolle und Steuerung des Systems. Auch an dieser Stelle konnten bedingt durch den in Abschnitt 13.3 beschrieben Zeitmangel nur Integrationstest durchgeführt werden, um die Anforderungserfüllung sicherzustellen. Die Anforderung SW3.1.5 konnte nicht erfüllt werden, da das System im aktuellen Zustand nicht ordnungsgemäß gestoppt, also in den Standbymodus versetzt, werden kann. Eine Übersicht über die Anforderungen findet sich in Abbildung 13.7.

13.6. Generelle Systemanforderungen

An dieser Stelle werden die generellen Systemanforderungen vorgestellt. Diese Anforderungen stellen die groben Anforderungen an das System, die zur Erfüllung des Szenarios erfüllt werden müssen. Eine Übersicht über den Status der Anforderungserfüllung findet sich in Abbildung 13.8.
Die Anforderungen Sys4.6 und Sys4.7 betreffen die Kollisionserkennung und können wie in 13.3 beschrieben nicht erfüllt werden.

SW3				JA
	SW3.1			JA
		SW3.1.1		JA
		SW3.1.2		JA
		SW3.1.3		JA
		SW3.1.4		JA
		SW3.1.5		NEIN
		SW3.1.6		JA
		SW3.1.7		JA
		SW3.1.8		JA
		SW3.1.9		JA
		SW3.1.10		JA
		SW3.1.11		JA
		SW3.1.12		JA
	SW3.2			JA
		SW3.2.1		JA
		SW3.2.2		JA

Abbildung 13.7.: Hier zu sehen ist der Stand der Anforderungen der Systemsteuerungssoftware. Eine grüne Markierung steht hierbei für erfüllte Anforderungen, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

Sys1		JA
	Sys1.1	JA
	Sys1.2	JA
	Sys1.3	JA
	Sys1.4	JA
	Sys1.5	JA
	Sys1.6	JA
Sys2		JA
	Sys2.1	JA
	Sys2.2	JA
	Sys2.3	JA
	Sys2.4	JA
	Sys2.5	JA
	Sys2.6	JA
	Sys2.7	JA
Sys3		JA
Sys4		NEIN
	Sys4.1	JA
	Sys4.2	JA
	Sys4.3	JA
	Sys4.4	JA
	Sys4.5	JA
	Sys4.6	NEIN
	Sys4.7	NEIN
	Sys4.8	JA
Sys5		JA
	Sys5.1	JA

Abbildung 13.8.: Hier zu sehen ist der Stand der generellen Systemanforderungen. Eine grüne Markierung steht hierbei für erfüllte Anforderungen, gelb für noch nicht getestete, und rot für Anforderungen, wo die Tests fehlgeschlagen sind.

14. Ergebnisse und Ausblick

In diesem abschließenden Kapitel werden die Ergebnisse der Projektgruppe *RCCARSng* zusammengefasst und mit dem als Projektziel angesetzten Szenario verglichen. Ferner gibt es in einem weiteren Abschnitt einen Ausblick auf weitere mögliche Arbeitsthemen.

14.1. Ergebnisse

Die Projektgruppe *RCCARSng* hat sich ein Szenario (siehe Abschnitt 2.3) mit mehreren Varianten überlegt, dessen Erfüllung als Ziel dieser Projektgruppe festgelegt wurde.
Die erste Variante „Überholvorgang“ konnte erfolgreich umgesetzt werden. Das System ist in der Lage, zwei Fahrzeuge so über die Rennstrecke zu steuern, dass ein Fahrzeug das andere Fahrzeug überholen kann. Der Überholvorgang wurde hierbei jedoch nicht so realisiert, wie die Projektgruppe sich den Vorgang anfangs überlegt hatte. Die ersten Überlegungen zielten auf einen dynamischen Überholvorgang, bei dem die Trajektorien für die Fahrzeuge online berechnet wurden, ab. Im aktuellen Zustand findet diese Trajektorieberechnung vor dem eigentlichen Rennbetrieb statt.
Auch die zweite Variante „Folgen anderer Fahrzeuge“ konnte erfolgreich umgesetzt werden. In dieser Variante bremst ein Fahrzeug ab, falls es zu dicht auf ein anderes Fahrzeug auffährt, indem das Geschwindigkeitsprofil, welches die Geschwindigkeit des Fahrzeugs beschreibt, zur Laufzeit angepasst wird.
Die dritte Variante „Hindernisintegration“ erlaubt die Positionierung von Hindernissen auf der Rennstrecke. Falls das Fahrverhalten „Überholen“ ausgewählt wurde, muss gewährleistet sein, dass zwei verschiedene Trajektorien um die Hindernisse gefunden werden können. Auch diese Bedingung erfüllt die Definition der Variante.
Insgesamt lässt sich also sagen, dass das Szenario mit den drei Varianten erfolgreich umgesetzt werden konnte, auch wenn das System nicht so dynamisch läuft, wie es anfangs angedacht war.
Um dieses Szenario zu realisieren wurde das System der Projektgruppe *RCCARS* übernommen und an vielen Stellen erweitert oder komplett überarbeitet. So mus-

ste die Positionsbestimmung angepasst werden, sodass mehrere Fahrzeuge und auch Hindernisse erkannt werden konnten und gleichzeitig die Berechnungsdauer für einzelne Bilder nicht die Grenze von 10 ms überschritt. Eine weitere Verbesserung war die Entwicklung einer Halterung für die Kamera und Infrarotstrahler, welches eine stabilere Befestigung dieser Komponenten sicherstellt und das System somit weniger störanfällig ist.
Die Control-Unit wurde im Laufe der Projektarbeit komplett überarbeitet. So wurden neue Rechner beschafft, sodass pro Fahrzeug ein Rechner zur Berechnung der Steuersignale zur Verfügung steht. Weiterhin wurden viele Module erstellt, die die sichere Steuerung eines Fahrzeuges gewährleisten und das System im Fehlerfall robust gestalten. Besondere Erwähnung findet an dieser Stelle die Entwicklung einer modellprädikativen Regelung, welche das Verhalten des Fahrzeuges vorhersagen kann und dadurch eine bessere Steuerung gewährleistet.
Auch die Systemsteuerungssoftware, die als Steuerungs- und Überwachungskomponente dient, wurde an die neuen Gegebenheiten angepasst. So können nun Einstellungen, wie das Fahrverhalten oder die Mindestdurchschnittsgeschwindigkeit, für die einzelnen Fahrzeuge festgelegt werden und gleichzeitig die Status der Control-Units überwacht werden.
Weitere Anpassungen betreffen das Netzwerk, in dem Informationen wie Konfigurationsdaten oder Hindernisdaten in Netzwerkpaketen verschickt werden können. Gleichzeitig ist es gelungen, die Größe der Pakete zu reduzieren und somit die Netzwerklast deutlich zu verringern.

14.2. Ergebnisse und Ausblick einzelner Komponenten

Fahrzeug

Die Hardware der Fahrzeuge ist in dieser Projektgruppe nicht angepasst worden. In folgenden Arbeiten könnte die in Michael Bukowskis' Masterarbeit evaluierte Fahrzeugplatine (siehe Abschnitt 4.4.2) eingebaut werden. Damit könnten Berechnungen der Control-Unit auf das Fahrzeug ausgelagert werden.

Positionsbestimmung

In den Abschnitten 11.2 und 12.2 wird der aktuelle Stand der Positionsbestimmung beschrieben. Es können mehrere Fahrzeuge gleichzeitig erkannt werden. Zudem werden Hindernisse erkannt. Des Weiteren wird eine Rotation der Rennstrecke erkannt.

Ein weiteres Problem ist immer noch die Verzerrung der aufgenommen Bilder. So weicht die von der Positionsbestimmung bestimmte Position von der per Hand gemessenen Position ab. Diese Abweichung entsteht durch die berechnete Transformationsmatrix und die Höhe der Objekte. Für eine genaue Bestimmung der Position müsste eine 3D-Rekonstruktion vorgenommen werden.

Control-Unit

In der Control-Unit gibt es verschiedene Optimierungsideen, die in den folgenden Abschnitten aufgeführt werden.

Krümmungsbegrenzte Trajektorienberechnung

Im unter 11.10 beschriebene Optimierungsproblem zur Trajektorienberechnung wird aktuell nur der freie Parameter d_i (Verschiebungsabstand der Stützpunkte) durch den Fahrbahnrand und gegebenenfalls Hindernisse begrenzt und anschließend die Summe der Krümmung minimiert. Hierbei wird die Krümmung zu keinem Zeitpunkt begrenzt, was Trajektorien erlaubt, welche nicht fahrbare Krümmungen aufweist. Daher wäre es sinnvoll, auch eine Krümmungsbegrenzung in die Constraints des Optimierungsproblems mit aufzunehmen.

Trajektorienberechnung in C++

Die Vorplanung benötigt in der Matlab-Implementierung je nach Rechner bis zu 1 min. Der größte Teil dieser Berechnungszeit wird vom Gurobi-Solver [Gur] in Anspruch genommen. Da die MPC-Regelung in der C++ Implementierung deutlich kürzere Laufzeiten im Vergleich zur Matlab-Simulation aufweist, gibt es die Überlegung, die Vorplanung ebenfalls vollständig in C++ zu implementieren. Die Projektgruppe *RCCARSng* erhofft sich damit eine kürzere Laufzeit.
Die verhältnismäßig lange Laufzeit der Vorplanung macht ein mehrfaches Neu- bzw. Umplanen von Trajektorien unpraktikabel. Aktuell wird die Trajektorienberechnung so selten wie nötig aufgerufen. Durch eine verkürzte Rechenzeit bestünde diese Einschränkung nicht mehr. Somit könnten in der Wegplanung um Hindernisse Trajektorien auf beiden Hindernisseiten berechnet und miteinander verglichen werden. Der Pfad würde nicht mehr zwangsläufig auf der Seite des kürzesten Pfades verlaufen. Des Weiteren hätte die verkürzte Rechenzeit die Basis für regelmäßige Trajektorienberechnungen während des Rennbetriebs.

Geschwindigkeitsanpassung

Die Anpassung des Geschwindigkeitsprofils auf Mindestdurchschnittsgeschwindig-

keit erfolgt derzeit durch Skalierung des maximierten Geschwindigkeitsprofils. Da hierbei jedoch alle Geschwindigkeiten herunterskaliert werden, weisen Stellen mit anfangs niedriger Geschwindigkeit nach der Skalierung eine noch geringere Geschwindigkeit auf, welche gegebenenfalls kleiner als notwendig ist. An dieser Stelle wäre eine Neuformulierung des Geschwindigkeitsprofils denkbar, welches Constraints an die Durchschnittsgeschwindigkeit stellt und somit direkt ein Geschwindigkeitsprofil mit gewünschter Durchschnittsgeschwindigkeit liefert. Dann wäre es jedoch nicht mehr sinnvoll das Problem nach maximaler Durchschnittsgeschwindigkeit zu optimieren, sondern die Beschleunigung für einen möglichst konstanten Geschwindigkeitsverlauf zu minimieren.

Modellprädikative Regelung

Da es sich bei der modellprädikativen Regelung um eine Regelung handelt, die Regelungswerte für die Zukunft berechnet, kann sich dieser Vorteil zu Nutze gemacht werden, um Bildverluste der Positionsbestimmung zu kompensieren. Zukünftig ist es sinnvoll, falls nach bestimmter Zeit kein Kamerabild empfangen wurde oder bei nicht validen Daten, die zweiten Regelungswerte, die im vorherigen Durchlauf durch die MPC berechnet wurden, an das Fahrzeug zu senden. Dies kann jedoch nicht beliebig oft, bis maximal N MPC-Schritte, durchgeführt werden, da die Daten nicht zwangsläufig noch aktuell sind. Dies kann vor allem beim Überholen der Fall sein.
Um eine noch bessere Regelung zu gewährleisten könnten die Parameter für die MPC, sowohl die Gewichte w_v, w_x, w_y, w_ψ als auch die Kosten ru_1 und ru_2, und die Motorparameter für die Berechnung der Beschleunigung und der Lenkwinkels weiter evaluiert und angepasst werden.
Ebenfalls kann die Latenz weiter untersucht werden, sodass der Grund für die hohe Latenz identifiziert wird.

Verbesserte Zustandsschätzung

In weiteren Projektarbeiten kann zudem noch eine verbesserte Zustandsschätzung des Fahrzeuges implementiert werden. Mit dieser Schätzung kann eine genauere Berechnung der Fahrzeugparameter durchgeführt werden. Des Weiteren hilft eine bessere Zustandsschätzung bei der Regelungsdatenbestimmung durch die MPC. Diese Schätzung kann Daten aus verschiedenen Quellen zu zusammenfassen. So können neben den Positionsdaten auch Daten der in Abschnitt 4.4.2 evaluierten Fahrzeugplatine zur Zustandsmessung verwendet werden. Somit kann hier ein „Sanity-Check“ durchgeführt werden. Dieser überprüft ob sich ein Fahrzeug entsprechend der gesendeten Signale verhält.

Systemsteuerungssoftware

Die Systemsteuerungssoftware (SSSW) ist im aktuellen Zustand in der Lage, das gesamte System zu initialisieren und anschließend den Rennbetrieb zu starten und wieder anzuhalten. Zudem bietet sie die Möglichkeit, durch verschiedene Konfigurationsparameter interne Prozesse und andere Module zu überwachen. Sollte ein sicherheitskritischer Fehler auftreten, versetzt die SSS das gesamte System in einen sicheren Fehlerzustand.
In der Systemsteuerungssoftware kann die Anzeige der Renndaten noch verbessert und übersichtlicher angezeigt werden. Weiterhin könnten beispielsweise aktuelle Systemzustände, Kontrolldaten und Fahrzeugdaten optional geloggt werden. Die Systemsteuerungssoftware könnte zudem eine Art Lexikon in der Benutzeroberfläche besitzen, in dem Fehlercodes des Systems nachgeschlagen werden können. Dadurch müsste nicht im Dokument nachgeschlagen werden. Die letzte, hier zu erwähnende mögliche Erweiterung, welche bereits durch die Projektgruppe *RCCARS* vorgeschlagen wurde, wäre eine Live-Anzeige des Renngeschehens in der SSS.

Netzwerk

Im aktuellen Zustand werden alle Netzwerkpakete mittels Broadcast versendet. Da insbesondere durch die Positionsbestimmung viele Pakete verschickt werden, wird die Internetnutzung aller Netzwerkteilnehmer stark beeinträchtigt. Zur Zeit wird das Problem dadurch umgangen, dass ein eigenes Netzwerk für Kommunikation innerhalb des Systems Verwendung findet. Wünschenswert wäre eine Lösung, die beispielsweise via Multicast arbeitet, sodass die Pakete gezielt nur an die Netzwerkteilnehmer geschickt werden, welche diese auch benötigen. Hierzu müsste in der Shared Library des Networkmessengers das Senden und Empfangen von Paketen auf Multicast umgestellt werden.

Des Weiteren ist eine Umstellung der Fehlercodes wünschenswert. Im aktuellen Zustand existieren 2 Fehlercodierungen. Zum Einen die externen Fehler (siehe Tabelle C.1) die zur Kommunikation zwischen den Subsystemen verwendet werden, zum Anderen die internen Fehlercodes der Control-Unit (siehe Tabelle C.2). Da die Control-Unit internen Fehlercodes einfacher zu Kodieren sind und sie mit ihr mehr Fehler kodieren lassen, bietet es sich an, diese oder eine leicht abgewandelte Kodierung in das Netzwerk zu ziehen. Hierbei müsste die Struktur des Netzwerkpaketes nicht angepasst werden, dennoch jedes Subsystem.

14.3. Fazit

Im folgenden Abschnitt werden die Probleme, die während des Projekts aufgetreten sind, besprochen. Das genutzte Vorgehensmodell wird im Folgenden zusammengefasst und im Hinblick auf die Probleme bewertet. Anschließend wird die Dynamik innerhalb der Gruppe beschrieben. Zum Ende findet noch eine Danksagung der Projektgruppe an alle Beteiligten statt.

Probleme

Während der Planungs- und Entwicklungszeit traten innerhalb der Projektarbeit in verschiedenen Bereichen unterschiedliche Probleme auf.
Die erste Hürde, die bewältigt wurde, war eine gemeinsame Basis der anfangs sieben Studierenden zu finden. Innerhalb des ersten Monats verließ ein Mitglied die Projektgruppe, wodurch bereits verteilte Aufgaben von den restlichen Gruppenmitgliedern übernommen werden mussten. In der folgenden Zeit bis zum 1. Review wurde nach keinem festen Vorgehensmodell gearbeitet, wodurch die Einarbeitungsphase in das Projekt *RCCARS* unorganisiert und länger als vielleicht notwendig ablief.
Nachdem die Projektgruppe *RCCARSng* mit der Entwicklungsphase nach dem ersten Review begonnen hatte, musste sich in den neuen Arbeitsprozess erst eingelebt werden. Oftmals konnten Deadlines nicht eingehalten werden, da die Aufgaben innerhalb der ersten Sprints stark unterschätzt wurden oder zu ungenau formuliert wurden. Dies besserte sich im Verlauf der Projektarbeit enorm.
Ein weiteres größeres Problem war, dass die Planung gewisser Bereiche wie beispielsweise der Schnittstellen und Architektur nicht in die Zeitplanung aufgenommen wurde. Durch notwendige ausführliche Besprechungen verzögerte sich der Entwicklungsprozess, wodurch der Zeitplan mehrmals innerhalb der Projektzeit angepasst werden musste. Besonders in der Zeit zwischen dem 1. und 2. Review wurde zudem das Testen der Komponenten stark vernachlässigt. Hier musste die Projektgruppe mit einer intensiven Testphase und Änderungen im Vorgehensmodell entgegenwirken. Durch die nebenläufigen Semester und Klausurenphasen kam es zeitweise zu einer Überbelastung der Mitglieder, wodurch die Arbeiten innerhalb der Projektgruppe vernachlässigt wurden.
Durch kontinuierliches Analysieren und Ausbessern dieser Probleme, konnte die Projektgruppe ein gutes Ergebnis bezüglich der gesetzten Projektziele erreichen.

Vorgehensmodell

Innerhalb der Projektgruppenarbeit wurde von der Projektleitung ein eigener Arbeitsprozess für die Realisierung der Projektziele erarbeitet. Dieser Prozess orientierte sich an dem Vorgehensmodell Scrum und wurde wie bereits beschrieben (siehe Abschnitt 3.2.2) für die Projektgruppe angepasst. Nach einer Eingewöhnungszeit durch die Gruppenmitglieder wurde der Prozess erfolgreich gelebt. Allerdings mussten auch gewisse Änderungen an dem Vorgehensmodell vorgenommen werden. Somit fand eine Weiterentwicklung der Arbeitsweise und eine Effektivitätssteigerung der Gruppenarbeit statt.
Aufgabenbereiche wie Projektleitung, Testbeauftragter oder Dokumentenbeauftragter die zu Beginn der Projektzeit innerhalb der Gruppe verteilt wurden, wurden von den Gruppenmitgliedern mit großer Eigenverantwortung über die gesamte Zeit durchgeführt. Zudem kristallisierten sich über die Entwicklungszeit Spezialisierungen der einzelnen Beteiligten heraus, die das Projektziel ermöglichten.
Das Vorgehensmodell der Projektgruppe *RCCARSng* ist im Großen und Ganzen als positiv bezüglich des erreichten Ergebnisses zu betrachten. Allerdings wäre es interessant, welche Auswirkungen ein klassisches Vorgehensmodell wie ein V-Modell auf den Projektverlauf gehabt hätte.

Gruppendynamik

Die Dynamik innerhalb der Projektgruppe entwickelte sich sehr positiv mit zunehmendem Projektverlauf. Am Anfang des Projekts traten die üblichen Schwierigkeiten auf. Dazu gehören fehlende Kommunikation und Missverständnisse zwischen den Gruppenmitgliedern. Doch dies verbesserte sich stark insbesondere durch die Vorbereitungszeit auf das 1. Review. Hierbei brachten sich alle Gruppenmitglieder verstärkt ein um ein gutes Ergebnis gemeinschaftlich präsentieren zu können. Auch das anschließende Boule-Turnier erwies sich als ein besonders prägendes Erlebnis.
Auch in den folgenden Monaten traf sich die Projektgruppe inklusive Betreuer zu teambildenden Maßnahmen. Dazu gehörte ein gemeinsamer Besuch auf dem Oldenburger Weihnachtsmarkt, sowie ein gemeinschaftlicher kulinarischer Abend.
Um die Disziplin innerhalb der Projektgruppe aufrecht zu erhalten entwickelte die Gruppe gemeinsam einen Strafenkatalog. Dieser sorgte immer für einen ausreichenden Keksvorrat im Projektraum.
Insgesamt waren alle Gruppenmitglieder sehr gerne an der Arbeit für das Projekt beteiligt und konnten durch das praktische Arbeiten viele Erfahrungen für das Berufsleben sammeln.

Danksagung

An dieser Stelle bedankt sich die Projektgruppe *RCCARSng* für die sehr gute Zusammenarbeit und Kooperation mit dem OFFIS, das Räumlichkeiten und sämtliche Hardware zur Verfügung gestellt hat. Ein besonderer Dank gilt Günter Ehmen, der zu jedem Zeitpunkt der Gruppe für Fragen und Hilfe zur Seite stand. Zusätzlich geht ein gesonderter Dank an Willem Hagemann, der durch seine wissbegierige und motivierende Art viele positive Impulse in der Projektgruppe setzten konnte. Ein weiteres Dankeschön geht an Stefan Puch, der durch seine Anmerkungen und Fragestellungen die Projektgruppe stets vorangebracht hat. Die Gruppe möchte sich bei allen drei Betreuern für die große Unterstützung und die reibungslose Zusammenarbeit bedanken. Außerdem bedankt sich die Projektgruppe bei Prof. Dr. Martin Fränzle für die Einführung in Regelungstechnik.
Zuletzt bedankt sich die gesamte Projektgruppe *RCCARSng* bei der Carl von Ossietzky Universität Oldenburg für das Ermöglichen eines solchen Projekts.

A. Systemtabellen

A.1. Netzwerkteilnehmer mit ID

ID	Sender
301 - 316	Control-Unit
200	Positionsbestimmung
500	Systemsteuerungssoftware

Tabelle A.1.: ID-Zuweisung der Sender im Netzwerk.

A.2. Zustände des Systems

ID	Zustand
0	Aus
1	Inspektion
2	Initialisierung
3	Standby
4	Rennbetrieb
5	Fehler

Tabelle A.2.: ID-Zuweisung Zustandsautomat der Systemsteuerungssoftware.

A.3. Fahrverhalten

ID	Fahrverhalten
0	Kein Fahrverhalten
1	Folgen anderer Fahrzeuge
2	Überholen
3	Überholt werden (passiv)

Tabelle A.3.: Kodierung der Fahrverhalten.

A.4. Control-Unit Modulübersicht

Modul-nummer	**Modul-nummer Hex**	**Modul**
1	0x01	Vorplanung
2	0x02	Kollisionserkennung
3	0x03	Kommunikationsinterface
16	0x10	Konfigurationsverarbeitung
17	0x11	Statusmanagement
18	0x12	Plausibilitätskontrolle
19	0x13	Watchdogs
20	0x14	Fehlermanagement
21	0x15	Realzeitüberprüfung
22	0x16	Renndatenberechnung
23	0x17	SerialControl
32-47	0x20 - 0x2F	Regelungsmodule

Tabelle A.4.: Kodierung der Control-Unit Modulnummern.

B. Konfigurationsparameter

B.1. Positionsbestimmung

Konfigurationsparameter				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
showDebug-Outputs	Flag zum Anzeigen von Debugausgaben	int	0, 1	—
showVideoStream	Flag Anzeigen von des VideoStreams	int	0, 1	—
printPoseToDisk	Flag zum Speichern der Fahrzeugposen	int	0, 1	—
poseCount	Anzahl der Posen die gespeichert werden	int	≥ 0	—
ignoreSSS	Flag zum Ignorieren der Systemsteuerungssoftware	int	0, 1	—
setupFilter	Flag zum anzeigen der Filtereinstellung	int	0, 1	—
allowedLostCars	Anzahl der erlaubten verlorenen Fahrzeugposen	int	≥ 0	—
sendToLocalhost-Only	Flag für die Netzwerkkommunikation	int	0, 1	—
receiveFrom-LocalhostOnly	Flag für die Netzwerkkommunikation	int	0, 1	—
MAX_THRES-HOLD_TYPE	Anzahl der verschiedenen Threshold Filter	int	≥ 0	—
MAX_THRES-HOLD_VALUE	Maximal Wert des Threshold Filters	int	≥ 0	—
MAX_BLUR_-VALUE	Maximal Wert des Blur Filters	int	≥ 0	—

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
DEFAULT_WINDOW_WIDTH	Maximale Fenster Breite	int	≥ 0	—
DEFAULT_WINDOW_HEIGHT	Maximale Fenster Höhe	int	≥ 0	—
thresholdType	Auswahl des Threshold Filters	int	≥ 0	—
lowerBound1	untere Grenze des 1. Filters	int	≥ 0	—
upperBound1	obere Grenze des 1. Filters	int	≥ 0	—
blurValue	Auswahl des Blurfilters	int	≥ 0	—
lowerBound2	untere Grenze des 2. Filters	int	≥ 0	—
upperBound2	obere Grenze des 2. Filters	int	≥ 0	—
fx	Brennweite der Kamera in x Richtung	double	≥ 0	—
fy	Brennweite der Kamera in y Richtung	double	≥ 0	—
cx	Hauptpunkt des Bildes (x-Koordinate)	double	≥ 0	—
cy	Hauptpunkt des Bildes (y-Koordinate)	double	≥ 0	—
k1	radialer Verzerrungskoeffizient 1. Ordnung	double	—	—
k2	radialer Verzerrungskoeffizient 2. Ordnung	double	—	—
p1	tangentialer Verzerrungskoeffizient 1. Ordnung	double	—	—
p2	tangentialer Verzerrungskoeffizient 2. Ordnung	double	—	—
k3	radialer Verzerrungskoeffizient 3. Ordnung	double	—	—

Eingabe	**Beschreibung**	**Datentyp**	**Werte-bereich**	**Ein-heit**
varI	Varianz Matrix zur Erkennung der ID-Marker	double [3][3]	≥ 0	—
erwI	Erwartungswert Vektor zur Erkennung der ID-Marker	double [3]	≥ 0	—
varH	Varianz Matrix zur Erkennung der Heck-Marker	double [3][3]	≥ 0	—
erwH	Erwartungswert Vektor zur Erkennung der Heck-Marker	double [3]	≥ 0	—
varF	Varianz Matrix zur Erkennung der Front-Marker	double [3][3]	≥ 0	—
erwF	Erwartungswert Vektor zur Erkennung der Front-Marker	double [3]	≥ 0	—
varE	Varianz Matrix zur Erkennung der Eck-Marker	double [3][3]	≥ 0	—
erwE	Erwartungswert Vektor zur Erkennung der Eck-Marker	double [3]	≥ 0	—
cornerPos	Koordinaten der Mittelpunkte der Eckmarker	double [8]	≥ 0	cm
cornerIndex	Orientierung der Eckmarker	int [4]	≥ 0	—

Tabelle B.1.: Konfigurationsparameter der Positionsbestimmung.

B.2. Control-Unit

Konfigurationsparameter				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
Vorplanungs-SourceFolder	Pfad zu Source Dateien der Vorplanung	String	-	—
MatlabSource-Folder	Pfad zu allgemeinen Matlab-Klassen	String	-	—
GurobiMatlab-Folder	Pfad zum Gurobi-Matlab Ordner	String	-	—
TrajPassiv	Dateiname der Trajektorie für „passives Fahren“	String	-	—
TrajOverFast	Dateiname der Trajektorie für „Überholen“	String	-	—
TrajOverSlow	Dateiname der Trajektorie für „überholt werden“	String	-	—
TrajectoryPath	Pfad zur berechneten Trajektorie	String	-	—
LoggingPath	Pfad zu den LogDateien	String	-	—
CUID	ID der Control-Unit	int	1 ... 16	—
FzID	ID des zu steuernden Fahrzeugs	int	0 ... 15	—
FzID2	ID des zweiten Fahrzeugs	int	0 ... 15	—
ignoreSS	Ignorieren der Systemsteuerungssoftware	int	0 ... 1	—
CollisionDetection-Enable	Ein- und Ausschalten der Kollisionserkennung	int	0 ... 1	—
BorderCollision-detectionEnable	Ein- und Ausschalten der Bandenkollisionserkennung	int	0 ... 1	—
ControlModul-Enable	Ein- und Ausschalten der Regelungsmodule	int	0 ... 1	—
PlausibilityCheck-Enable	Ein- und Ausschalten der Plausibilitätskontrolle	int	0 ... 1	—
PreplanningEnable	Ein- und Ausschalten der Vorplanung	int	0 ... 1	—

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
RacedataCalculationEnable	Ein- und Ausschalten der Renndatenberechnung	int	0 ... 1	—
RealTimeCheckEnable	Ein- und Ausschalten der Realzeitüberprüfung	int	0 ... 1	—
SerialControlEnable	Ein- und Ausschalten der seriellen Schnittstelle	int	0 ... 1	—
WatchdogEnable	Ein- und Ausschalten der Watchdogs	int	0 ... 1	—
amin	maximale Rückwärtsbeschleunigung	double	-2 ... -0.5	m/s^2
amax	maximale Vorwärtsbeschleunigung	double	0.8 ... 4	m/s^2
qmax	maximale Querbeschleunigung	double	0 ... 3	m/s^2
vmax	maximale Geschwindigkeit	double	0.0 ... 4.0	m/s
deltamin	minimaler Lenkwinkel	double	-25.0 ... 0.0	°
deltamax	maximaler Lenkwinkel	double	0.0 ... 25.0	°
Fahrzeugbreite	Fahrzeugbreite	double	0.01 ... 0.045	m
Fahrzeuglänge	Fahrzeugbreite	double	0.01 ... 0.15	m
SafetyDistCollisonDetection	Sicherheitsabstand für die Kollisionserkennung	double	0.0 ... 1.0	m
L	Radstand des Fahrzeugs	double	0.01 ... 0.1	m
c1	Gewichtsverteilung des Fahrzeugs (0.5 = Schwerpunkt in der Mitte)	double	0.4 ... 0.6	—
cm1	Motorparameter	double	0.1 ... 20.0	—

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
cm2	Motorparameter	double	0.1 ... 20.0	—
cr0	Reibungsparameter: Reifen ↔ Untergrund	double	0.1 ... 20.0	—
cr2	Reibungsparameter: Luftwiderstand	double	0.0 ... 20.0	—
SM	Koeffizient für die Umrechnung von Lenkwinkel zu Stellgröße	int	0 ... 4095	—
SB	Koeffizient für die Umrechnung von Lenkwinkel zu Stellgröße	int	0 ... 4095	—
TM1	Koeffizient für die Umrechnung von DutyCycle zu Stellgröße (Vorwärtsfahrt)	int	0 ... 4095	—
TB1	Koeffizient für die Umrechnung von DutyCycle zu Stellgröße (Vorwärtsfahrt)	int	0 ... 4095	—
TM2	Koeffizient für die Umrechnung von DutyCycle zu Stellgröße (Rückwärtsfahrt)	int	0 ... 4095	—
TM2	Koeffizient für die Umrechnung von DutyCycle zu Stellgröße (Rückwärtsfahrt)	int	0 ... 4095	—
NumberOf-PathIterations	Anzahl der Iterationen zur Pfadplanung	int	0 ... 255	—
NumberOf-SamplingPoints	Anzahl der Stützstellen zur Trajektorienberechnung	int16	0 ... 10000	—
PrintResolution	Auflösung bei Plots der Rennstrecke	int	0 ... 10000	—
SafetyDistance	Sicherheitsabstand für die Berechnung der Trajektorie	double	0.0 ... 1.0	m

Eingabe	Beschreibung	Datentyp	Wertebereich	Einheit
vAverage	Mindestdurchschnittsgeschwindigkeit für die Berechnung des Geschwindigkeitsprofils	double	0.0 ... 10	m/s
adaptVAverage-Enable	Flag zum An- und Ausschalten der Geschwindigkeitsanpassung	int	0 ... 1	—
showTrajMfigure	Flag zum An- und Ausschalten des Trajektorienplots der Vorplanung	int	0 ... 1	—
printVirtual-Obstacles	Flag zum Anzeigen von virtuellen Hindernissen in Plots	int	0 ... 1	—
lengthStraight-Segments	Länge Geradensegmente	double	0.0 ... 3.0	m
widthOutside-Centerline	Länge von Mittellinie zum Fahrbahnaußenrand	double	0.0 ... 3.0	m
widthBorder	Breite des Fahrbahnrandes	double	0.0 ... 3.0	m
radianCenterline-Curvesegments	Radius der Mittellinie in Kurven	double	0.0 ... 3.0	m
smallRadian	Radius des Kurvensegments am Fahrbahnrand	double	0.0 ... 3.0	m
xStartpoint	x-Koord. Startpunkt erstes Geradensegments vor der Mittellinie.	double	0.0 ... 10.0	m
yStartpoint	y-Koord. Startpunkt erstes Geradensegments vor der Mittellinie.	double	0.0 ... 10.0	m
roadsizeX	Länge der Rennstrecke in x-Richtung	double	0.0 ... 10.0	m
roadsizeY	Länge der Rennstrecke in y-Richtung	double	0.0 ... 10.0	m

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
FinishLineLength	Länge der Start-Ziel-Linie	double	0.0 ... 3.0	m
FinishLineWidth	Breite der Start-Ziel-Linie	double	0.0 ... 3.0	m
receiveFrom-LocalhostOnly	Einstellung, ob Pakete nur vom Localhost empfangen werden sollen	int	0,1	—
sendToLocal-hostOnly	Einstellung, ob Pakete nur an den Localhost gesendet werden sollen	int	0,1	—
broadcastPort	Port für den Broadcast	int	0 ... 65535	—
VehicleDataDelay	Zeit, die maximal zwischen den Fahrzeugdatenpaketen liegen darf	double	≥ 0	s
ControlDelay	Zeit, die maximal zwischen zwei aufeinander folgenden Regelungswerten liegen darf liegen darf	double	≥ 0	s
c_to_v_diff	Zeit, die maximal zwischen einlesen eines Bildes bis zum Aussenden der Stellgrößen vergehen darf	double	≥ 0	s
serialport	Serielle Schnittstelle zur CarControl	string	—	—
Initialization-mode	Modus der CarControl im Systemzustand “Initialisierung”	int	0 ... 4	—
Standbymode	Modus der CarControl im Systemzustand “Standby”	int	0 ... 4	—
Racemode	Modus der CarControl im Systemzustand “Rennbetrieb”	int	0 ... 4	—

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
onlyStoreForID	Einstellung für die Ringpuffer -> ignorieren von Paketen für andere IDs	int	0, 1	—
storeOwnPackages	Einstellung für die Ringpuffer -> speichern von eigenen Paketen	int	0, 1	—
storeNonCriticalError	Einstellung für die Ringpuffer -> Speichern von nicht kritischen Fehlern	int	0, 1	—
connectionCheckInterval	Intervall, in dem Verbindung zur CarControl überprüft werden soll. Bei "-1" wird nicht überprüft	double	-1, ≥ 0	ms
wv	Gewichtung der Geschwindigkeit	double	≥ 0	—
wphi	Gewichtung der Ausrichtung	double	≥ 0	—
wx	Gewichtung der Abweichung in Fahrtrichtung	double	≥ 0	—
wy	Gewichtung der Abweichung orthogonal Fahrtrichtung	double	≥ 0	—
ru1	Gewichtung des Lenkens	double	≥ 0	—
ru2	Gewichtung der Beschleunigung	double	≥ 0	—
latenz	Anzahl der möglichen Bilder, die die Berechnung hinterherhängt.	int	≥ 0	—
T	Schrittweite der MPC	double	≥ 0	s
N	Anzahl der Schritte, die die MPC berechnet	int	≥ 0	—

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
numberOfVgoalEntries	Anzahl der Einträge, die die Zielgeschwindigkeit nacheinander in der Trajektorie enthalten	int	≥ 0	—
Fv	Fahrverhalten	int	0, 1, 2	—
splitPoint0x	X-Koordinate des Punktes, an dem die Ausweichtrajektorie beginnen soll	double	0.0 ... 10.0	m
splitPoint0y	Y-Koordinate des Punktes, an dem die Ausweichtrajektorie beginnen soll	double	0.0 ... 10.0	m
splitPoint1x	X-Koordinate des Punktes, an dem die Ausweichtrajektorie enden soll	double	0.0 ... 10.0	m
splitPoint1y	Y-Koordinate des Punktes, an dem die Ausweichtrajektorie enden soll	double	0.0 ... 10.0	m

Tabelle B.2.: Konfigurationsparameter der Control-Unit.

B.3. Systemsteuerungssoftware

Konfigurationsparameter				
Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
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	-

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
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 TimedataPackage von der CU des jeweiligen Fahrzeugs ausgegeben.	double	≥ 0.0	-
keepAliveDelay	Konstante, welche festlegt nach wie viel Zeit (in Millisekunden) die SSSW das zuletzt von dieser ausgegebene Netzwerkdatenpaket - als keep alive Signal - wiederholt ausgeben soll.	long	≥ 10	-
maxInternalProcessingTime	Konstante, welche die interne Realzeitanforderung an die SSSW festlegt. Es wird definiert wie viel Zeit (in Millisekunden) ein Ausführungsdurchgang (Auswertung von Netzwerkdatenpaketen und Benutzereingaben usw.) der SystemControl-Klasse benötigen darf.	long	≥ 1	-

Eingabe	**Beschreibung**	**Datentyp**	**Wertebereich**	**Einheit**
maxInternalProcTimeViolation	Fehlertoleranzkonstante, welche festlegt in welchem Umfang die interne Realzeitanforderung der SSSW 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.	bool	0 / 1	-
logNetworkDataPackages	Speichert, ob das Logging von Netzwerkdatenpaketen aktiv ist oder nicht.	bool	0 / 1	-
minVel	Konfigurationsparameter zur Angabe der Mindestdurchschnittsgeschwindigkeit	double	0.0 ... 4.0	m/s
drivingBehavior	Konfigurationsparameter zur Angabe des Fahrverhaltens	int	0 ... 3	-
carCuLink	Konfigurationsparameter zur Angabe der Control-Unit zugewiesenen Fahrzeug ID	int	0 ... 15	-

Tabelle B.3.: Konfigurationsparameter der Systemsteuerung.

C. Fehlercodetabellen

C.1. Fehlercodes des Systems

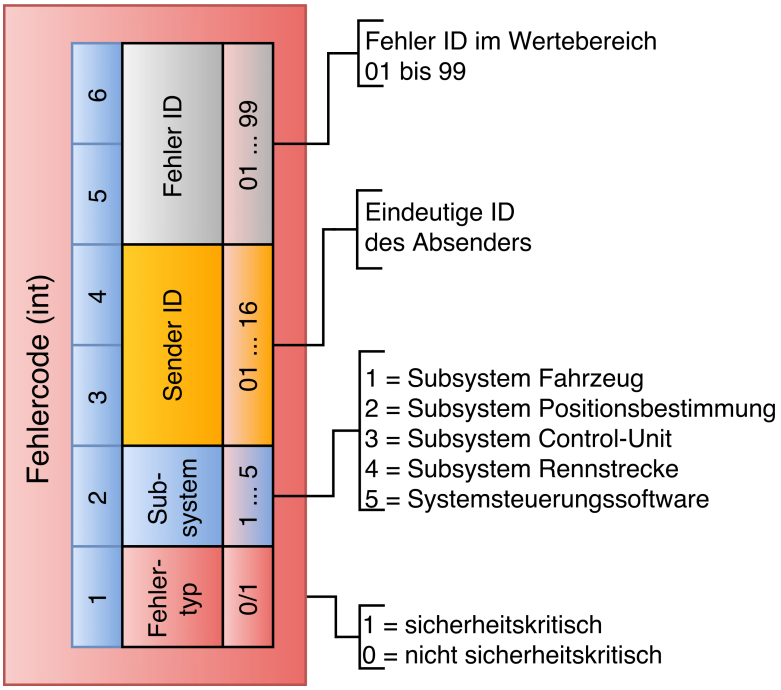


Abbildung C.1.: Aufbau des Fehlercodes.

Kritisch	Subsys.	Sender-ID	Fehler-ID	Beschreibung
0	2	00	00	Kamerainitialisierung fehlgeschlagen.*
0	2	00	01	Keine Balken oder Marker gefunden.*
0	2	00	02	Keine oder zu wenig Rennstreckenmarker gefunden.*
0	2	00	03	Zu viele Rennstreckenmarker gefunden.*
0	2	00	04	Anzahl der initialisierten Fahrzeuge stimmt nicht mit der Anzahl der gefundenen Balken überein.*

Kritisch	Subsys.	Sender-ID	Fehler-ID	Beschreibung
0	2	00	05	Realzeitanforderung konnte nicht eingehalten werden.*
1	2	00	06	Rennstrecke oder Gerüst wurden bewegt.*
0	2	00	07	Kamera liefert weniger als 100 fps.*
1	2	00	08	Positionsbestimmung wurde beendet.*
1	2	00	09	Fahrzeugpose seit mehreren Durchläufen nicht feststellbar.*
0	2	00 ... 15	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	2	00	13	Ein Hindernis wurde bewegt.
1	2	00	14	Zu viele Hindernisse gefunden (Anzahl > 16).
1	2	00 ... 15	15	Fahrzeug mit der ID mehrfach gefunden. dies deutet auf Fehler in der Ausleuchtung hin
1	2	00	16	Es wurde ein Hindernis während des Rennbetriebes hinzugefügt
1	2	00	17	Es wurde ein Hindernis während des Rennbetriebes entfernt
1	2	00	18	Es wurde ein Fahrzeug während des Rennbetriebes hinzugefügt
1	3	01 ... 16	01	Kritischer Fehler in der Vorplanung
0	3	01 ... 16	01	Fehler in der Vorplanung
1	3	01 ... 16	02	Kritischer Fehler in der Kollisionserkennung
0	3	01 ... 16	02	Fehler in der Kollisionserkennung
1	3	01 ... 16	17	Kritischer Fehler in dem Statusmanagement
0	3	01 ... 16	17	Fehler in dem Statusmanagement

Kritisch	Subsys.	Sender-ID	Fehler-ID	Beschreibung
1	3	01 ... 16	18	Kritischer Fehler in der Plausibilitätskontrolle
0	3	01 ... 16	18	Fehler in der Plausibilitätskontrolle
1	3	01 ... 16	21	Kritischer Fehler in der Realzeitüberprüfung
0	3	01 ... 16	21	Fehler in der Realzeitüberprüfung
1	3	01 ... 16	22	Kritischer Fehler in der Renndatenberechnung
0	3	01 ... 16	22	Fehler in der Renndatenberechnung
1	3	01 ... 16	23	Kritischer Fehler in der SerialControl
0	3	01 ... 16	23	Fehler in der SerialControl
1	3	01 ... 16	42	Kritischer Fehler in dem Regelungsmodul
0	3	01 ... 16	42	Fehler in dem Regelungsmodul
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.*
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).*

Kritisch	Subsys.	Sender-ID	Fehler-ID	Beschreibung
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.*
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.*

Kritisch	Subsys.	Sender-ID	Fehler-ID	Beschreibung
1	5	00	71	Liste der von der Positionsbestimmungs-software 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 C.1.: Fehlercodetabelle für das System.

C.2. Control-Unit interne Fehlercodes

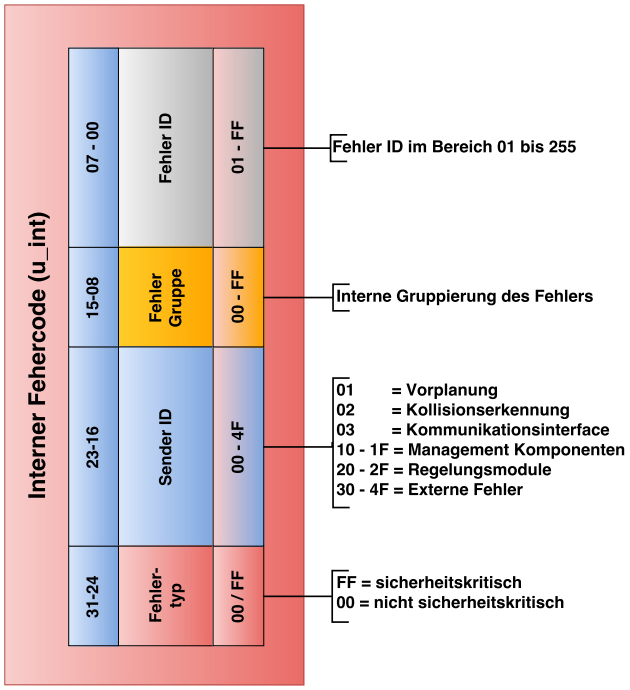


Abbildung C.2.: Aufbau der Control-Unit internen Fehlercodes.

Kritisch	Sender Id	Gruppe	Fehler-ID	Beschreibung
0xFF	0x01	0x00	0x01	Nicht identifizierbarer Matlab-Fehler.
0xFF	0x01	0x01	–	**Verbindungsfehler**
0xFF	0x01	0x01	0x01	Verbindungsversuch zur Matlab-Engine fehlgeschlagen.
0xFF	0x01	0x01	0x02	Empfange statische Umgebungsdaten sind keine Matrix mit acht Spalten.
0xFF	0x01	0x01	0x03	Informationszeile der statischen Umgebungsdaten enthält nichtnumerische Werte..
0xFF	0x01	0x01	0x04	Empfange Vorplanungsparameter ist kein Vektor.
0xFF	0x01	0x01	0x05	Vorplanungsparameter enthält nichtnumerischen Wert.
0xFF	0x01	0x02	–	**Fehler in der Trajektorienberechnung**
0xFF	0x01	0x02	0x01	Anzahl der Iterationen kleiner 0.

Kritisch	Sender Id	Gruppe	Fehler-ID	Beschreibung
0xFF	0x01	0x02	0x02	Gurobi hat keine Lösung gefunden. Konnte ggf. Sicherheitsabstand nicht einhalten.
0xFF	0x01	0x02	0x03	Maximal geplante Durchschnittsgeschwindigkeit kleiner als gewünschte.
0xFF	0x01	0x02	0x04	Krümmung der Trajektorie an einer Stützstelle zu groß.
0xFF	0x01	0xFF	0x01	Watchdog: Zeit abgelaufen.
0xFF	0x02	0x01	0x01	Keine gültigen Fahrzeugdaten verfügbar.
0xFF	0x02	0x02	i	Kollision mit Fahrzeug i aufgetreten.
0xFF	0x02	0x03	i	Kollision mit Hindernis i aufgetreten.
0xFF	0x02	0x04	0x01	Kollision mit Fahrbahnrand aufgetreten.
0xFF	0x02	0xFF	0x01	Watchdog: Zeit abgelaufen.
0xFF	0x10	0x00	0x01	Es wird versucht auf einen Parameter als String zuzugreifen, der keiner ist.
0xFF	0x10	0x00	0x02	Es wird versucht auf einen Parameter als Integer zuzugreifen, der keiner ist.
0xFF	0x10	0x00	0x03	Es wird versucht auf einen Parameter als Double zuzugreifen, der keiner ist.
0xFF	0x10	0x01	0x01	Es wird versucht auf Konfigurationsdaten zuzugreifen, obwohl sie noch nicht eingelesen wurden.
0xFF	0x10	0x01	0x02	Es wird versucht auf Hindernisdaten zuzugreifen, obwohl sie noch nicht eingelesen wurden.
0xFF	0x11	0x00	0x01	Das System hat versucht, einen ungültigen Zustandswechsel zu vollziehen.
0xFF	0x12	0x01	0x01	Configfile falsch eingelesen.
0xFF	0x12	0x02	–	**Fehler in den statischen Umgebungsdaten**
0xFF	0x12	0x02	0x01	Statische Umgebungsdaten enthalten Werte kleiner -1.
0xFF	0x12	0x02	0x02	Fehlerhafte Infozeile.
0xFF	0x12	0x02	0x03	Fehlerhafter Rennstreckentyp in Fahrbahndaten.

Kritisch	Sender Id	Gruppe	Fehler-ID	Beschreibung
0xFF	0x12	0x02	0x04	Geradensegment nicht schnittstellenkonform oder außerhalb der Rennstrecke.
0xFF	0x12	0x02	0x05	Kurvensegment nicht schnittstellenkonform.
0xFF	0x12	0x02	0x06	Fahrbahnrand nicht schnittstellenkonform.
0xFF	0x12	0x02	0x07	Hinderniskoordinaten außerhalb der Rennstrecke.
0xFF	0x12	0x02	0x08	Leere Hinderniszeilen nicht mit -1 initialisiert.
0xFF	0x12	0x03	0x01	Fahrzeugdaten nicht schnittstellenkonform.
0xFF	0x12	0x03	0x02	Fahrzeugdaten nicht von Subsystem 2 (Positionsbestimmung) empfangen.
0x00	0x12	0x03	0x03	Fahrzeug nach Fahrzeugdaten schneller als maximal mögliche Geschwindigkeit.
0xFF	0x12	0x03	0x04	Timestamps der Fahrzeugdaten nicht monoton steigend.
0xFF	0x12	0x04	0x01	Empfangene Mindestdurchschnittsgeschwindigkeit größer als maximal mögliche Geschwindigkeit.
0xFF	0x12	0x04	0x02	Zugewiesenes Fahrzeug nicht im Bereich 0 - 15.
0xFF	0x12	0x04	0x03	Timestamps der Fahrzeugdaten nicht monoton steigend.
0xFF	0x12	0xFF	0x01	Watchdog: Zeit abgelaufen.
0xFF	0x15	0x00	0x01	Es wurden keine neuen Fahrzeugdaten in dem vorgegebenen Zeitintervall empfangen
0xFF	0x15	0x01	0x01	Die Regelungsdaten wurden nicht in dem vorgegebenen Intervall berechnet.
0xFF	0x15	0x01	0x02	Regelungsdaten wurden in falscher Reihenfolge empfangen.

Kritisch	Sender Id	Gruppe	Fehler-ID	Beschreibung
0xFF	0x15	0xFF	0x03	Die Dauer der vom Empfangen des Bildes bis zum Aussenden der Regelungswerte überschritt den vorgegebenen Wert.
0xFF	0x17	0x00	0x01	Verbindung zur CarControl konnte nicht aufgebaut werden.
0xFF	0x17	0x00	0x02	Verbindung zur CarControl verloren.
0xFF	0x17	0x00	0x03	Verbindung zur CarControl konnte nicht geschlossen werden.
0xFF	0x17	0x00	0x04	CarControl Reagiert nicht -> falsche Schnittstelle in der Konfigurationsdatei.
0xFF	0x17	0x01	0x01	Nicht definierter Moduswechsel.
0xFF	0x17	0x01	0x02	Moduswechsel konnte nicht gesendet werden.
0x00	0x17	0x01	0x03	Steering-Wert out of Bounds -> korrigiert.
0x00	0x17	0x01	0x04	Throttle-Wert out of Bounds -> korrigiert.
0xFF	0x17	0x01	0x05	Steering-Wert konnte nicht gesendet werden.
0xFF	0x17	0x01	0x06	Throttle-Wert konnte nicht gesendet werden.
0xFF	0x17	0x01	0x07	Steering- und Throttle-Wert konnte nicht gesendet werden.
0xFF	0x17	0x01	0x08	Reset-Signal konnte nicht gesendet werden.
0xFF	0x17	0x01	0x09	NotAus-Signal konnte nicht gesendet werden.
0x00	0x17	0x01	0x10	IR-LED Toggle konnte nicht gesendet werden.
0xFF	0x17	0x01	0x11	Connection-Signal konnte nicht gesendet werden.
0xFF	0x17	0x01	0x12	StartHWWatchdog-Signal konnte nicht gesendet werden.

Kritisch	Sender Id	Gruppe	Fehler-ID	Beschreibung
0xFF	0x17	0x01	0x13	StopHWWatchdog-Signal konnte nicht gesendet werden.
0xFF	0x17	0x02	0x01	Connect-Antwort nicht empfangen.
0xFF	0x17	0x02	0x02	Reset-Antwort nicht empfangen.
0xFF	0x17	0x02	0x03	NotAus-Antwort nicht empfangen.
0x00	0x17	0x02	0x04	Kein Feedback von der CarControl empfangen.
0x00	0x17	0x02	0x05	Kein Feedback von der CarControl gesendet.
0xFF	0x17	0x02	0x06	Moduswechsel-Antwort nicht empfangen.
0xFF	0x17	0x02	0x07	WatchdogStart-Antwort nicht empfangen.
0xFF	0x17	0x02	0x08	WatchdogStop-Antwort nicht empfangen.
0xFF	0x17	0x03	0x01	NotAus-Taster auf der CarControl gedrückt.
0xFF	0x17	0x03	0x02	Reset-Taster auf der CarControl gedrückt.
0xFF	0x17	0x03	0x03	NotAus von der CarControl empfangen.
0xFF	0x2a	0x00	0x01	Trajektorie nicht geladen.
0xFF	0x2a	0x00	0x02	kein Fahrverhalten.
0xFF	0x2a	0x01	0x02	zu lange keine Fahrzeugdaten für die Regelung.
0x00	0x2a	0x01	0x01	Modell durch Relaxion gelöst.
0x00	0x2a	0x01	0x03	keine Validen Fahrzeugdaten.

Tabelle C.2.: interne Fehlercodetabelle.

D. Zeitplan

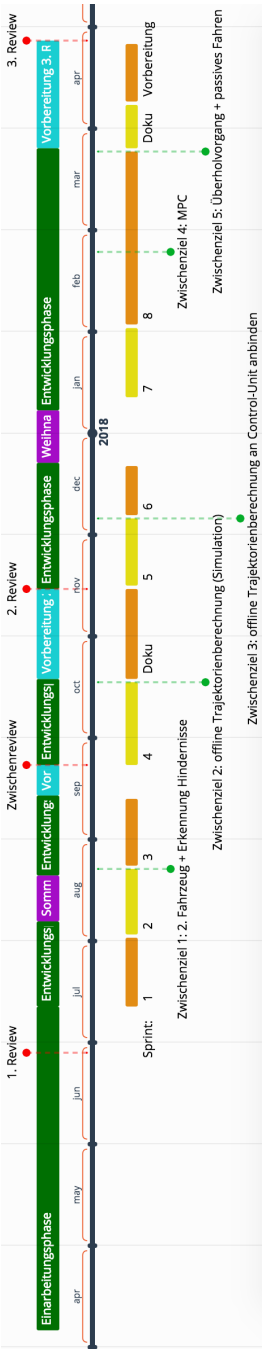
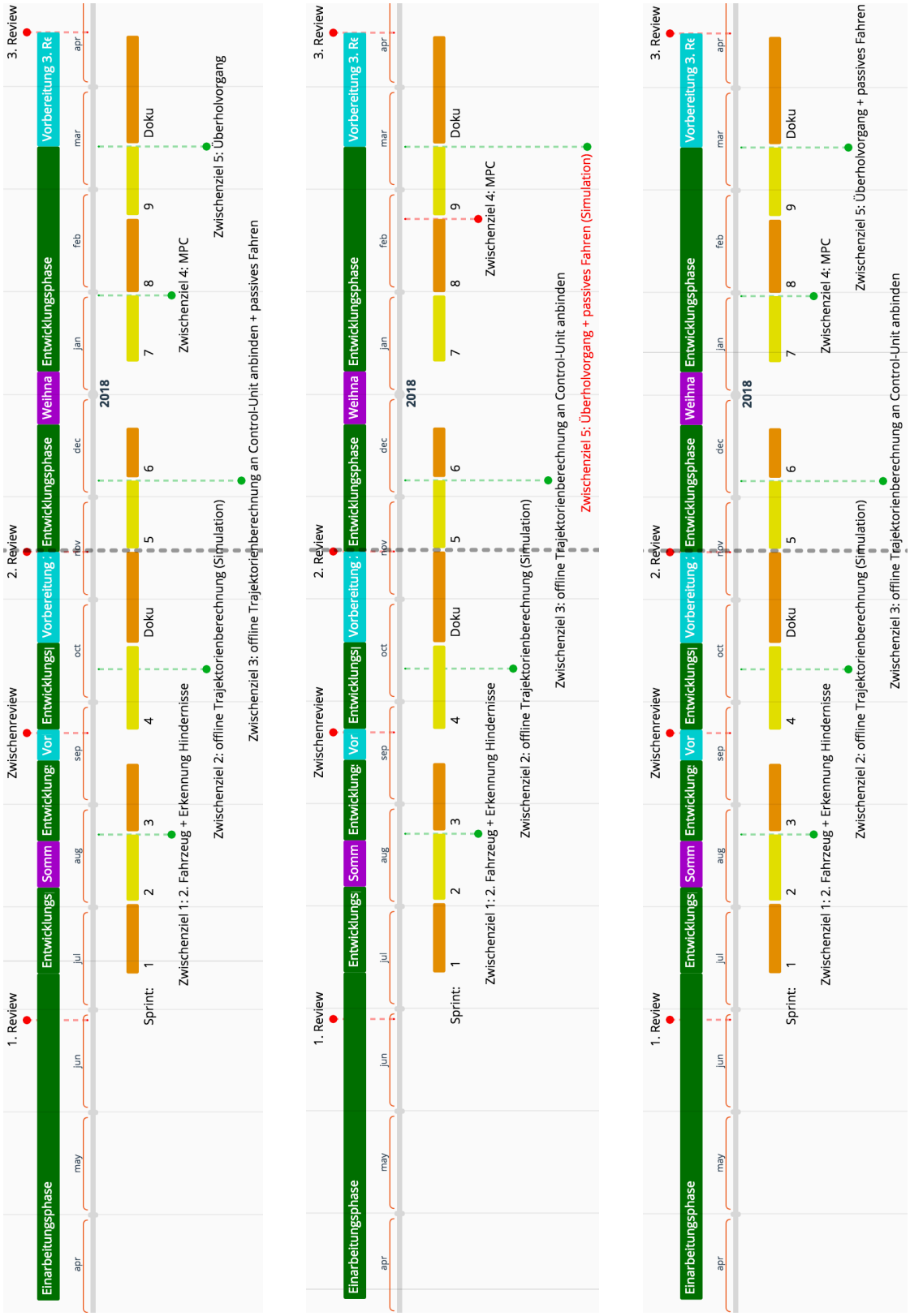


Abbildung D.1.: Grafische Darstellung der finalen Zeitplanung.



(a) Vollständige Zeitplanung. (b) Alternativen Zeitplanung. (c) Angepasste Zeitplanung.

Abbildung D.2.: Grafische Darstellung der Zeitplanung.

E. Risikokatalog

Die folgende Tabelle listet alle identifizierten Risiken der Projektgruppe auf.

Risiko	**Maßnahmen (präventiv)**	**Maßnahmen (korrektiv)**	**Maßzahl**
Motivation und Klima im Team ist schlecht.	Sprint Retrospektive zum Besprechen teaminterner Probleme	Persönliches Gespräch mit Betroffenen. Ggf. Unterstützung durch Betreuer.	2
Aufgabe nach DoD nicht erfüllt bei Sprintreview.	Schätzen der Aufgaben zur frühzeitigen Identifizierung von mangelnder Detaillierung oder fehlendem Verständnis. Erinnerungsmails können durch Agantty verschickt werden.	Aufgabe muss zurück in Product Backlog, bzw. in den neuen Sprint. Neue Schätzung, höhere Priorität, Besprechung im Team.	2
Hardware fällt aus.	Wenn möglich, ein weiteres Ersatzteil vorrätig. Softwareteile im GIT einchecken.	Hardware schnellstmöglich durch Ersatzteil ersetzen.	2
Szenario kann zum Ende des Projektes nicht erfüllt werden.	Risikozeitpläne, wie bspw. D.2(b), wurden erarbeitet, um Szenario so weit wie möglich zu erfüllen durch weniger Funktionalität.	Projektschnittstellen so weit wie möglich für Folgegruppen vorbereiten.	6

Risiko	**Maßnahmen (präventiv)**	**Maßnahmen (korrektiv)**	**Maßzahl**
Sprintzeitraum zu gering, um eindeutige Fortschritte zu erarbeiten und zu testen.	Sprintreview zur Kontrolle des Sprintverlaufs. Retrospektive, um allgemeines Empfinden zu evaluieren. Keine Aufgaben aus Product Backlog in Sprint Backlog, während eines Sprints verhindert Überlagerung.	Sprintzeitraum verlängern. Zeitplanung inklusive Zwischenziele ggf. durch Risikozeitplan anpassen.	3
Gesetzte Zwischenziele und Dokumentationen werden zu einem Review nicht erreicht.	Zeitpuffer: Letzte Entwicklungsphase endet 3 Wochen vor 2. Review. Letzte Entwicklungsphase endet vier Wochen vor 3. Review.	Team wird aufgeteilt in Entwicklung (notwendige Implementierungen zur Vorstellung) und Vorbereitung (Anpassung Dokumentation, Präsentation für Review).	4
Projektmitglied fällt über längeren Zeitraum aus.	Expertenwissen wird immer auf Teams mit einer Größe von mindestens zwei Personen aufgeteilt	Zweites Mitglied muss nötige Arbeit des ausgefallenen Mitglieds übernehmen.	2
Ein Task wird nicht innerhalb der zwei Wochen Entwicklungszeit fertiggestellt.	Während der Sprintplanung wird die Komplexität des Task geschätzt.	Testmethoden werden so weit wie möglich erstellt. Task muss in nächsten Sprint mit aufgenommen werden.	3
Ein Task wird nicht innerhalb der Testwoche fertig getestet.	Tasks müssen so unterteilt sein, dass Entwicklung und das Testen nicht den zeitlichen Rahmen sprengen.	Tests werden im nächsten Sprint schon in der Entwicklungsphase nachgeholt.	2
Tests eines Tasks schlagen fehl.	Testfälle müssen vor der Testwoche detailliert und verständlich in die Test Datenbank aufgenommen sein.	Task muss wieder in den nächsten Sprint mit aufgenommen werden.	2

Risiko	**Maßnahmen (präventiv)**	**Maßnahmen (korrektiv)**	**Maß-zahl**
Zwischenziel 3 wird nicht innerhalb eines Sprints realisiert.	Risikozeitplan D.2(b) wurde mit angepassten Zwischenzielen und Zeiten erarbeitet. Arbeitspakete müssen komplett und detailliert definiert und geschätzt werden.	Zwischenziele und Zeiten werden anhand des Risikozeitplans nach dem ersten Sprint nach dem 2. Review evaluiert und angepasst.	4
Zur Realisierung des Szenarios und somit Zwischenziels 5 ist zum Ende zu wenig Zeit.	Zeitplanung sowie alternative Zeitplanung wird laufend evaluiert und ggf. angepasst.	Zwischenziel 5 "Überholvorgang" wird innerhalb des letzten Sprints nur in der Simulation realisiert. Bei überschaubaren Rückstand kann die Entwicklungszeit verlängert werden und die Vorbereitungszeit (vier Wochen) auf das 3. Review gekürzt werden.	5

Tabelle E.1.: Risikokatalog.

F. Betriebs- und Wartungshandbuch

F.1. Allgemeine Hinweise

Die unterschiedlichen Softwarekomponenten werden auf verschiedenen Rechnern ausgeführt. Alle verwendeten Rechner müssen betriebsbereit sein und die Hardwarekomponenten ordnungsgemäß angeschlossen sein. Folgende Schritte sind zur Inbetriebnahme des Systems zu erledigen:

- Alle Systemrechner müssen hochgefahren und betriebsbereit werden.
- Die USB-Verbindung zur Kamera muss hergestellt werden.
- Netzstecker der USB-Verlängerung muss angeschlossen werden.
- IR-Strahler müssen mit Spannung versorgt werden.
- Systemsteuerung auf dem SSSW-Rechner muss gestartet werden.
- Positionsbestimmung auf dem PB-Rechner muss gestartet werden.
- Control-Units auf den CU-Rechnern müssen gestartet werden.

Folgende Schritte sind zum Abschalten des Systems zu erledigen:

- Systemsteuerungssoftware beenden.
- Control-Unit-Software beenden.
- Positionsbestimmung beenden.
- Verwendete Hilfssoftware schließen.
- Netzstecker der Kamera-USB-Verlängerung abziehen.
- Spannungsversorgung zu den IR-Strahlern trennen.
- Rechner herunterfahren.

F.2. Positionsbestimmung

Zum Starten der Positionsbestimmung müssen folgende Schritte durchgeführt werden:

- Sonneneinstrahlung auf die Rennstrecke verhindern.
- USB-Verbindung zur Kamera herstellen.
- Betriebsmodus der Kamera überprüfen. (Grüne Lampe auf der Kamera sollte leuchten.)
- Fahrzeuge in initiale Position auf der Rennstrecke platzieren.
- Eventuelle Hindernisse auf der Rennstrecke platzieren.
- Software der Positionsbestimmung starten.
  - Software über Konsole starten.
  - Die Konfigurationsdatei „positionsbestimmung.config“ muss sich im selben Verzeichnis befinden.

F.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 wie viele 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 Konfigurationsdatei stehen, momentan 53, mit dem Flycapture2 Programm aufnehmen. Dabei muss das Schachbrettmuster in verschiedenen Positionen und Winkeln aufgenommen werden.
- Die neu aufgenommenen Bilder mit „XnView“ umbenennen.
- Im Debug Ordner befindet sich die Datei „images.xml“. Dort müssen die Bilder alle eingetragen werden.
- CameraCalibration Programm starten. Hier könnte einige Wartezeit auftreten.

- 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 Datei „output.xml“ stehen dann die Distortionsparameter und die Kameramatrix. Diese Parameter in die Datei „positionsbestimmung.config“ übernehmen.
- Den Rest übernimmt die Positionsbestimmung selbst.

F.2.2. Konfiguration

Die Positionsbestimmung ermöglicht viele Konfigurationsmöglichkeiten. Alle möglichen Einstellungen sind über die Konfigurationsdatei einzustellen. Folgend sind einige Einstellungsmöglichkeiten aufgeführt.
Um Einstellungen an den Filtern vorzunehmen muss in der Konfigurationsdatei die Variable „setupFilter“ auf „1“ gesetzt werden. Die eingestellten Filterwerte müssen für den nächsten Systemstart per Hand an den entsprechenden Stellen in der Datei angepasst werden, wenn diese gespeichert werden sollen. Mit „k“ können die Filterwerte für einen Systemstart übernommen werden.

$$setupFilter = 1 \rightarrow ignoreSSS = 1.$$

Ist „setupFilter“ gesetzt, dann ist auch „ignoreSSS“ gesetzt.
Die Variable „ignoreSSS“ zeigt an, ob die Systemsteuerungssoftware ignoriert werden soll.
Die Variable „showVideoStream“ zeigt an, ob der Videostream angezeigt werden soll. Dies kann im laufenden System, ohne Systemsteuerungssoftware, mit der Taste „s“ an- und ausgeschaltet werden. Sobald der Videostream angezeigt wird, können die Realzeitanforderungen nicht mehr eingehalten werden.
Debugausgaben können über die Variable „showDebugoutputs“ eingeschaltet werden. Dies kann im laufenden System, ohne Systemsteuerungssoftware, mit der Taste „d“ an- und ausgeschaltet werden.

F.3. Control-Unit

Die Initialisierung der Control-Unit unterteilt sich in zwei Abschnitte. Zum Einen die Initialisierung der CarControl und zum Anderen das Starten der Control-Unit Software.

F.3.1. CarControl

- Herstellen der USB-Verbindung zwischen dem Rechner und der CarControl.
  - Überprüfen, welche Schnittstelle welchen Namen erhält.
- Verbindung zwischen Fernbedienung und Fahrzeug herstellen.
- Debugausgaben können über die dritte Schnittstelle ausgegeben.
  - Debugausgaben können über das Tool „hterm“ ausgelesen werden.
  - Hierbei muss die Schnittstelle (z.B. „/dev/tty/USB0“) ausgewählt werden.
  - Für empfangene bzw. gesendete Nachrichten muss das Feld „Newline at“ auf das Trennzeichen „CR+LF“ gesetzt werden.
  - Hierbei gibt es verschiedene Loggingstufen.

    „A“ Starten aller Debugausgaben

    „B“ Starten einfacher Debugausgaben

    „C“ Starten ausführlicher Debugausgaben
  - Ausschalten der Debugausgaben geschieht über „a“, „b“ oder „c“.
- Der Betriebsmodus der CarControl kann über den blauen Taster auf dem Entwicklungsboard gewechselt werden.
- Ein HardReset kann über den schwarzen Taster auf dem Entwicklungsboard ausgelöst werden. Dieser startet das Board neu.
- Der rote Taster auf der Platine überführt die CarControl in den sicheren Fehlerzustand.
- Der schwarze Taster auf der Platine führt einen SoftReset durch.

F.3.2. Control-Unit Software

Die Control-Unit Software wird über die Konsole gestartet. Hierbei ist zu beachten, das die Konfigurationsdatei im selben Verzeichnis liegt. Des Weiteren müssen alle Pfade in der Konfigurationsdatei korrekt sein. In der Konfigurationsdatei lassen sich verschiedene Komponenten an- und ausschalten. Diese Einstellungsmöglichkeiten sind in Tabelle B.2 zu sehen.

F.4. Systemsteuerungssoftware

- Die Systemsteuerungssoftware über die Konsole starten.
  - Befehl zum Starten: „java -jar SSSW.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.
- Erwartete Anzahl der am Rennen teilnehmenden Fahrzeuge eingeben. Danach über den Button „Confirm number“ bestätigen.
- Anschließend können die Konfigurationsparameter für die jeweiligen Fahrzeuge eingestellt werden.
- Manuelle Inspektion durchführen:
  - Einhaltung aller Rahmenbedingungen gemäß Rahmenbedingungen (siehe Abschnitt 7.1) kontrollieren.
  - Positionsbestimmung starten.
  - Control-Unit starten und CarControl auf vollständig autonomes Fahren stellen.
- 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. Der Standby-Zustand wird nach erfolgreicher Initialisierung automatisch erreicht.
- Im Standby-Zustand über den Button „Start Race“ den Rennbetrieb starten.
- Der Systembetrieb kann unabhängig vom Systemzustand jederzeit beendet werden indem die Systemsteuerungssoftware geschlossen wird. Beim Beenden kommt es abhängig von den gewählten Systemparametern zu einer kurzen Verzögerung, da die Systemsteuerungssoftware den anderen Systemkomponenten wiederholt mitteilt, dass der Systembetrieb beendet wird.
- Der Button „EMERGENCY STOP“ muss unverzüglich betätigt werden, 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.
- Daten über das aktuelle Renngeschehen können im „Race Data“-Tab abgelesen werden.

Glossar

Black-Box-Test Beschreibt das Testen einer Software ohne Kenntnis über die innere Funktion des zu testenden Systems. Dieser Test beschränkt sich auf die Funktionsüberprüfung.

Chassis Fahrgestell des Fahrzeugs.

Constraint Einschränkung eines Lösungsraumes.

Control-Unit Kontrolleinheit.

Duty-Cycle Auslastungsgrad eines Elektromotors.

Exzentrizität Maß für die Abweichung einer Ellipse von der Kreisform.

Fahrverhalten Reaktion auf Objekte, die auf der Trajektorie liegen.

Fahrzeugmodell Mathematische Beschreibung des Verhaltens eines Fahrzeuges.

Fehlerzustand Zustand, in den das System bei einem Fehler gebracht wird. Fahrzeug hält in diesem Zustand an.

Geschwindigkeitsprofil Beschreibt Soll-Geschwindigkeit und Soll-Beschleunigung an einer Stützstelle der Trajektorie.

GUI Graphisches User Interface.

Hindernis Quader aus Schaumstoff, welcher von der Positionsbestimmung erkannt wird.

HW Hardware.

kollisionsfrei Ohne Unfälle.

Kontrollsignale Gas-, Brems- und Lenkbefehle.

kritischer Abstand Wenn sich in diesem Abstand ein Objekt befindet, muss das Fahrzeug anhalten.

Lineare Programmierung Optimierung einer linearen Zielfunktion.

Mindestdurchschnittsgeschwindigkeit Beschreibt die Geschwindigkeit, die ein Fahrzeug mindestens erreichen soll.

offline Vor der Ausführung, nicht zur Laufzeit, generiert/ausgeführt.

online Während der Ausführung/der Laufzeit.

Optimierungsproblem Das zu optimierende Kriterium (Zielfunktion) einer modellprädikativen Regelung.

Pose Bezeichnet die räumliche Lage eines Objektes aus Kombination von Position und Orientierung.

Prädiktion Prognose, Vorhersage, Vorausschauen.

Prädiktionshorizont Beschreibt den zeitlichen Horizont, der für eine Prädiktion (zum Beispiel für das Verhalten eines Fahrzeugs) herangezogen wird.

Referenztrajektorie Bahnkurve (Weg/Pfad), der ein bestimmtes Objekt folgen soll.

Relaxierung Lösen eines Optimierungsproblems ohne bestimmte Constraints.

Road Software-Objekt, welches die Fahrbahn mit eventuellen Hindernissen darstellt. Liefert Funktionen zur Ermittlung der Umgebungsinformationen.

SCADE Programm zur modellgetriebenen Entwicklung von sicherheitskritischen Softwaresystemen.

Spline Kurve durch Stützpunkte, die aus mehreren Polynomen n-ten Grades zusammengesetzt ist.

Subsystem Einzelne Komponenten des gesamten Systems, bestehend aus Fahrzeug, Positionsbestimmung, Control-Unit, Rennstrecke und Systemsteuerungssoftware.

SW Software.

System Gesamtheit der Komponenten.

Trajektorie Linie, die das Fahrzeug fahren soll. Beinhaltet ein Geschwindigkeitsprofil.

Unfall Kollision mit Fahrzeugen, Hindernissen und der Fahrbahnbegrenzung.

White-Box-Test Beschreibt das Testen einer Software mit Kenntnis über die innere Funktion des zu testenden Systems.

Überholvorgang Beschreibt das Umfahren von Hindernissen und das Überholen von Fahrzeugen.

Toolverzeichnis

[Atl] Atlassian: *Trello*. https://trello.com/

[D'E] D'Errico, John: *interparc*. https://de.mathworks.com/matlabcentral/fileexchange/34874-interparc

[ESR] Eastern Switzerland Rapperswil, Institute for Software University of Applied Sciences o.: *CUTE*. http://cute-test.com/

[Est] Esterel Technologies SCADE: *SCADE*. http://www.esterel-technologies.com/products/scade-suite/

[FIIS] FLIR Integrated Imaging Solutions, Inc.: *FlyCapture SDK Point Grey*. https://www.ptgrey.com/flycapture-sdk. – Zugiff am 17.06.2017

[Fou] Foundation, The E.: *Eclipse*. https://www.eclipse.org/

[Gur] Gurobi GmbH: *GUROBI OPTIMATION*. http://www.gurobi.com/

[Hee] Heesch, Dimitri van: *Doxygen*. http://www.stack.nl/~dimitri/doxygen/

[jmo] *JModelica*. http://www.jmodelica.org/

[Mat] MathWorks®: *MATLAB*. https://de.mathworks.com/products/matlab.html

[med] media herrlich: *Agantty*. https://www.agantty.com/

[Mica] Microsoft: *Microsoft Excel*. https://products.office.com/de-de/excel

[Micb] Microsoft: *Microsoft Powerpoint*. https://products.office.com/de-de/powerpoint

[Mus] Mustfinn, Eugene: *Time.Graphics*. https://time.graphics

[SC16] Sanderson, Conrad ; Curtin, Ryan: Armadillo: a template-based C++ library for linear algebra. In: *Journal of Open Source Software* 1 (2016), S. 26

[Teaa] Team, LaTeX P.: *LaTeX*. https://www.latex-project.org/

[teab] team, OpenCV: *OpenCV*. – Zugiff am 17.06.2017

[Tor] Torvalds, Linus: *Git*. https://git-scm.com/

Literaturverzeichnis

[BCMS08] Braghin, F. ; Cheli, F. ; Melzi, S. ; Sabbioni, E.: Race Driver Model. In: *Computers & Structures* 86 (2008), S. 1503–1516

[Bun] Bundesregierung: *Automatisiertes Fahren auf dem Weg*. https://www.bundesregierung.de/Content/DE/Artikel/2017/01/2017-01-25-automatisiertes-fahren.html. – Eingesehen am 12.06.2017, 15:31

[Fon14] Fontana, Giorgio: *AUTONOMOUS DRIVING OF MINIATURE RACE CARS*. 2014

[HPA14] Harichandan, Suchismita ; Panda, Namita ; Acharya, Arup A.: *Scrum Testing With Backlog Management in Agile Development Environment*. 2014

[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

[LDM15] Liniger, Alexander ; Domahidi, Alexander ; Morari, Manfred: Optimization-based autonomous racing of 1:43 scale RC cars. In: *Optimal Control Applications and Methods* 36 (2015), S. 628–647

[lsq] *Methode der kleinsten Quadrate*. https://de.wikipedia.org/wiki/Methode_der_kleinsten_Quadrate, . – [Eingesehen am 03.04.2018]

[NES+16] Nordhamn, Amanda ; Eriksson, Emil ; Setterquist, Erik ; Teerikoski, Sakari ; Olofssoni, Simon: *Replacing Electronics and Extending Control*. 2016

[RCC16] RCCARS, Projektgruppe: Endbericht / Carl von Ossietzky Universität Oldenburg. 2016. – Forschungsbericht

[Roj03] Rojas, Raul: The Kalman Filter. (2003)

[Rut10] Rutschmann, Martin: *Infrared based vision system for tracking 1:43 scale race cars*. 2010

[SCR] *SCRUM Schaubild*. http://rebeccanoeh.com/wp-content/uploads/2013/10/Scrum.jpg. – Eingesehen am 17.06.2017

[SG10] Spengler, Patrick ; Gammeter, Christoph: Modeling of 1:43 scale race cars. (2010)

[Ver14] Verschueren, Robin: *Design and implementation of a time-optimal controller for model race cars*, KU Leuven Fakultät Ingenieurswissenschaften, Diplomarbeit, 2014

[Vit] Vitzthum, Thomas: *Als Merkel in die Zukunft blicken soll, lacht das Auditorium*. https://www.welt.de/politik/deutschland/article165359594/Als-Merkel-in-die-Zukunft-blicken-soll-lacht-das-Auditorium.html. – Eingesehen am 12.06.2017, 15:18

[Wun] Wunderli, Lukas: *MPC based Trajectory Tracking for 1:43 scale Race Cars*