Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Hauke Schröder

Modellierung eines humanbiologisch motivierten Regelungssystems zur Steuerung autonomer Fahrzeuge

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hauke Schröder

Modellierung eines humanbiologisch motivierten Regelungssystems zur Steuerung autonomer Fahrzeuge

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Computer Science
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Stephan Pareigis
Zweitprüfer: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 20.12.2013

Hauke Schröder

Thema der Arbeit

Modellierung eines humanbiologisch motivierten Regelungssystems zur Steuerung autonomer Fahrzeuge

Stichworte

FAUST, Carolo-Cup, Eyetracking, Spurverfolgung, biologisch-motiviert, Simulation, menschliches Verhalten, menschliche Wahrnehmung,visuelle Wahrnehmung

Kurzzusammenfassung

Im Forschungsprojekt FAUST der Hochschule für Angewandte Wissenschaften Hamburg werden Fahrerassistenz- und autonome Systeme entwickelt. Die Regelung dieser Systeme ist in der Lage, unter bestimmten Bedingungen, ihre Aufgabe zu erfüllen. Ändern sich die Rahmenbedingungen zu stark ist dieses nicht mehr möglich. Der Mensch ist hingegen fähig, komplexe Situationen zu analysieren und auf Grundlage seiner Erfahrungen bestimmte Entscheidungen zu treffen. In dieser Arbeit wird das menschliche Verhalten beim Steuern eines Fahrzeugs in einer Simulation mit Hilfe eines Eyetrackers untersucht. Auf Basis dieser Ergebnisse wird die Grundlage geschaffen, bestimmte Teile der bestehenden Regelung der im FAUST Forschungsprojekt eingesetzten Carolo-Cup Plattform, durch human-biologisch motivierte Algorithmen auszutauschen.

Hauke Schröder

Title of the paper

Modeling of a human-biological motivated system to control autonomous vehicles

Keywords

FAUST, Carolo-Cup, Eyetracking, Tracing, biological-motivated, Simulation, human behavior, human perception, visual perception

Abstract

In the research project FAUST at the University of Applied Sciences Hamburg driver assistance and autonomous systems are developed. The control algorithms of these systems have the ability to control them under certain conditions. If the conditions change too much the autonomous system is no longer able to fulfill its task. Humans are able to analyze these complex situations and make decisions based on their experience. In this work, human behavior will be investigated while operating a vehicle within a simulation with an eyetracker. Based on these results it is possible to replace parts of the existing control algorithms used in the Carolo-Cup vehicles of the FAUST research project with human-biologically motivated control algorithms.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation . . . 2
- 1.2. Problemstellung . . . 3
- 1.3. Zielsetzung . . . 6
- 1.4. Entwicklungsprozess und Gliederung dieser Arbeit . . . 6

2. Grundlagen **7**
- 2.1. FAUST . . . 7
  - 2.1.1. Projektumfeld . . . 8
- 2.2. Regelung . . . 8
- 2.3. Visuelle Wahrnehmung . . . 9

3. Analyse menschlichen Verhaltens **11**
- 3.1. Personenbefragung . . . 11
  - 3.1.1. Auswertung der Befragung und Klassifikation der Antworten . . . 12
  - 3.1.2. Definition der Anforderungen an die Testsoftware . . . 13
- 3.2. Testsoftware . . . 13
  - 3.2.1. Softwarearchitektur . . . 14
  - 3.2.2. Projektion der Fahrbahn . . . 17
  - 3.2.3. Steuerung . . . 23
  - 3.2.4. Parametrierung von Tiles . . . 26
    - 3.2.4.1. Parametrierung von Events . . . 27
- 3.3. Testumgebung . . . 33
- 3.4. Testszenarien . . . 36
  - 3.4.1. Test 1 - Sakkadenmessung . . . 36
  - 3.4.2. Test 2 - Messung der Sprungantwort . . . 36
  - 3.4.3. Test 3 - Ermittlung von Ausweichtrajektorien . . . 37
  - 3.4.4. Test 4 - Beschleunigungsverhalten . . . 37
- 3.5. Testergebnis . . . 38
  - 3.5.1. Ermittlung der Region of Interest . . . 38
  - 3.5.2. Messung der Sprungantwort . . . 41
  - 3.5.3. Ermittlung der Ausweichtrajektorie . . . 45
  - 3.5.4. Ermittlung des Beschleunigungsverhaltens . . . 49
  - 3.5.5. Statistische Auswertung . . . 49

4. Reglerentwurf **53**
4.1. Reglerentwurf 53
4.1.1. Evaluation 55
4.2. Tracing-Algorithmus 57

5. Zusammenfassung **76**
5.1. Bewertung 76
5.2. Ausblick 77

Anhang A. Konfigurationsdatei der Simulation **82**

Danksagungen

Ich möchte mich bei allen Menschen, Verwandten, Freunden und Kommilitonen bedanken, die mich während des gesamten Studiums begleitet und mental unterstützt haben.

Mein besonderer Dank gilt Prof. Dr. Stephan Pareigis.

Für die moralische und finanzielle Unterstützung danke ich meinen Eltern Ortrud und Erwin Schröder sowie meiner Freundin Sandra Krafft, ohne die ich diesen Schritt in meinem Leben nicht hätte machen können!

Hauke Schröder, Dezember 2013

"Die dünnen Falten, die sich in die Stirn der Denker fressen, bekamen nicht selten ihre Nahrung durch den Zweifel."

Christa Schyboll

1. Einleitung

Im Großteil der heute ausgelieferten Kraftfahrzeuge sind verschiedenste elektronisch geregelte Fahrerassistenzsysteme vorhanden. Dazu gehört ein Antiblockiersystem, einige haben ein elektronisches Stabilitätsprogramm (ab 01.11.2014 Pflicht bei allen neu zugelassenen Fahrzeugen [12]) und wenige einen Spurhalte- oder Spurwechsel-, Notbrems- oder Toter-Winkel-Assistenten, Abstandsregelautomaten oder andere aktiv eingreifende Assistenzsysteme. Alle diese Systeme wurden entwickelt, um die Zahl der auftretenden Unfälle im Straßenverkehr zu reduzieren. Für den Monat Juli 2013 meldet das Statistische Bundesamt [33] 31.574 Unfälle mit Personenschaden und fast 300.000 für das Jahr 2012. Rund 90% dieser Unfälle resultierten aus menschlichem Versagen, wobei die übrigen 10% auf schlechte Straßen- und Witterungsverhältnisse, Hindernisse und sonstige Ursachen entfallen. Abbildung 1.1 zeigt eine Übersicht, über die am häufigsten gemeldeten Unfälle mit Personenschaden:

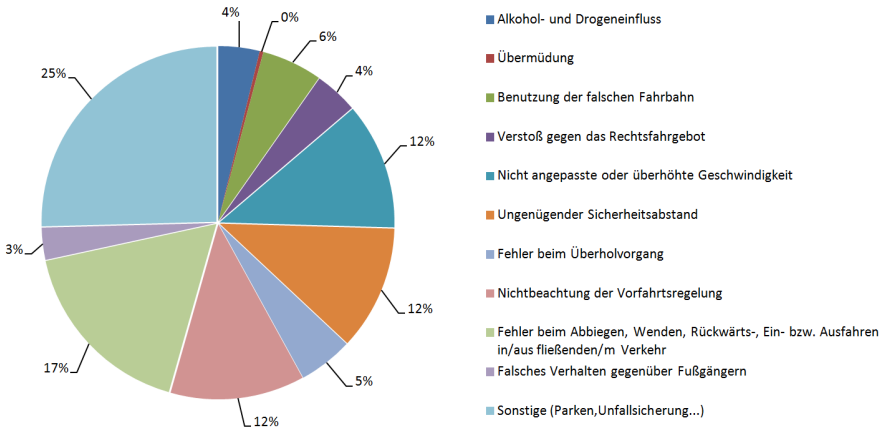

Abbildung 1.1.: Unfälle mit Personenschaden, verursacht durch Fahrerfehler nach Unfallursachen [33]

Mit der Unterstützung des Menschen durch die zuvor genannten Assistenzsysteme soll sich die allgemeine Sicherheit im Straßenverkehr verbessern. Durch Verwendung von Assistenzsystemen, die aktiv in die Längs- und Querführung des Fahrzeugs eingreifen können, ändert sich der Automatisierungsgrad von manueller hin zu teilautomatisierter Steuerung.

Der nächste Entwicklungsschritt ist der Übergang zur vollautomatisierten Steuerung eines Fahrzeugs, was unlängst viel komplexer ist, als die vorangegangenen Schritte. Hierbei besteht die Anforderung an das System, dass es alle Aufgaben des Fahrers übernimmt und diese mindestens genauso gut ausführt wie es der Mensch könnte. Dass diese Systeme keine Utopie sind, wurde in verschiedenen Forschungsprojekten (VW, BMW, Audi, Darpa, Google etc.) bereits nachgewiesen. Beispielweise nutzen Volkswagen [28] und BMW [13] in ihren Projekten neben viel Rechenleistung und diversen Sensoren auch eine sehr detaillierte Karte der möglichen Fahrstrecke. Mittels Metainformationen der Karte, kann das Fahrzeug beispielsweise auf die aktuell zulässige Geschwindigkeit zurückgreifen. Diese Anforderung schränkt das Einsatzgebiet stark ein. Allerdings kann es selbst in diesen stark begrenzten Einsatzgebieten zu unvorhersehbaren Situationen kommen, die Menschen tadellos meistern können. Während ein autonomes System nur versuchen kann, einen Fail-Safe-Zustand zu erreichen, in dem den Insassen und der Umwelt kein Schaden widerfährt. Um den ständig neuen Anforderungen der realen Welt gerecht werden zu können, hat der Mensch die Fähigkeiten erlernt, mit solchen Situationen und den teilweise sehr speziellen Anforderungen umgehen zu können. Adaption und Abstraktion ermöglichen es, bekannte Probleme zu erkennen, Unterschiede zu Vergangenem festzustellen und eine speziell auf die vorliegende Aufgabe angepasste Lösung zu entwickeln. Diese Fähigkeiten erscheinen uns selbstverständlich, doch sind sie alles andere als trivial.

1.1. Motivation

Seit Februar 2008 findet jährlich, ausgetragen von der Technischen Universität Braunschweig, der Carolo-Cup [5] statt. Beim Carolo-Cup handelt es sich um einen Hochschulwettbewerb, bei dem sich Studententeams mit der Entwicklung von autonomen Modellfahrzeugen im Maßstab 1:10 beschäftigen. Das Ziel dieses Wettbewerbs besteht darin, in dynamischen und statischen Disziplinen möglichst viele Punkte zu erreichen. Hierzu gehört das Befahren eines Parcours mit und ohne Hindernissen, Einparkvorgänge und ein Vortrag vor Fachpublikum, das aus Experten der Sponsoren und Lehrkräften verschiedener Hochschulen besteht.

Der im Februar 2013 ausgetragene Carolo-Cup brachte das schlechteste Ergebnis aller Zeiten für das Forschungsprojekt FAUST der Hochschule für Angewandte Wissenschaften Hamburg. Die Suche nach der Begründung für das Ergebnis ergab, dass mit jedem Zyklus den die Software durchlief, die verwendete projektive Transformation neu initialisiert worden ist. Dieser Prozess nimmt etwa 50 ms in Anspruch und wird üblicherweise nur während der Initialisierung der Software einmalig durchgeführt. Das bedeutet, dass die Ausführungszeit von geplanten 30 ms auf über 80 ms gestiegen ist. Welche Konsequenzen dies nach sich zieht, wird in der

Problemstellung (siehe Kapitel 1.2) genauer betrachtet. Somit sank die Abtastrate von 33 Hz auf 12 Hz, was ein korrektes Regeln unmöglich gemacht hat. Die Sieger des diesjährigen Carolo-Cups [6], das Team TUM Phoenix Robotics [40], regelt nach eigener Aussage mit einer Abtastrate von rund 100 Hz und befährt den Rundkurs schneller als alle anderen Teilnehmer. Hier zeigt sich erneut, dass eine möglichst hohe Samplerate das beste Ergebnis liefert, wie es im Nyquist-Shannon-Abtasttheorem (beziehungsweise WKS-Abtasttheorem) bereits 1948 definiert worden ist. In dem es heißt, dass die exakte Rekonstruktion eines Signals nur mit einer unendlich hohen Anzahl an Abtastpunkten möglich sei [8].

Laut Read und Meyer [27] ist die visuelle menschliche Wahrnehmung auf 10-12 Bilder pro Sekunde beschränkt. Dies entspricht der Samplerate, die dem Modellfahrzeug beim letzten Carolo-Cup zur Verfügung gestanden hat. Während das Regelungssystem, dass das Modellfahrzeug steuert nicht mehr in der Lage war, dieses stabil auf der Fahrbahn zu halten, tut der Mensch dieses im großen Maßstab bei deutlich höheren Geschwindigkeiten und in weitaus komplexeren Situationen. Egal ob die Fahrbahnmarkierung fehlt, es sich um einen Feldweg, eine mehrspurige Straße im dichten Stadtverkehr oder eine Autobahn handelt, ob die Straße nass, trocken oder glatt ist. Zumeist ist der menschliche Fahrer in der Lage, das Fahrzeug auf der Fahrbahn zu halten. Währenddessen übt er weitaus mehr Tätigkeiten (beispielsweise Telefonieren, Bedienung des Multimediasystems oder Kommunikation mit anderen Insassen) nebenher aus, als es das Regelsystem der Modellfahrzeuge tut. Dieses lässt den Schluss zu, dass der human-biologische Ansatz zur Beherrschung dieser Aufgabe sehr fehlertolerant und gut geeignet ist.

1.2. Problemstellung

Bei der Carolo-Cup Plattform des FAUST Projekts der Hochschule für Angewandte Wissenschaften Hamburg wird derzeit ein Regelungssystem mit einer Samplerate von rund 40 Hz eingesetzt. Aus Sicht der Regelungstechnik ist eine möglichst hohe Samplerate anzustreben, um auch bei hohen Geschwindigkeiten das Fahrzeug auf der Spur halten zu können. Der Grund hierfür besteht darin, dass das Fahrzeug durch seine Samplerate eine Totzeit, zwischen den einzelnen Samples hat, in denen sich der Zustand des Gesamtsystems kontinuierlich ändert, aber die Randbedingungen für die Regelung konstant bleiben. Bei einer Samplerate von 40 Hz bedeutet dies eine Totzeit von 25 Millisekunden. Die derzeit stabil regelbare Geschwindigkeit

liegt bei 2,5 m/s auf Geraden und 1,8 m/s in Kurven. So lässt sich die zurückgelegte Distanz in dieser Totzeit bestimmen:

$$d_{Totzeit\_Kurve} = \frac{1,8m * 0,025s}{s} \tag{1.1}$$

$$d_{Totzeit\_Kurve} = 0,045m \tag{1.2}$$

$$d_{Totzeit\_Gerade} = \frac{2,5m * 0,025s}{s} \tag{1.3}$$

$$d_{Totzeit\_Gerade} = 0,0625m \tag{1.4}$$

Die resultierenden Distanzen, die innerhalb der Totzeit zurückgelegt werden, sind im Rahmen des Kontextes angemessen und unproblematisch, denn die begrenzende Komponente ist in diesem Fall die Fahrphysik des Autos in Kurven, nicht die Regelung des Fahrzeugs. Aufgrund des hohen Gewichts, bedingt durch die Aufbauten und den deutlich erhöhten Schwerpunkt, neigt das Fahrzeug zum Ausbrechen in Kurven. Zu der zuvor benannten Totzeit kommt noch die Verarbeitungszeit der Aktorik (Lenkservo und Motor) hinzu, welche an dieser Stelle allerdings vernachlässigt wird. Da die Erhöhung der Geschwindigkeit also nicht möglich ist, wurde in Experimenten die Samplerate von 40 Hz auf zunächst 20 Hz (50 Millisekunden) gesenkt, welche bei konstanter Geschwindigkeit bereits zu großen Problemen bei der Regelung geführt hat:

$$d_{Totzeit\_Kurve} = \frac{1,8m * 0,05s}{s} \tag{1.5}$$

$$d_{Totzeit\_Kurve} = 0,09m \tag{1.6}$$

$$d_{Totzeit\_Gerade} = \frac{2,5m * 0,05s}{s} \tag{1.7}$$

$$d_{Totzeit\_Gerade} = 0,125m \tag{1.8}$$

Entsprechend beträgt die zurückgelegte Distanz in einer Kurve bei einer Samplerate von 10 Hz 18 cm, was nahezu der Breite einer Spur entspricht. Unter gegebenen Umständen ist dieser Fall nicht mehr regelbar, da die Richtungsänderungen der Fahrbahn zu spät erkannt werden.

Im realen Straßenverkehr treten innerhalb Deutschlands an vielen Stellen Geschwindigkeiten von 50 km/h innerorts, 100 km/h auf Landstraßen und teilweise deutlich über 200 km/h auf Autobahnen auf. Durch Umrechnung dieser Geschwindigkeiten in *m/s* kann entsprechend dem

vorhergehenden Modell, die zurückgelegte Distanz innerhalb der Totzeit bei einer Samplerate von 40 Hz bestimmt werden:

$$50km/h \approx 13,89m/s \tag{1.9}$$

$$100km/h \approx 27,78m/s \tag{1.10}$$

$$130km/h \approx 36,11m/s \tag{1.11}$$

$$200km/h \approx 55,56m/s \tag{1.12}$$

$$d_{Totzeit\_50} = \frac{13,89m * 0,025s}{s} \tag{1.13}$$

$$d_{Totzeit\_50} \approx 0,35m \tag{1.14}$$

$$d_{Totzeit\_100} = \frac{27,78m * 0,025s}{s} \tag{1.15}$$

$$d_{Totzeit\_100} \approx 0,69m \tag{1.16}$$

$$d_{Totzeit\_130} = \frac{36,11m * 0,025s}{s} \tag{1.17}$$

$$d_{Totzeit\_130} \approx 0,90m \tag{1.18}$$

$$d_{Totzeit\_200} = \frac{55,56m * 0,025s}{s} \tag{1.19}$$

$$d_{Totzeit\_200} \approx 1,39m \tag{1.20}$$

$$\tag{1.21}$$

Allerdings ist die visuelle menschliche Wahrnehmung inklusive der Weiterleitung an das Gehirn, wie bereits zuvor erwähnt, auf 10-12 Bilder pro Sekunde beschränkt (Read und Meyer [27]), auch wenn das menschliche Auge leistungsfähiger ist und einzelne Lichtblitze von mindestens 16 Millisekunden wahrnehmen kann [1]. Erschwerend kommt hinzu, dass der Mensch, anders als die Modellfahrzeuge aus dem FAUST Projekt, zusätzliche Arbeit und Zeit aufbringen muss, um eine Geschwindigkeits- oder Richtungsänderung in Gang zu setzen. Um die Reisegeschwindigkeit zu halten, befindet sich der Fuß auf dem Gaspedal. Tritt nun ein unerwartetes Ereignis auf, wird der Fuß vom Gaspedal angehoben und das Bremspedal betätigt. Dieser Prozess nimmt laut Triggs und Harris [37], welche sich auf diverse Studien aus der Zeit von 1950 bis 1973 stützen, 430 ms in Anspruch.

Aufbauend auf der vorherigen Argumentation bleibt eine Frage offen, welche die Problemstellung dieser Arbeit beschreibt:

Wie kann der Mensch trotz der widrigen Umstände, denen er unterliegt, ein Fahrzeug auch bei hohen Geschwindigkeiten so zuverlässig steuern?

1.3. Zielsetzung

Das Primärziel für diese Masterarbeit besteht darin, ein biologisch motiviertes Regelungssystem zu entwickeln. Für dieses wurden folgende Rahmenbedingungen definiert:

- Der Regler ist human-biologisch motiviert
- Die Samplerate entspricht der des Menschen
- Der Regler basiert auf Handlungsentscheidungen von menschlichen Fahrern

Als Sekundärziele werden weiter nachfolgende Aspekte betrachtet:

- Aggregation von Informationen zur Augenbewegung beim Steuern eines Fahrzeugs
- Extraktion von Handlungsvorschriften zur Steuerung von Vehikeln

1.4. Entwicklungsprozess und Gliederung dieser Arbeit

Die Gliederung der Arbeit richtet sich konsequent nach dem Entwicklungsprozess, der während des Masterprojekts durchlaufen worden ist.

Nach dieser Einführung wird in Kapitel 2 kurz auf Grundlagen eingegangen und themenrelevantes Hintergrundwissen zum besseren Verständnis vermittelt.

In Kapitel 3 wird auf die Vorgehensweise bei der Untersuchung des menschlichen Verhaltens eingegangen, wobei das Hauptaugenmerk auf der detaillierten Beschreibung der entwickelten Testsoftware, den Testszenarien und den darauf basierenden Ergebnissen liegt.

In Kapitel 4 wird auf den Entwurf des entwickelten Reglers eingegangen, dessen Grundlage die Ergebnisse der Tests aus Kapitel 3.5 sind. Im Anschluss daran, wird in Kapitel 4.2 auf den im Zuge des Masterprojekts entwickelten Tracing-Algorithmus zur Fahrspurextraktion eingegangen. Eine Evaluation der Versuchsergebnisse in Kapitel 4.1.1 vervollständigt den Prozess. Das Fazit schließt in Kapitel 5 die Ausarbeitung.

2. Grundlagen

Im folgenden Kapitel werden Grundlagen und Hintergrundwissen zum besseren Verständnis der Arbeit beschrieben.

2.1. FAUST

Diese Masterarbeit wird im Kontext des FAUST-Projekts der Hochschule für Angewandte Wissenschaften Hamburg geschrieben. Bei FAUST (**F**ahrerassistenz und **Au**tonome **S**ys**t**eme) handelt es sich um eine Forschungsgruppe, die sich schwerpunktmäßig mit der Erforschung von neuen Technologien für autonome Fahrzeuge und Fahrerassistenzsysteme beschäftigt. Die teilnehmenden Studenten können in einem sehr praxisnahen Projektumfeld ihre Projekt-, Bachelor- und Masterarbeiten schreiben, ihre theoretischen Ansätze an Hardware realitätsnah testen und die Funktionsfähigkeit evaluieren.

Insgesamt stehen vier Fahrzeugplattformen zur Verfügung, die in Abbildung 2.1 dargestellt sind.

Abbildung 2.1.: Übersicht der FAUST Fahrzeugplattformen

Von links nach rechts sind der IntelliTruck[1], die Carolo-Cup v.1 Plattform[2], die Carolo-Cup v.2 Plattform[3] und ganz rechts der CampusBot[4], der auf dem Chassis des vom Fraunhofer

[1] IntelliTruck: http://faust.informatik.haw-hamburg.de/index.php?section=fahrzeuge&sub=3
[2] CaroloCup v.1: http://faust.informatik.haw-hamburg.de/index.php?section=fahrzeuge&sub=2
[3] CaroloCup v.2: http://faust.informatik.haw-hamburg.de/index.php?section=fahrzeuge&sub=1
[4] CampusBot: http://faust.informatik.haw-hamburg.de/index.php?section=fahrzeuge&sub=4

Institut entwickelten Volksbot [43] basiert, dargestellt. Die Forschungsschwerpunkte des FAUST-Projektes sind in vier verschiedene Bereiche einzuteilen:

- Sensorintegration, Telemetrie und Bildverarbeitung
- Echtzeit- und Bussysteme
- Software- und Hardwarearchitektur
- Algorithmik und Regelungstechnik

2.1.1. Projektumfeld

Nachdem zuvor die Forschungsgruppe beschrieben worden ist, wird nachfolgend genauer auf das Projektumfeld der Masterarbeit eingegangen.

Abbildung 2.2.: Diamond - 2. Generation der Carolo-Cup Plattform

Während die in Kapitel 3.2 beschriebene Software auf einem Linux-PC mit angeschlossenem Lenkrad als Eingabegeräte ausgeführt wird, werden die Tests für den Tracing-Algorithmus aus Kapitel 4.2 auf der zweiten Version der Carolo-Cup Plattform durchgeführt. Hierbei handelt es sich um ferngesteuerte Modellfahrzeuge im Maßstab 1:10, mit denen am jährlich stattfindenden Carolo-Cup (vgl. [7]) der Technischen Universität Braunschweig teilgenommen wird. Das Ziel besteht darin, in vier verschiedenen Disziplinen möglichst viele Punkte zu erreichen. Darunter befindet sich eine statische und drei dynamische Disziplinen. Bei der statischen Disziplin handelt es sich um eine bewertete Präsentation vor Fachpublikum. Die drei dynamischen Disziplinen sind ein rückwärts paralleles Einparkmanöver, das Befahren eines Rundkurses ohne Hindernisse und das Befahren eines Rundkurses mit Hindernissen.

2.2. Regelung

Der Begriff *Regelung* beschreibt im Umfeld der Carolo-Cup Plattform des FAUST Projekts den fortlaufenden Prozess, das Fahrzeug auf der Fahrbahn zu halten, dabei Hindernissen auszuweichen, an Kreuzungen anzuhalten und in Parklücken einzuparken. Die genaue Definition einer

Regelung ist in dem Buch *Einführung in die DIN-Normen* von Martin Klein und Peter Kiehl, gemäß der DIN Norm 19226, zu finden:

> *Das Regeln, die Regelung ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (die zu regelnde Größe), erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.*
>
> *Martin Klein und Peter Kiehl - Einführung in die DIN-Normen [20]*

2.3. Visuelle Wahrnehmung

Visuelle Informationen werden vom Auge aufgenommen und durch den Sehnerv zum Sehzentrum des Gehirns transportiert und dort verarbeitet. Die Aufnahme dieser Informationen geschieht mit Hilfe von lichtempfindlichen Nervenzellen (Stäbchen und Zapfen). Elektromagnetische Strahlung aus dem sichtbaren Spektrum von 380 - 780 nm regt diese Nerven unterschiedlich stark an.
Die menschlichen Augen können zusammen horizontal ein Gesichtsfeld von etwa 180° und vertikal von etwa 130° [36] abdecken. Der genaue Wert ist von der physischen Beschaffenheit des einzelnen Menschen abhängig.

In Abbildung 2.3 ist ein Diagramm dargestellt, in dem die Abszisse die Abbildungsschärfe und die Ordinate die Abweichung von der fokalen Sichtlinie (0°) darstellt. Da es sich hier um das Diagramm eines linken Auges handelt, ist der sichtbare Bereich nach rechts kleiner als nach links, da dieser durch die Nase begrenzt wird.
Das Gesichtsfeld bezeichnet alle Dinge, die bei bewegungslosem Blick visuell wahrgenommen werden können. Innerhalb des Gesichtsfeldes gibt es drei zu unterscheidende Bereiche. Den fovealen Bereich, dieser nimmt 2,5° rund um die fokale Sichtlinie ein und ist der Bereich, in dem der Mensch mit maximaler Schärfe sehen kann, so Bornard et al. [18]. Der para-foveale Bereich deckt 2,5° bis 9° ab, hier können Farben noch unterschieden werden. Bei über 9° wird vom peripheren Bereich gesprochen. Eine mögliche Visualisierung der verschiedenen Bereiche des Gesichtsfeldes ist in Abbildung 2.4 dargestellt.

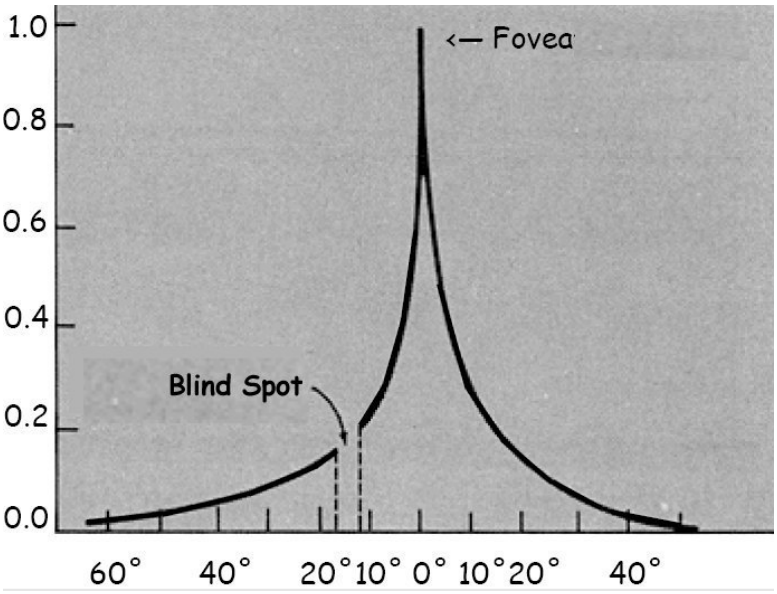

Abbildung 2.3.: Abbildungsschärfe auf der Netzhaut des linken Auges [15]

Abbildung 2.4.: Visualisierung der unterschiedlichen Bereiche des Gesichtsfeldes; von innen nach außen: foveal, parafoveal, peripher [18]

Diese Visualisierung wurde von dem *virtual eye*, einem Teilmodul des Moduls zur Simulation der visuellen Wahrnehmung aus COSMODRIVE [2] erzeugt. Bei COSMODRIVE (**Co**gnitive **S**imulation **Mo**del of the **Drive**r) handelt es sich um eine Software zur Simulation des menschlichen Verhaltens beim Fahren eines PKWs.

3. Analyse menschlichen Verhaltens

Um in Schritt 4 einen Regler entwickeln zu können, der ein ähnliches situationsbedingtes Verhalten aufweist, wie es ein Mensch beim Steuern eines Fahrzeugs macht, ist es zwingend notwendig zu untersuchen, aufgrund welcher Bedingungen eine bestimmte Entscheidung getroffen wird. Hierbei sind zwei Kategorien des Handelns zu unterscheiden. Zum einen das geplante Handeln, was den Regelfall bildet und zum anderen ungeplante Handlungen, die in der Regel durch äußere Einflüsse, wie beispielsweise Glätte, Nässe, schlechte Sicht, Wildwechsel oder spielende Kinder, hervorgerufen werden.
In diesem Abschnitt wird in 3.1 zunächst auf eine im Rahmen dieser Masterarbeit durchgeführten Personenbefragung eingegangen. Anschließend wird in Abschnitt 3.2 die zur Untersuchung des menschlichen Verhaltens entwickelte Software beschrieben. Es handelt sich hierbei um eine Fahrsimulation, mit der viele Situationen nachgestellt werden können, welche von den befragten Personen als problematisch eingestuft wurden. Danach wird in Kapitel 3.3 die Testumgebung beschrieben. Die Auswertung der Tests in Abschnitt 3.5 bildet den Abschluss des Kapitels.

3.1. Personenbefragung

Um herauszufinden, welche Situationen beim Steuern eines Fahrzeugs Probleme bereiten können, wurden insgesamt 23 Personen verschiedener Altersgruppen befragt. Die Auswahl dieser erfolgte zufällig. Wichtig war hierbei, erfahrene und weniger erfahrene Personen zu befragen. Voraussetzung war lediglich der Besitz eines Führerscheins der Klasse B. Die Frage, die gestellt wurde, lautete:

*Benennen Sie bitte **drei** Situationen, die bei der Steuerung eines PKWs kritisch beziehungsweise problematisch sein können!*

Das Ziel der Befragung war es nicht, gewichtete Aussagen (entsprechend einem Ranking von 1-10 oder dergleichen) zu erhalten, sondern die Benennung dreier im alltäglichen Straßen-

verkehr auftretenden Situationen. Wurden mehr als drei genannt, wurde die befragte Person gebeten, ein subjektives Ranking dieser zu erstellen und die drei unliebsamsten zu nennen.

3.1.1. Auswertung der Befragung und Klassifikation der Antworten

Die Auswertung der erhaltenen Antworten bedurfte einer Klassifikation der Antworten. Die Begründung hierfür besteht in der Vielfältigkeit der gegebenen Antworten, da bewusst keine Auswahl vorgegeben worden ist. Nachfolgend ist das bereits klassifizierte und dem Auftreten nach absteigende Ergebnis der Befragung zu sehen. Die Anzahl der jeweiligen Stimmen sind in Klammern zu finden. Sobald für eine Thematik öfter als ein Mal votiert worden ist, wurde diese der Liste hinzugefügt, die Übrigen sind unter dem Sammelbegriff "Sonstige" zusammengefasst worden.

- Änderung des Lenk- oder Fahrverhaltens **(13)**
  (Glätte, Nässe,...)
- Nebentätigkeiten **(12)**
  (Telefonieren, Rauchen, Bedienung von Multimediaequipment oder Handy,...)
- Änderung des Fahrbahnuntergrundes **(11)**
  (Schlaglöcher, Spurrillen, Aquaplaning,...)
- Zunehmende Verkehrsintensität oder äußere Einflüsse **(8)**
  (Stau, Stadtverkehr, Baustellen,...)
- Eingeschränkte Sichtverhältnisse **(7)**
  (Nebel, starker Regen oder Schneefall,...)
- Beschädigungen des Fahrzeugs oder Verletzungen des Fahrers **(6)**
  (Versagen der Lenkung oder der Bremsen, Reifenschaden,...)
- Störung durch andere Verkehrsteilnehmer **(4)**
  (Überholmanöver des Gegenverkehrs, Unachtsamkeit,...)
- Sonstige **(8)**
  (Müdigkeit, Unterschätzung von Gefahren, Fehleinschätzung von Abständen,...)

Insgesamt haben an der Befragung 23 Personen teilgenommen, wobei sich etwa 52% der Stimmen auf die ersten drei Punkte verteilen.

3.1.2. Definition der Anforderungen an die Testsoftware

Aufgrund der vorgenommenen Klassifikation (vgl. Kapitel 3.1.1), werden die Anforderungen an die Testsoftware nachfolgend definiert.
Folgende simulierbare Ereignisse zur Abdeckung der klassifizierten Ergebnisse der Befragung werden benötigt:

- Anpassung der Lateralgeschwindigkeit
- Anpassung der horizontalen Position des Fahrzeugs
- Anpassung der Lenkintensität
- Ausblenden oder Überblenden der Simulation
- Einblenden von Objekten auf der Fahrbahn

Auf die tatsächliche Implementierung und die Parametrierung der nachfolgend als *Events* bezeichneten unerwarteten Geschehnisse in der Simulation, die die zuvor beschriebene Liste abbildet, wird detailliert in Kapitel 3.2 eingegangen.
Weiter werden folgende allgemeine Anforderungen an die Simulation definiert:

- Perspektive: Pseudo-3D
- Straßenelemente: Geraden und Kurven
- Hindernisse mit Kollisionserkennung
- Steuerung des Fahrzeugs durch ein an den PC angeschlossenes Lenkrad mit Pedalen
- Aufzeichnung und Wiedergabe eines Durchlaufs
- Implementierung eines Interface zur Steuerung der Simulation durch ein anderes Programm

3.2. Testsoftware

In der Projektarbeit [16] wird auf die Vorgängerversion dieser Anwendung eingegangen, die im Zuge einer Projektarbeit entwickelt worden ist und ebenfalls das Ziel der Verhaltensanalyse hatte. Ein Screenshot dieser Software ist in Abbildung 3.1 dargestellt.

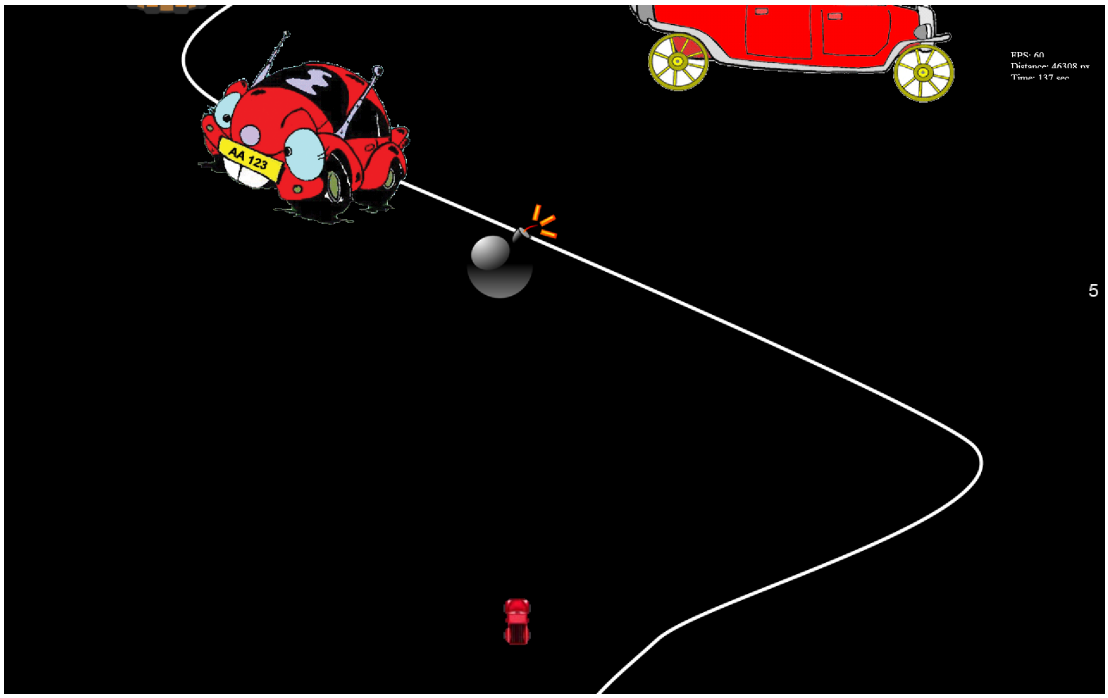

Abbildung 3.1.: Screenshot der Simulationssoftware aus [16]

Bei der Auswertung der Testergebnisse dieser Software hat sich allerdings gezeigt, dass die Ermittlung einer *Region of Interest*, basierend auf einer Software aus Vogelperspektive, nicht eindeutig durchzuführen ist. Aufgrund der gewählten Perspektive fehlen Informationen darüber, wann ein Objekt für den Steuernden von Interesse ist. Zudem war die Steuerung des Fahrzeugs in der Simulation lediglich mit der Tastatur möglich, was es nicht nur weniger intuitiv, sondern die ganze Simulation auch weniger realitätsnah wirken lies. Die an der ersten Simulationssoftware geübte Kritik wurde in Kapitel 3.1.2 beachtet und korrigiert.

Basierend auf den Anforderungen aus Kapitel 3.1.2, wurde das nachfolgend beschriebene Konzept für eine Software zur Verhaltensanalyse entwickelt.

3.2.1. Softwarearchitektur

Die in Java entwickelte Software wurde auf Basis des VCM-Paradigmas [21], einer Variation des MVC-Paradigmas [30] entwickelt.
Das VCM-Paradigma wurde zur Verwendung in stark interaktiven Anwendungen entwickelt und bildet heute den Standard bei der Spielentwicklung. Dieses Entwurfsmuster teilt eine

Applikation mit Benutzerinteraktion entsprechend dem MVC-Paradigma in drei grundlegende Komponenten ein: Model, View und Controller.

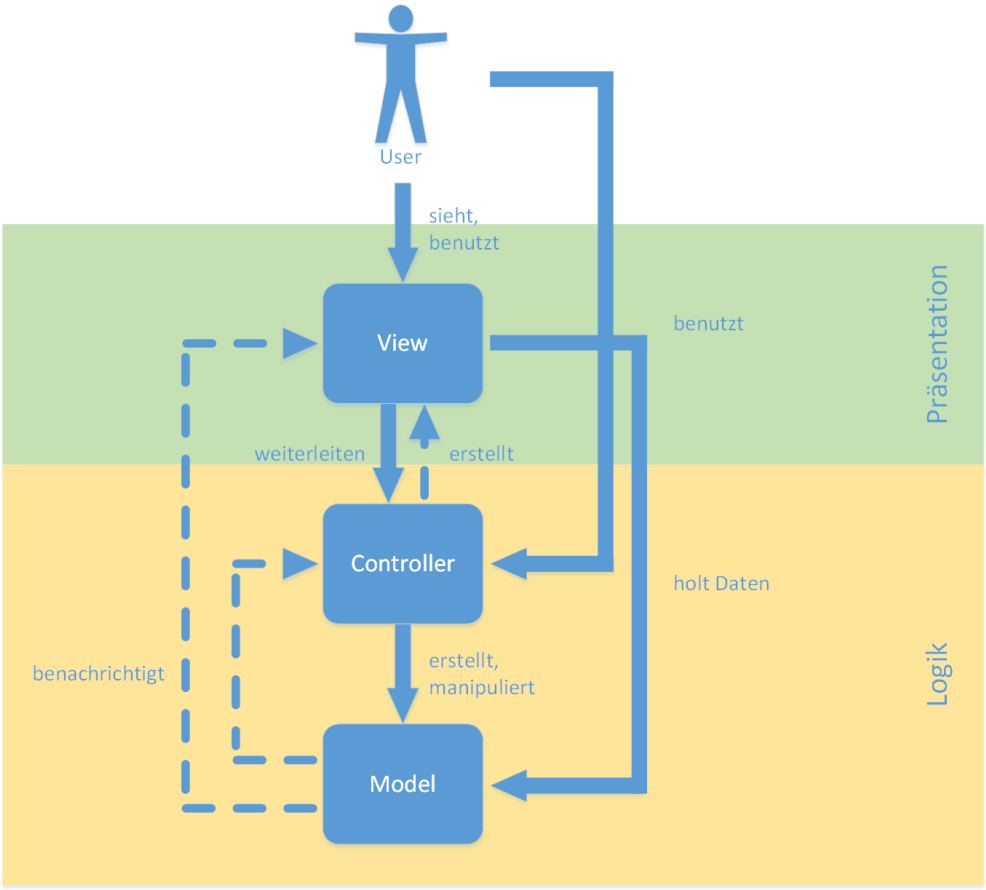

Abbildung 3.2.: VCM-Architektur der Simulationssoftware

Anders als beim MVC-Entwurfsmuster, sind die drei Komponenten bei der Variation in Schichten angeordnet. Die Beziehungen zwischen den Schichten sind anders als beim Original MVC-Entwurfsmuster sehr klar definiert. So ist klar, dass es bei Anwendungen nach dem VCM-Paradigma keine direkten Zugriffe von der View auf das Model geben darf. Weiter sind direkte Zugriffe (siehe Abbildung 3.2 - durchgezogene Pfeile) von Komponenten niedriger Schichten auf Komponenten höherer Schichten nicht möglich. Kommunikation in aufsteigender Hierarchie wird mit Hilfe von Nachrichten oder wie in diesem Projekt durch das Observer-Pattern etabliert (siehe Abbildung 3.2 - gestrichelte Pfeile “benachrichtigt“). Ein weiterer Unterschied des VCM-Paradigmas zum Original besteht darin, dass Programmlogik auch auf Ebene des Models vorhanden sein darf. Entsprechend dem Original erzeugt der Controller, die

View und das Model. Tastendrücke auf der Tastatur werden von einem *KeyEventDispatcher* angenommen und entsprechend der Funktionalität an die dazugehörige Methode weitergeleitet. Ein zyklischer Timer, triggert eine Update-Methode im Controller, durch die eine Verarbeitung der Software durchgeführt wird.

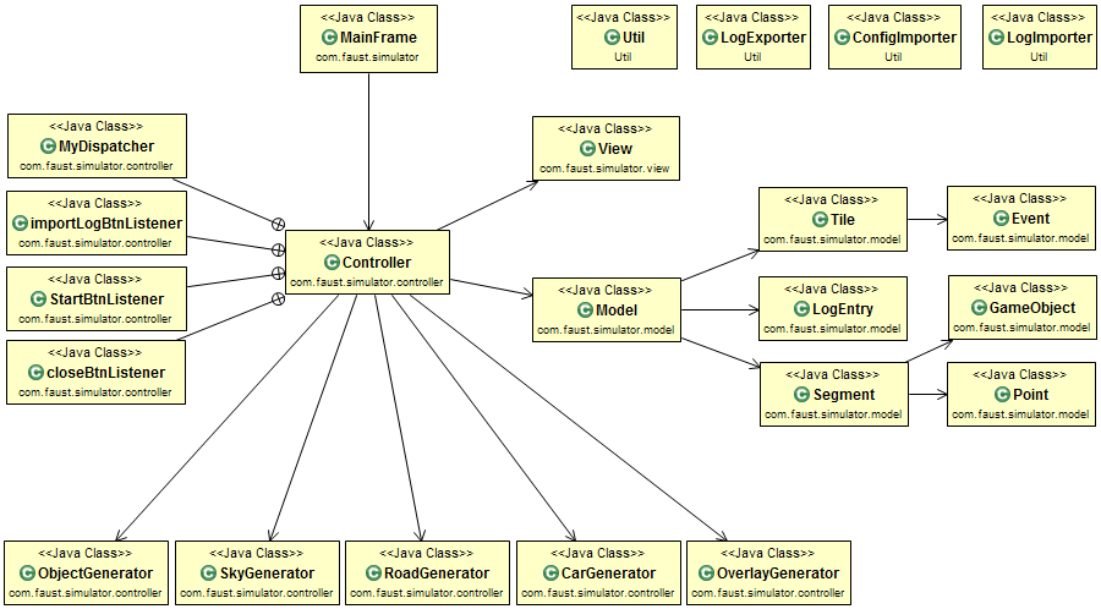

Abbildung 3.3.: Klassendiagramm der Simulationssoftware

Entsprechend dem Diagramm in Abbildung 3.3 hat der Controller Zugriff auf diverse *Helper-*, *Generator-* und *Util-Klassen*. Neben Im- und Export von Konfigurations- und Logdateien gibt es für jede Komponente der gezeichneten Simulation eine eigene Generator-Klasse, die für die Erzeugung bestimmter Elemente (Fahrbahn, Himmel, Objekte, Fahrzeug und Overlays) verantwortlich ist. Das Ergebnis ist beispielhaft in Abbildung 3.4 zu sehen:

Abbildung 3.4.: Vorschau der Simulationssoftware

Die Vorschau der gerenderten Simulation in Abbildung 3.4 zeigt, dass die Simulation in zwei Teile geteilt ist. Die oberen 50% stellen den Himmel dar, die unteren 50% die Fahrbahn. Die verwendete Perspektive wird häufig als *Pseudo-3D* oder *2.5D* bezeichnet. Hierbei wird vom gleichen visuellen Effekt Gebrauch gemacht, wie er beispielsweise von langen geraden Straßen oder Eisenbahntrassen bekannt ist. Jeder Betrachter weiß, dass sich die Parallelen niemals berühren, was wiederum den Eindruck vermittelt, dass sich die Entfernung mit zunehmender Nähe der Linien vergrößert. Die Objekte werden entsprechend der Entfernung zur Kamera im gleichen Verhältnis skaliert wie die Breite der Straße. Die Umsetzung dieser Linearperspektive wird in Kapitel 3.2.2 detailliert beschrieben.

3.2.2. Projektion der Fahrbahn

Nachfolgend wird auf die Erzeugung der Fahrbahn eingegangen. Hierzu wird zunächst das Fahrbahnmodell für Kurven und Geraden beschrieben, anschließend mathematisch auf die Projektion der Fahrbahn und die Implementierung in Java eingegangen.

Louis Gorenfeld beschreibt in seinem Online-Artikel [23] zunächst, dass eine Raster-Straße in der Pseudo-3D Perspektive eine Subklasse der Raster-Grafiken ist. Diese Grafiken zeichnen sich dadurch aus, dass jede Pixelzeile, die Teil der Simulation ist und die zuvor genannte Perspektive

aufweisen soll, eine andere Entfernung zum Betrachter abbildet. Durch die Skalierung der Straße mit zunehmender Entfernung zum Betrachter entsteht automatisch der in Kapitel 3.2.1 beschriebene Effekt stürzender Linien.

Die verwendete Straßengeometrie ist in 3.5 abgebildet.

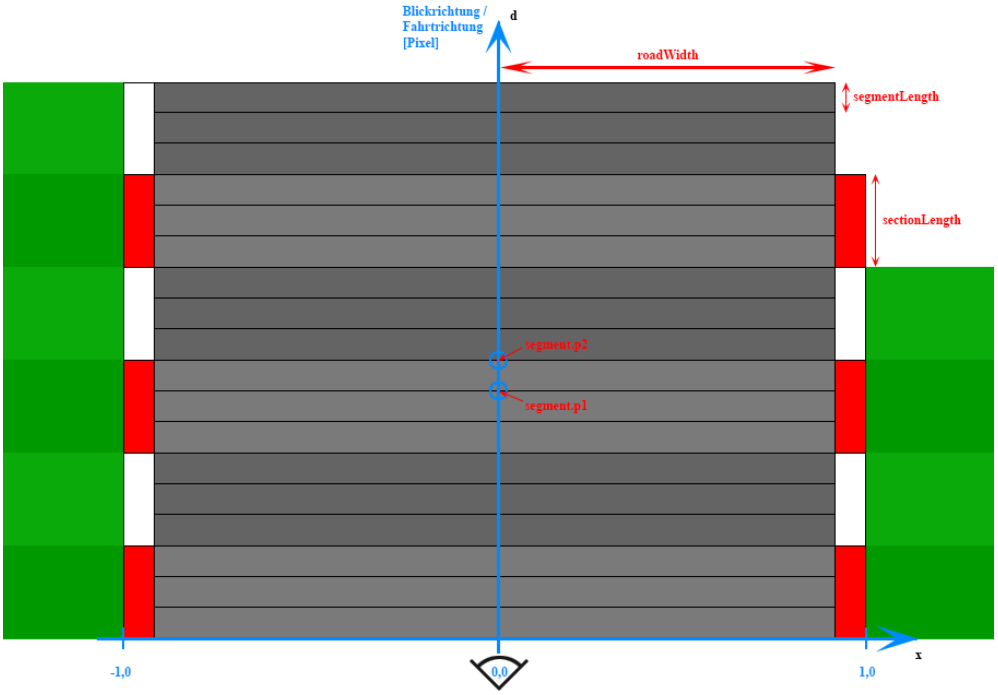

Abbildung 3.5.: Straßengeometrie

In blau ist das Koordinatensystem K_W eingezeichnet. Es ist in Blickrichtung beziehungsweise Fahrtrichtung ausgerichtet und für dieses gilt:

$$Abszisse(K_W) = x \tag{3.1}$$

$$wobei\ x := \{x \in \mathbb{R} \mid -1 \leq x \leq 1\} \tag{3.2}$$

$$und \tag{3.3}$$

$$Ordinate(K) = d \tag{3.4}$$

$$wobei\ d := \{d \in \mathbb{N} \mid 0 \leq d \leq L\} \tag{3.5}$$

$$mit\ L = sichtbare\ Fahrbahnlänge\ in\ [Pixel] \tag{3.6}$$

Als *Section* wird die Zusammenfassung von drei *Segmenten* bezeichnet. Ein Segment ist in K_W zunächst definiert als ein Rechteck mit der Länge L und der Breite B:

$$roadWidth = 2000Pixel \tag{3.7}$$

$$segmentLength = \ 200Pixel \tag{3.8}$$

$$B_{Segment} = 2 * roadWidth \tag{3.9}$$

$$L = segmentLength \tag{3.10}$$

Der Mittelpunkt der oberen horizontalen Begrenzung eines Segments wird als $p2$ und der Mittelpunkt der unteren horizontalen Begrenzung als $p1$ bezeichnet. Der Übersichtlichkeit halber, wird jedem Segment ein Punkt $p1$ und ein Punkt $p2$ zugeordnet, obwohl

$$Segment_0.p2 = Segment_1.p1 \tag{3.11}$$

ist. Bei einer geraden Fahrbahn befinden sich die Mittelpunkte aller Segmente auf einer geraden Linie.
Die Länge einer *Section* beträgt also 600 Pixel. Um den Eindruck einer Bewegung beim Fahren zu verstärken, sind neben einer gestrichelten Mittellinie ebenso die Farben der Fahrbahnbegrenzung, der Fahrbahn selbst und der Grünfläche neben der Fahrbahn entsprechend der *Section*-Grenzen abwechselnd unterschiedlich.
Neben Straßen werden für die Simulation auch Kurven benötigt. Es gibt hier verschiedene Ansätze, wie Kurven gebildet werden können. Der für diesen Zweck praktikabelste Ansatz wird sehr treffend von Louis Gorenfeld beschrieben:

> *To curve a road, you just need to change the position of the center-line in a curve shape. There are a couple ways to do this. One way is to do it the way the Z positions were done in "The Simplest Road": with three variables. That is, starting at the bottom of the screen, the amount that the center of the road shifts left or right per line steadily increases.*
>
> *Louis Gorenfeld*
> *[23]*

Sein Ansatz ist also, bezogen auf Abbildung 3.5, die *center-line* (blaue Linie) in die Form einer Kurve zu bringen. Um dieses umzusetzen, wird kurvenspezifisch eine Verschiebung von $segment.p2$ relativ zu $segment.p1$ durchgeführt. Wird für die Abbildung der angestrebten Krümmung der Fahrbahn mehr als ein Segment benötigt, muss der Krümmungswert akkumuliert werden, wie es in Abbildung 3.6 unten dargestellt ist:

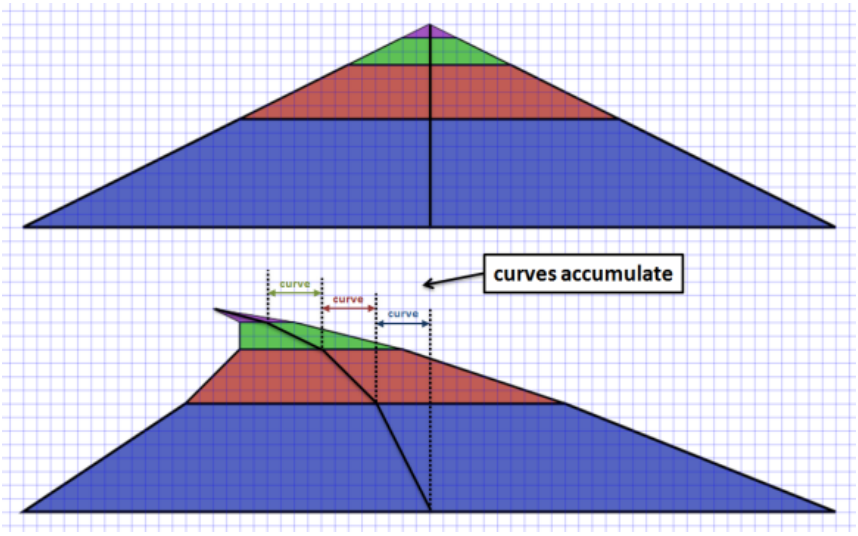

Abbildung 3.6.: Kurvenbildung bei verwendeter Straßengeometrie [17]

Die Abbildung stellt eine Gerade und eine nach links gekrümmte Fahrbahn mit je vier Segmenten in der angestrebten Perspektive dar. Die Pfeile zwischen den gestrichelten vertikalen Linien im unteren Bild zeigen den Abstand zwischen den beiden Punkten eines Segments ($segment.p1$ und $segment.p2$) bei der Bildung einer Kurve. Je größer der horizontale Abstand ist, desto stärker ist die resultierende Kurve gekrümmt. In der Simulation wurden Kurven entsprechend den Vorbildern der Realität gestaltet. Kurven beginnen mit einer leichten Krümmung und haben einen Zielkrümmungswert, der über eine festgelegte Anzahl von Segmenten erreicht wird. Gleiches Verfahren wird beispielsweise auch beim Ein- und Ausfahren einer Kurve angewendet.
An dieser Stelle ist es möglich eine Fahrbahn in Vogelperspektive in K_W (vgl. Abbildung 3.5) zu erzeugen. Um diese in die gewünschte Pseudo-3D Perspektive zu transformieren, sind einige Schritte notwendig, die nachfolgend beschrieben werden.

Zunächst werden einige grundlegende Parameter festgelegt, welche des Öfteren referenziert werden:

```
1 private int width          = 1000;
2 private int height         = 750;
3 private int skyHeight      = 375;
4 private int roadHeight     = height-skyHeight;
5 private int roadWidth      = 2000;
6 private int segmentLength  = 200;
7 private int sectionLength  = 3;
```

```
8   private int fieldOfView       = 100;
9   private int cameraHeight      = 1000;
10  private double cameraDepth    = (1.0d / Math.tan((fieldOfView / 2.0d)
11                                  * Math.PI / 180.0d));
12  private int drawDistance      = 300;
```

Die Ebene, auf die die Simulation projiziert wird, wird durch die Parameter *width* und *height* in Pixeln beschrieben. Das Fahrzeug und somit auch das horizontale Zentrum dieses Koordinatensystems befindet sich unten in der Mitte der Zeichenebene. Die Straße ist in der Breite durch *roadWidth* angegeben. Die Höhe der gezeichneten Straße auf der Zeichenebene ist auf *roadHeight* begrenzt. Der Parameter *drawDistance* gibt an, wieviele Segmente zur Zeichnung der Straße verwendet werden. Das bedeutet, dass 300 Segmente auf 375 Pixel verteilt werden. Zur Ermittlung der vertikalen Position jedes Segments, wird aufbauend auf Abbildung 3.7 wie nachfolgend beschrieben verfahren:

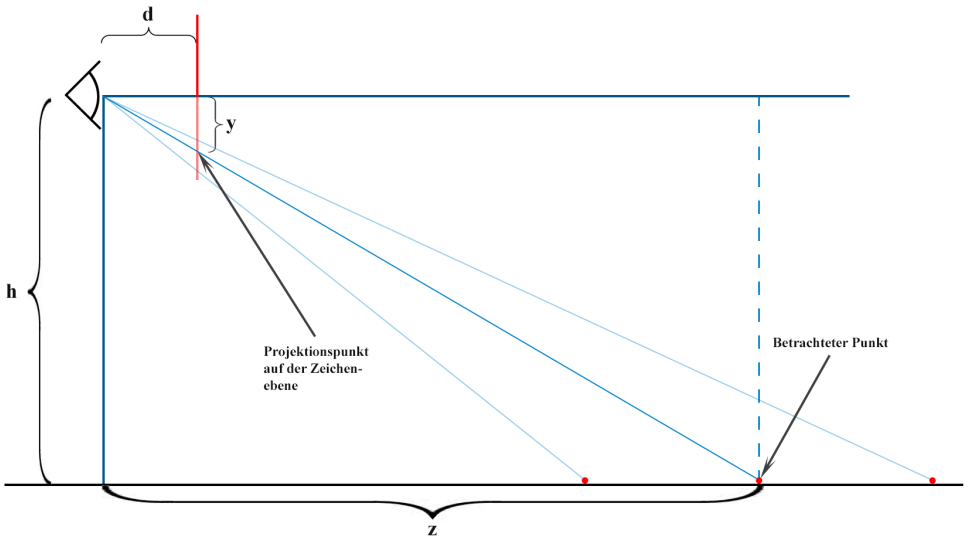

Abbildung 3.7.: Projektionsgeometrie zur Ermittlung der vertikalen Position eines Straßensegments

Links oben ist ein stilisiertes Auge dargestellt, das durch die Zeichenebene (rote vertikale Linie) auf drei entfernte Objekte blickt. Die Blicklinien zu diesen sind in blau dargestellt. Die mittlere Blicklinie wurde hervorgehoben, da die eingezeichneten Bemaßungen für diese gelten. Jeder betrachtete Punkt bildet zusammen mit dem Ursprung der auf ihn gerichteten Blicklinie und einem rechten Winkel gegenüber der Blicklinie ein großes Dreieck. Innerhalb dessen entsteht ein winkelgleiches kleineres Dreieck, das auf der rechten Seite durch die Zeichenebene

begrenzt wird. Aufgrund der Winkelgleichheit, kann folgende Beziehung zwischen den Seiten der beiden Dreiecke hergeleitet werden:

$$\frac{y}{h} = \frac{d}{z} \tag{3.12}$$

$$y = h * \frac{d}{z} \tag{3.13}$$

$$wobei\ h = cameraHeight \tag{3.14}$$

Durch Umstellung der Gleichung 3.12 nach y wird der Projektionspunkt auf der Zeichenebene, also die vertikale Koordinate, bestimmt werden. Wie zu Beginn dieses Kapitels beschrieben worden ist, werden Segmente durch zwei Punkte positioniert und durch *roadWidth* in der Breite begrenzt. Wird angenommen, dass sich das Auge direkt vor der Zeichenfläche befindet, kann $d = 1$ gesetzt werden. Entsprechend Gleichung 3.13 wird zur Ermittlung der horizontalen Koordinate nachfolgende Gleichung genutzt werden:

$$x = w * \frac{d}{z} \tag{3.15}$$

$$wobei\ w = \frac{roadWidth}{2} \tag{3.16}$$

Der letzte Schritt bei der Projektion besteht darin, die Fahrbahn mit zunehmender Entfernung schmaler werden zu lassen. Für diese Skalierung wird ein virtuelles Sichtfeld, häufig auch als FOV[1] bezeichnet, verwendet. In dieser Simulation wird ein Sichtfeld von 100° angenommen und entsprechend dem Artikel von Jake Gordon [17] umgesetzt. Durch Projektion auf die Zeichenebene wird der Anteil der gezeichneten Fahrbahn gegenüber dem Grünstreifen mit zunehmender Entfernung stetig kleiner:

$$dist_n = Segment_n.p1.y \tag{3.17}$$

$$scale_n = cameraDepth * dist_n \tag{3.18}$$

$$wobei\ n := \{n \in \mathbb{N}^+ \mid 1 \leq n \leq drawDistance\} \tag{3.19}$$

Wie in Abbildung 3.3 unten zu sehen ist, gibt es insgesamt 5 Generator-Klassen. Jede dieser Klassen erzeugt einige Teile der Simulation. Die Darstellung der Fahrbahn übernimmt im ersten Schritt der *RoadGenerator*. Die zuvor beschriebene Projektion wird für jede Pixelzeile im sichtbaren Bereich (*drawDistance*) durchgeführt. Für jedes Segment werden die zuvor

[1] FOV: Das ***field of view*** bezeichnet den sichtbaren Bereich, der innerhalb eines Momentes wahrgenommen werden kann.

genannten Projektionsparameter, also die Positionen seiner beiden Mittelpunkte und die Breite der Fahrbahn an diesen, bestimmt. Die eigentliche Darstellung auf der Zeichenebene geschieht durch die Verwendung von Polygonen. Das Ergebnis ist in Abbildung 3.8 oben links zu sehen.

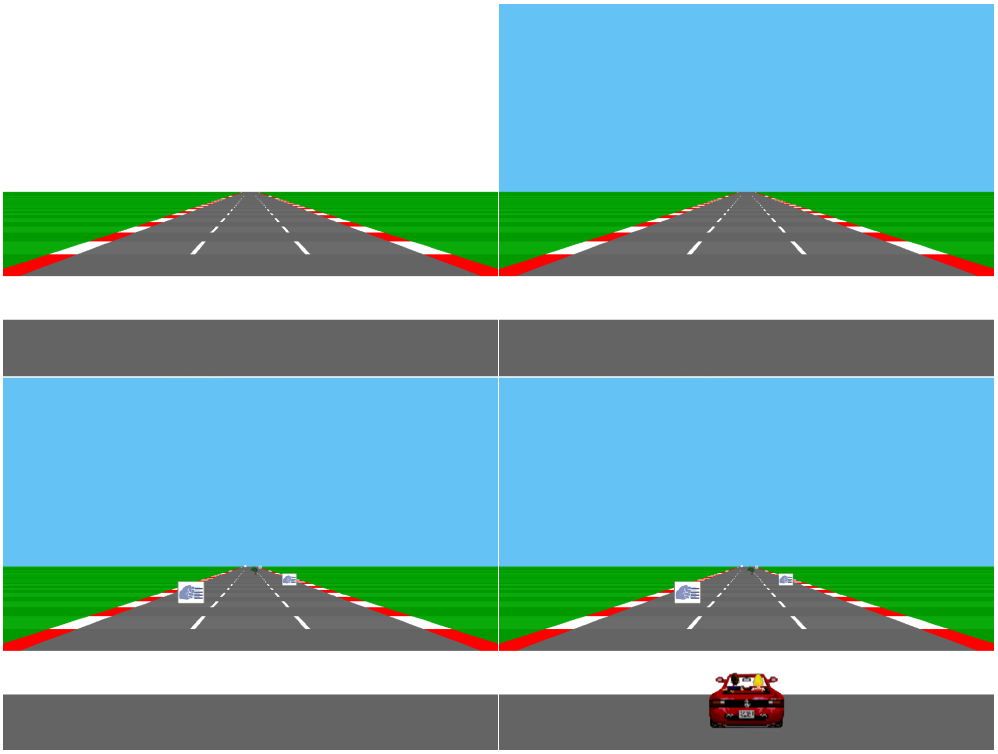

Abbildung 3.8.: Übersicht der Einzelschritte zur Erzeugung der Simulation

Im oberen rechten Teil wird der Himmel hinzugefügt. Unten links werden Objekte hinzugefügt, die der gleichen Skalierung unterliegen, wie die Fahrbahn. Im vierten Schritt, wird das Fahrzeug unten in die Mitte der Zeichenebene gezeichnet. Da die Position des Fahrzeugs fix ist, wird beim Lenken die Fahrbahn horizontal verschoben. In Kurven wirken geschwindigkeitsabhängige Zentrifugalkräfte.

3.2.3. Steuerung

Eine der Anforderungen aus Kapitel 3.1.2 bestand darin, eine intuitive und möglichst realitätsgetreue Steuerung für die Simulation zu entwickeln. Bei vielen Rennsimulationen wird die Tastatur als primäre Steuerung genutzt. Mögliche Alternativen hierfür sind: *Gamepads*, *Joysticks* oder *Lenkräder*. Unter den genannten Optionen ist das Steering Wheel die für diesen Zweck am besten geeignete Art der Steuerung. Während bei einer Tastatur lediglich Tas-

tendrücke, also zwei Zustände (gedrückt und nicht gedrückt) ausgewertet werden können, liefert die Auswertung des Steeringwheels permanent Werte darüber, wie stark das Lenkrad eingeschlagen ist. Bei dem für dieses Projekt gewählten Steering Wheel handelt es sich um ein *Logitech Momo Racing Wheel* [22] (siehe Abbildung 3.9).

Abbildung 3.9.: Logitech Momo Racing

Es bietet eine Auflösung von 9 Bit bei 270° auf der Lenkachse, wobei das erste Bit als Vorzeichenbit fungiert und 8 Bit Genauigkeit bei jedem der beiden Pedale. Die übrigen Buttons sind bitweise auszuwerten, werden in diesem Kontext allerdings nicht benötigt.

Der Anschluss des Logitech Momo Racing erfolgt via USB. Um auf das Steering Wheel zugreifen zu können, wird die *Java HID API* [9] verwendet. Bei dieser API handelt es sich um einen *JNI Wrapper* für die in C entwickelte Multi-Plattform *HID-API* [32]. Der JNI Wrapper abstrahiert den Zugriff auf die *HID API* auf Basis des Java Native Interfaces, das Zugriffe aus der Java Virtual Machine auf externe Komponenten, wie Betriebssystemfunktionen, Geräte oder externe Bibliotheken ermöglicht.

Die zugrundeliegende HID API benötigt erweiterte Rechte, um Zugriff auf die vom Betriebssystem geschützten USB-Komponenten, genauer HID-Class[2] Komponenten zu erhalten. Die beiden nachfolgend dargestellten Funktionen, die für den Zugriff auf ein Device genutzt werden, können unter Windows 7 nicht mehr genutzt werden.

```
/**
 * Convenience method to find and open device by path
 *
 * @param path USB device path
 * @return open device reference <code>HIDDevice<code> object
 * @throws IOException in case of internal error
 * @throws HIDDeviceNotFoundException if devive was not found
 */
public HIDDevice openByPath(String path)
       throws IOException, HIDDeviceNotFoundException
{
   ...
}

/**
 * Convenience method to open a HID device using a Vendor ID
 * (VID), Product ID (PID) and optionally a serial number.
 *
 * @param vendor_id USB vendor ID
 * @param product_id USB product ID
 * @param serial_number USB device serial number (could be <code>null<code>)
 * @return open device
 * @throws IOException in case of internal error
 * @throws HIDDeviceNotFoundException if devive was not found
 */
public HIDDevice openById(int vendor_id,
                          int product_id,
                          String serial_number)
       throws IOException, HIDDeviceNotFoundException
{
   ...
}
```

Da dieses Problem zum Zeitpunkt der Entwicklung noch nicht behoben war, wurde die Software zur Ausführung unter einem Linux Betriebssystem entwickelt. Hier gibt es keine Einschrän-

[2] Die USB HID-Class ist Teil der USB Spezifikation [41] und definiert eine Klasse von Geräten, zur Kommunikation des Menschen mit dem PC, wie Tastaturen, Mäusen, Gamecontrollern oder alphanumerischen Displays

kungen beim Zugriff auf die angeschlossenen Geräte. Das verwendete Logitech Momo Racing Wheel wird mit folgender VendorID / ProductID Kombination adressiert:

```
static final int VENDOR_ID  = 1133;
static final int PRODUCT_ID = 51715;
```

Mit diesen Daten kann der Zugriff auf das Gerät erfolgen:

```
com.codeminders.hidapi.ClassPathLibraryLoader.loadNativeHIDLibrary();
HIDManager hidManager = HIDManager.getInstance();
HIDDeviceInfo[] infos = hidManager.listDevices();
HIDDevice hidDevice;
for (HIDDeviceInfo info : infos) {
  if (info.getVendor_id() == VENDOR_ID
      && info.getProduct_id() == PRODUCT_ID) {
        hidDevice = info.open();
        byte[] buf = new byte[2048];
    while (true) {
      int n = hidDevice.readTimeout(buf, 5000);
      ...
    }
  }
}
```

Nach der Deklaration und Instanziierung, wird in den *Zeilen 5* bis *7* mit der Suche nach dem richtigen Device begonnen. Sobald dieses gefunden wurde, werden in *Zeile 12* Daten vom Gerät gelesen und in *buf* abgelegt. Auf die Darstellung der weiteren Verarbeitung ist hier verzichtet worden.

3.2.4. Parametrierung von Tiles

Ein Streckenabschnitt wird nachfolgend als *Tile* bezeichnet. Ein Tile enthält zum einen Parameter, wie den Namen, die Initialgeschwindigkeit, die Maximalgeschwindigkeit und das Ende des Tiles. Zum anderen eine Liste von Events mit n Einträgen, wobei $n <= Integer.MAX\_VALUE$ Elemente, also die maximale Länge einer Array-List aus dem Package java.util, ist.
Die Reihenfolge der Tiles in der Konfigurationsdatei *ressources/config.xml* sollte aufsteigend nach dem Ende der Tiles sortiert sein. Es ist dabei zu beachten, dass das Ende von Tile 1 gleichzeitig der Beginn von Tile 2 ist. Dies impliziert, dass im Fall von Tiles mit gleichen Ende-Parametern, lediglich das erstgenannte Tile gestartet wird.

Da es sich bei der Konfigurationsdatei um eine XML-Datei handelt, ist ein fehlerfreies DOM[3] zwingend notwendig, ansonsten kann die Datei nicht erfolgreich geparsed werden. Nachfol-

[3] **D**ocument **O**bject **M**odel: Spezifikation einer Schnittstelle für den Zugriff auf HTML- und XML-Dokumente

gend ist ein Minimalbeispiel für eine Konfiguration mit einem Tile ohne Events dargestellt:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <fps>30</fps>
  <speed>0</speed>
  <track>
    <tile>
      <name>Level 1</name>
      <speed>100</speed>
      <playerposx>-0.8</playerposx>
      <end>100000</end>
      <events>
      </events>
    </tile>
  </track>
</configuration>
```

XML-Konfiguration 3.1: Beispielkonfiguration eines Tiles ohne Events

Mit dieser Konfiguration würde das Spiel auf 30 Frames pro Sekunde und einer Initialgeschwindigkeit von 0 Pixeln pro Frame starten. Mit einem Klick auf den *Start-Button* wird das erste Tile mit dem Namen *Level 1* geladen. Die Initialgeschwindigkeit des Fahrzeugs wird auf 100 Pixel pro Frame gesetzt. Anschließend ist die Geschwindigkeit wieder durch die Versuchsperson kontrollierbar. Zu Beginn des Tiles wird das Fahrzeug einmalig horizontal auf der Straße positioniert, entsprechend Abbildung 3.11 zieht eine Position von -0.8 eine Positionierung im linken äußeren Bereich der Straße nach sich.

3.2.4.1. Parametrierung von Events

In diesem Kontext wird ein Event als ein Vorkommnis bezeichnet, das ohne das Zutun der Versuchsperson zu einem festgelegten Zeitpunkt eintritt und eine bestimmte Reaktion seitens der Testsoftware auslöst. Die möglichen Events sollen die in Kapitel 3.1.1 genannten problematischen Situation abbilden und eine situationsbedingte Verhaltensanalyse der Versuchspersonen ermöglichen. Nachfolgend werden zunächst alle Events beschrieben. Im Anschluss daran wird das beschriebene Tile aus 3.2.4 um einige Events erweitert.

Objekteinblendung

Es stehen verschiedene Objekte zur Verfügung, welche frei auf der Fahrbahn positioniert werden können:

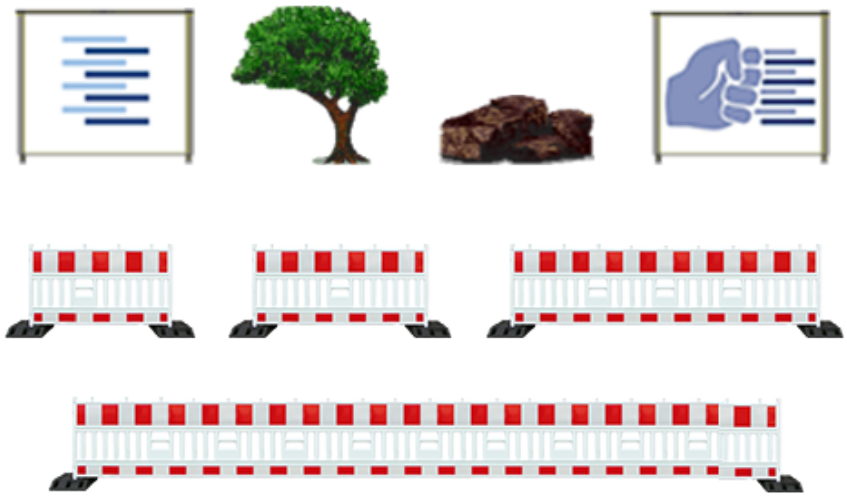

Abbildung 3.10.: Objekte zur Platzierung auf der Fahrbahn

Die horizontale und vertikale Position eines Objekts O wird durch einen Punkt $P(x, d)$ aus dem Koordinatensystem K bestimmt. Für K gilt:

$$Abszisse(K) = x \tag{3.20}$$
$$wobei\ x := \{x \in \mathbb{R} \mid -1,5 \leq x \leq 1,5\} \tag{3.21}$$
$$und \tag{3.22}$$
$$Ordinate(K) = d \tag{3.23}$$
$$wobei\ d := \{d \in \mathbb{N} \mid 0 \leq d \leq L\} \tag{3.24}$$
$$mit\ L = sichtbare\ Fahrbahnlänge\ in\ [Pixel] \tag{3.25}$$

Die Fahrbahnlänge L ist abhängig vom Aufbau der Strecke und somit variabel. Ein Teil des Wertebereichs der Abszisse wird durch den als Straße gekennzeichneten Bereich der Simulation beschrieben. In Abbildung 3.11 ist ein Ausschnitt der Simulation mit vier verfügbaren Objekten zu sehen. In schwarz sind einige Werte für x beispielhaft eingezeichnet worden:

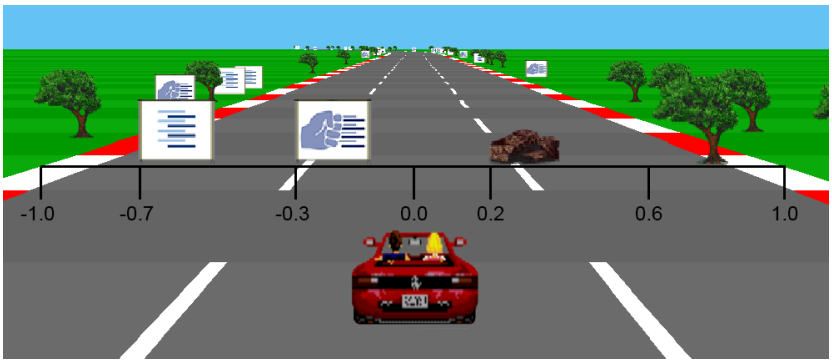

Abbildung 3.11.: Positionsangaben zur Platzierungen von Objekten auf der Fahrbahn

Um die Objekte entsprechend der Abbildung 3.11 einzublenden, sind folgende Einträge in der *config.xml* innerhalb des *<events></events>*-Tags eines Tiles einzufügen:

```
<!-- HAW Schild -->
<object gameObjectNumber="1" distance="10000" value="0.6" />
<!-- FAUST Schild -->
<object gameObjectNumber="2" distance="10000" value="0.2" />
<!-- Felsen -->
<object gameObjectNumber="3" distance="10000" value="-0.3" />
<!-- Baum -->
<object gameObjectNumber="4" distance="10000" value="-0.7" />
<!-- Schranke klein -->
<object gameObjectNumber="9" distance="10000" value="-0.2" />
<!-- Baum mittel -->
<object gameObjectNumber="10" distance="10000" value="-0.2" />
<!-- Baum groß -->
<object gameObjectNumber="11" distance="10000" value="-0.2" />
```

XML-Konfiguration 3.2: Auflistung und Konfiguration verfügbarer Objekte

Bewegtes Objekt

Analog zu der Konfiguration von statischen Objekten 3.2 steht ein sich bewegendes Objekt zur Verfügung. Die vertikale Position (also die Distanz) ist konstant. Die horizontale Position des Objekts ändert sich mit jedem Zyklus um 0,025 (entsprechend Definition 3.20). Das Maximum und Minimum aus Definition 3.20 wird dabei nicht überschritten. Ist eine dieser Grenzen erreicht wird die Bewegungsrichtung geändert.

```
1 <object gameObjectNumber="5" distance="16000" value="0.2" />
```

XML-Konfiguration 3.3: Konfiguration eines bewegten Objekts

Geschwindigkeitsänderung

Zur Abbildung von Geschwindigkeitsänderungen wurde das Event mit der *gameObjectNumber* 30 definiert. Beim Erreichen der eingetragenen Pixeldistanz (Attribut *distance*) wird die aktuell gefahrene Geschwindigkeit auf den Wert des *value*-Attributs gesetzt. Die Geschwindigkeit kann anschließend direkt wieder kontrolliert werden.

```
1 <object gameObjectNumber="30" distance="16000" value="50" />
```

XML-Konfiguration 3.4: Konfiguration einer Geschwindigkeitsänderung

Positionsänderung des Fahrzeugs

Das Event mit der *gameObjectNumber* 31 ermöglicht die sofortige Änderung der horizontalen Position des Fahrzeugs auf der Fahrbahn bei Erreichen der angegebenen Pixeldistanz (Attribut *distance*). Die Positionierung des Fahrzeugs geschieht entsprechend Abbildung 3.11. Der Wertebereich des *value*-Attributs unterliegt dabei der Definition der Abszisse aus 3.20.

```
1 <object gameObjectNumber="31" distance="17000" value="0.8" />
```

XML-Konfiguration 3.5: Konfiguration einer Positionsänderung des Fahrzeugs

Geschwindigkeitsbegrenzung

Die maximal mögliche Geschwindigkeit kann durch das Event mit der *gameObjectNumber* 32 geändert werden. Bei Erreichen der eingetragenen Pixeldistanz (Attribut *distance*) wird die bisher mögliche Maximalgeschwindigkeit auf den Wert des *value*-Attributs gesetzt. Die aktuelle Geschwindigkeit des Fahrzeugs in der Simulation wird gegebenenfalls der neuen maximalen Geschwindigkeit angepasst. Die Änderung der Höchstgeschwindigkeit ist bis zur nächsten Änderung dieser persistent.

```
1 <object gameObjectNumber="32" distance="18000" value="100" />
```

XML-Konfiguration 3.6: Konfiguration einer Geschwindigkeitsbegrenzung

Fixierung der Geschwindigkeit

Für einige Tests ist es notwendig, Geschwindigkeit festzulegen und eine Änderung durch die Versuchsperson nicht zu gestatten. Das Event mit der *gameObjectNumber* 33 deaktiviert bei Erreichen der eingetragenen Pixeldistanz (Attribut *distance*) Geschwindigkeitsänderungen durch die Bedienung der Pedale. Eine Änderung dieser durch die Konfiguration ist weiterhin möglich. Mit *gameObjectNumber* 34 kann die Kontrolle der Geschwindigkeit durch die Versuchsperson wieder aktiviert werden.

```
1 <object gameObjectNumber="33" distance="17000" value="0" />
2 <object gameObjectNumber="34" distance="19000" value="0" />
```

XML-Konfiguration 3.7: Konfiguration der Geschwindigsfixierung

Änderung der Lenkrad-Nullstellung

Das Event mit der *gameObjectNumber* 35 bietet die Möglichkeit die Lenkrad-Nullstellung zu verschieben. Ein negativer Wert d im *value*-Attribut verschiebt die Nullstellung gegen den Uhrzeigersinn, ein positiver Wert d entsprechend im Uhrzeigersinn.
$d := \{d \in \mathbb{N} \mid -100 \leq d \leq 100\}$

```
1 <object gameObjectNumber="35" distance="20000" value="-20" />
```

XML-Konfiguration 3.8: Konfiguration der Lenkrad-Nullstellungsänderung

Pausierung von Tests

Mit *gameObjectNumber* 50 kann ein Test für eine festgelegte Zeit in Sekunden unterbrochen werden.

```
1 <object gameObjectNumber="50" distance="21000" value="5" />
```

XML-Konfiguration 3.9: Konfiguration der Pausierung von Tests

Anzeige von Texten

Durch Verwendung von *gameObjectNumber* 70 kann bei einer bestimmten Distanz ein einzeiliger Text angezeigt werden. Es bietet sich an, dieses Event in Verbindung mit dem vorhergehend beschriebenen Event *50* 3.9 zu nutzen.

```
1 <object gameObjectNumber="70" distance="21000" value="Text" />
```

XML-Konfiguration 3.10: Konfiguration der Texteinblendung

Screenüberblendung

Um auch die Verhaltensweisen untersuchen zu können, die eine Versuchsperson beim Steuern eines Fahrzeugs ohne Sicht an den Tag legt, besteht die Möglichkeit, die gesamte Simulationsfläche einfarbig zu überblenden. Hierfür stehen die Farben schwarz, weiß, rot, gelb, grün und blau zur Verfügung. Jede Farbe ist mit einer dazugehörigen *gameObjectNumber* kodiert.

```
<!-- grün -->
<object gameObjectNumber="94" distance="24600" value="100" />
<!-- gelb -->
<object gameObjectNumber="95" distance="24700" value="100" />
<!-- blau -->
<object gameObjectNumber="96" distance="24800" value="100" />
<!-- rot -->
<object gameObjectNumber="97" distance="24900" value="100" />
<!-- weiß -->
<object gameObjectNumber="98" distance="25000" value="100" />
<!-- schwarz -->
<object gameObjectNumber="99" distance="25100" value="100" />
```

XML-Konfiguration 3.11: Konfiguration möglicher Screenüberblendungen

Die zuvor beschriebenen Events können in beliebiger Reihenfolge und Anzahl in der config.xml eingebunden werden. Nachfolgend ist ein Beispiel für eine 1-Tile Konfiguration zu sehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <fps>30</fps>
  <speed>0</speed>
  <track>
    <tile>
      <name>Level 1</name>
      <speed>100</speed>
      <playerposx>-0.8</playerposx>
      <end>100000</end>
      <events>
        <object gameObjectNumber="1" distance="10000" value="0.6" />
        <object gameObjectNumber="2" distance="10000" value="0.2" />
        <object gameObjectNumber="3" distance="10000" value="-0.3" />
        <object gameObjectNumber="4" distance="10000" value="-0.7" />
        <object gameObjectNumber="30" distance="15000" value="0" />
        <object gameObjectNumber="30" distance="16000" value="50" />
        <object gameObjectNumber="31" distance="16900" value="0.0" />
```

```
19          <object gameObjectNumber="31" distance="17000" value="0.8" />
20          <object gameObjectNumber="31" distance="17100" value="-0.8" />
21          <object gameObjectNumber="31" distance="17100" value="0.0" />
22          <object gameObjectNumber="32" distance="18000" value="100" />
23          <object gameObjectNumber="32" distance="23000" value="220" />
24          <object gameObjectNumber="94" distance="24600" value="100" />
25          <object gameObjectNumber="95" distance="24700" value="100" />
26          <object gameObjectNumber="96" distance="24800" value="100" />
27          <object gameObjectNumber="97" distance="24900" value="100" />
28          <object gameObjectNumber="98" distance="25000" value="100" />
29          <object gameObjectNumber="99" distance="25100" value="100" />
30        </events>
31      </tile>
32    </track>
33  </configuration>
```

XML-Konfiguration 3.12: Vollständiges Beispiel einer 1-Tile Konfiguration

3.3. Testumgebung

Da das Usability Labor der Hochschule für Angewandte Wissenschaften Hamburg keine Testumgebung für Linuxanwendungen zur Verfügung stellen kann, wurde der Test in Räumlichkeiten der Forschungsgruppe FAUST durchgeführt. Die Installation der Software zur Auswertung des Tobii X120 Eyetrackers ist lediglich auf einem Windows PC und die Ausführung der Testanwendungen, wie in Kapitel 3.2.3 beschrieben, ausschließlich unter Linux möglich. Der Eyetracker macht die Sakkadensprünge, also das Springen der Augen vom einen Fixationspunkt zum nächsten Fixationspunkt sichtbar. Um eine Zuordnung der Sakkadensprünge zu Ereignissen in der Simulation durchführen zu können, schneidet die von der Firma Tobii [38] entwickelte und für diese Masterarbeit kostenlos zur Verfügung gestellt Software Tobii Studio 3.2 das aktuelle Bild, das vom PC angezeigt wird, mit der Auswertung des Eyetrackers zusammen. So entsteht ein Video, das zu jedem Zeitpunkt der Simulation zeigt, was die Versuchsperson wie lange fixiert hat.
Die Problematik besteht darin, das Bild mit möglichst geringer Verzögerung vom Linux PC auf den Windows PC zu übertragen, damit es dort angezeigt und mit der Auswertung des Eyetrackers zusammengeschnitten werden kann. Zur Lösung wurden verschiedene Ansätze getestet, die kurz vorgestellt und bewertet werden:

VLC Streaming [42]

Bei dem *Video Lan Project* handelt es sich um eine Organisation, die einen quelloffenen Media Player entwickelt. Dieser ermöglicht es, den eigenen Desktop in hoher Qualität als Videostream zu exportieren. Der Test hat ergeben, dass die Simulation ohne Fragmentbildung und ruckelfrei übertragen wird. Der Gründe hierfür sind Buffering bei der Übertragung und Komprimierung des Videostreams. Sowohl ausgehend auf der Serverseite als auch eingehend beim Client, wird der Stream bearbeitet.
Mit rund 1,5 Sekunden Verzögerung bei der Übertragung ist der *VLC Mediaplayer* für diesen Zweck unbrauchbar.

RealVNC [29]

Bei *RealVNC* handelt es sich um eine Software zur Fernwartung, beziehungsweise Fernsteuerung, von PCs. Die Software ermöglicht also neben Darstellung des Linux Desktops auf dem Windows PC auch die Fernsteuerung dessen, wobei diese Funktion in diesem Kontext nicht benötigt wird. Tests haben ergeben, dass die Software zur Übertragung der Simulation in Echtzeit ungeeignet ist. Als problematisch einzustufen sind hierbei Probleme, wie unregelmäßige Verzögerungen von ein bis drei Sekunden bei der Übertragung und starke Fragmentbildung im Simulationsbereich. Somit ist diese Software für diesen Zweck unbrauchbar.

Teamviewer [35]

Ähnlich wie bei *RealVNC* handelt es sich auch bei *TeamViewer* um eine Software zur Fernwartung. Eine zusätzliche Funktion besteht darin, dass diese Software speziell für Onlinecollaboration in Echtzeit optimiert worden ist. Der einzige Flaschenhals der Software besteht in der Netzwerkanbindung. Da die beiden Rechner über ein GigE Netzwerk (1000 Mbit/s) miteinander verbunden sind, spielt Bandbreite eine zu vernachlässigende Rolle. Der Test der Software mit Hinblick auf diesen Anwendungsfall hat gezeigt, dass weder schlechte Qualität bei der Übertragung der Simulation, noch Ruckler im Videostream oder hohe Latenzen auftreten. Auf Grundlage dieser Argumentation wurde *Teamviewer* zur Übertragung des Videostreams genutzt.

Der Versuchsaufbau ist in Abbildung 3.12 dargestellt:

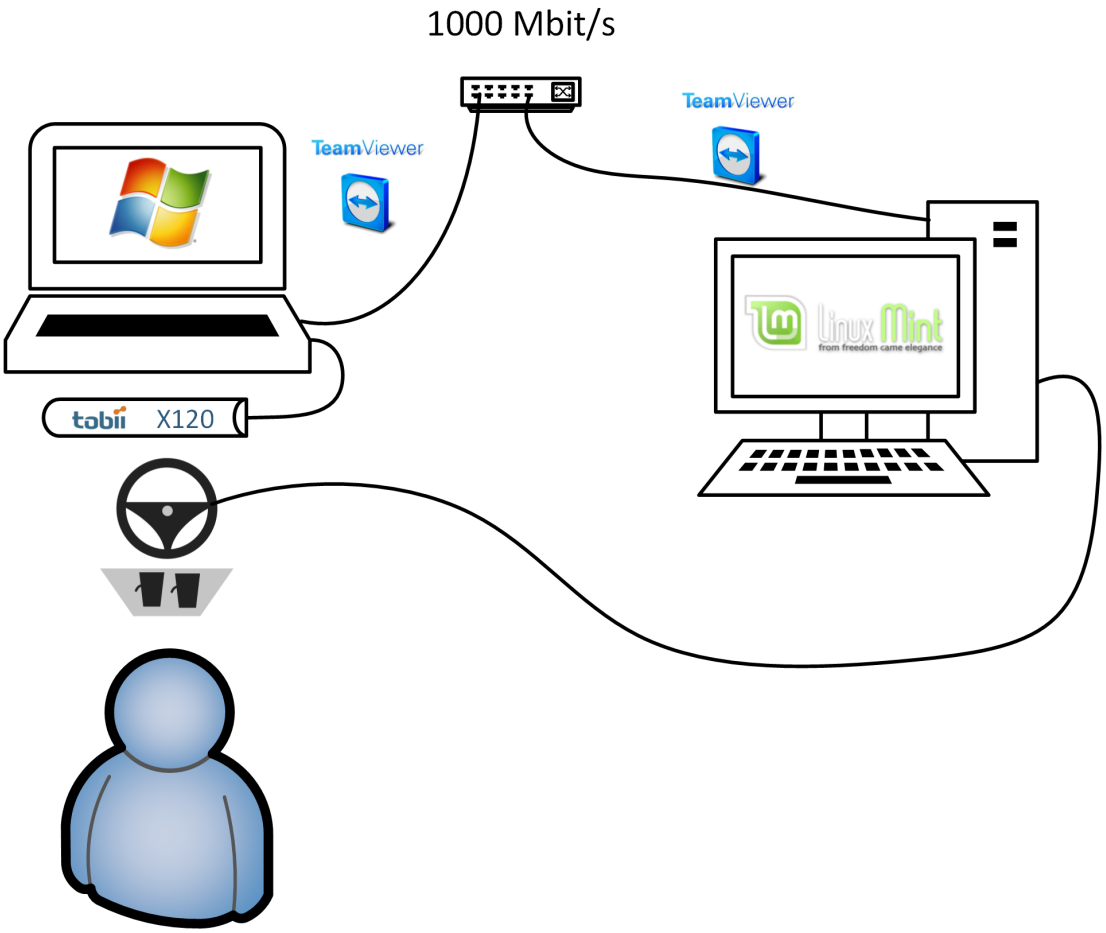

Abbildung 3.12.: Übersicht der Testumgebung

Der Versuchsperson wird als Interface ein *Logitech Momo Racing Steering Wheel* inklusive der dazugehörigen Pedale zur Verfügung gestellt. Da als Versuchspersonen lediglich Probanden mit Führerschein Klasse *B* oder höherwertig eingeladen werden, kann so eine intuitive Steuerung sichergestellt werden. Die Testfahrten werden zum einen von der Software selbst aufgezeichnet, um ein nachträgliches Abspielen zu ermöglichen und zum anderen wird der gesamte Test als Videomitschnitt abgespeichert. Das Video beinhaltet die Aufzeichnung des Bildschirminhalts, der sich vor dem User befindet. In dieses Video werden nach Ende der Aufnahme die Fixationspunkte und Sakkadensprünge als Overlay eingezeichnet. So kann den Sakkaden gefolgt und eine *Region of Interest* ermittelt werden. Die *Region of Interest* ist einer der Gründe für die Durchführung des Test und soll der Postulatsextraktion dienen, auf diese wird in Kapitel 3.5 erneut eingegangen.

3.4. Testszenarien

Wie bereits in Kapitel 3.2.4.1 beschrieben wurde, ist lediglich die Parametrierung der Strecke und der auftretenden Events notwendig. Um aussagekräftige Informationen über bestimmte Situationen erlangen zu können, ist eine detaillierte Planung der Testszenarien unumgänglich. Um eine Gruppierung von Events zu ermöglichen, bietet die Software die Möglichkeit, die gesamte Strecke in Teilabschnitte (Tiles) einzuteilen. Die regulären Tests beginnen nach einer kurzen Eingewöhnungsphase für die Versuchsperson. Nachfolgend wird zunächst das Ziel und der grundsätzliche Aufbau des jeweiligen Tests beschrieben.

3.4.1. Test 1 - Sakkadenmessung

Ziel

- Ermittlung der *Region of Interest* eines menschlichen Fahrers beim Steuern eines Fahrzeugs.
- Ab welcher Entfernung werden Hindernisse beachtet?
- Wird Hindernissen außerhalb der geplanten Trajektorie überhaupt Aufmerksamkeit geschenkt?

Versuchsdurchführung

Der Test ist in vier Teile gegliedert. Der erste Teil findet auf einer geraden Fahrbahn mit wenigen statischen Hindernissen statt. Der Zweite besteht aus einer kurvenreichen Fahrbahn mit wenigen Hindernissen. Bei Teil 3 handelt es sich wieder um eine gerade Fahrbahn mit ausschließlich dynamischen Hindernissen. Den Abschluss bildet der vierte Subtest mit einem kurvigen Streckenverlauf und ausschließlich dynamischen Hindernissen. Während des gesamten Tests fährt das Fahrzeug mit einer konstanten Geschwindigkeit. Der Versuchsperson wird mitgeteilt, das Fahrzeug mittig auf der Fahrbahn zu halten, aber in erster Linie Kollisionen zu vermeiden.

3.4.2. Test 2 - Messung der Sprungantwort

Ziel

- Messung der Sprungantwort

Versuchsdurchführung

Der zweite Test wird auf einer geraden Strecke durchgeführt. Die Versuchsperson bekommt den Auftrag, das Fahrzeug in der Mitte der mittleren Fahrspur zu halten. Während des gesamten Versuchs gibt es keinerlei Hindernisse. Die Geschwindigkeit wird im Laufe des Tests um 100% der Ausgangsgeschwindigkeit erhöht. Um die Sprungantwort und die Reaktionszeit messen zu können, wird die horizontale Position des Fahrzeugs aus Sicht der Versuchsperson willkürlich verändert. Die Versuchsperson wird entsprechend der Anweisung das Fahrzeug schnellstmöglich zurück in die Mitte der Fahrbahn steuern. Die zweite Komponente des Tests besteht darin zu ermitteln, wie schnell sich die Versuchsperson auf eine Änderung der Lenkradnullstellung einstellen kann.

3.4.3. Test 3 - Ermittlung von Ausweichtrajektorien

Ziel

- Ermittlung der Ausweichtrajektorie
- Wann beginnt die Versuchsperson mit dem Ausweichmanöver?

Versuchsdurchführung

Das primäre Ziel des dritten Tests besteht darin, die von den Versuchspersonen gewählten Ausweichtrajektorien und die Zeitpunkte zu denen mit dem Ausweichen begonnen wird, zu ermitteln. Ein sekundäres Ziel besteht in einem Vergleich mit dem ersten Test, um herauszufinden, ob sich ROI (Region of Interest) und Sakkadensprünge bei einer größeren Anzahl von Hindernissen ändern. Die Geschwindigkeit kann durch die Versuchsperson nicht verändert werden. Im Laufe des Tests ändert sich diese, um die Versuchsperson mit unterschiedlichen Geschwindigkeiten zu konfrontieren. Der Test wird einmal auf gerader und einmal kurviger Strecke durchgeführt.

3.4.4. Test 4 - Beschleunigungsverhalten

Ziel

- Ermittlung des Beschleunigungsverhaltens

Versuchsdurchführung

Im vierten Test kann die Versuchsperson selbstständig über die Geschwindigkeit des Fahrzeugs entscheiden, wobei diese lediglich durch eine Maximalgeschwindigkeit begrenzt wird. Es tauchen, aus Sicht der Versuchsperson, willkürliche Hindernisse vor dem Fahrzeug auf, die passiert werden müssen. Dieser Test soll zeigen, ob und wie stark die Geschwindigkeit durch die Versuchsperson in den auftretenden Situationen variiert wird.

3.5. Testergebnis

In diesem Kapitel werden die Ergebnisse zu den Tests des vorangegangenen Kapitels vorgestellt. Die Zuordnung der Testergebnisse findet im jeweiligen Abschnitt implizit statt. Die Argumentation wird durch Screenshots aus der Simulation und Diagrammen, basierend auf den Logfiles, gestützt. Basierend auf den Annahmen aus diesem Kapitel, wird in Kapitel 4.1 exemplarisch auf den Entwurf eines Reglers zur Spurhaltung bei einer horizontalen Verschiebung eingegangen. Die Tests wurden entsprechend dem Versuchsaufbau (siehe Abbildung 3.12) mit sechs Versuchspersonen (VP1-6) durchgeführt.

3.5.1. Ermittlung der Region of Interest

Zur Ermittlung der Region of Interest wurden für alle Versuchspersonen die vom Eyetracker aufgezeichneten Daten als akkumulierende Heatmap und als Clusteranalyse dargestellt. Als Hintergrund ist ein Screenshot der Simulation auf gerader Strecke gewählt worden, um zu ermitteln, welche Teile der Simulation von besonderer Relevanz sind. Bei einer akkumulierenden Heatmap handelt es sich um eine Analyseform, bei der die Häufigkeit von betrachteten Punkten die Farbe dieser Region am Ende bestimmt, wobei grün dafür steht selten, gelb mittelmäßig oft und rot häufig fokussiert worden zu sein. Die Clusteranalyse gruppiert fokussierte Punkte zu Clustern. Durch die Verbindung der Heatmap und der Clusteranalyse kann eine Region of Interest auf Grundlage der vorliegenden Testergebnisse gebildet werden. Nachfolgend werden die Heatmaps und Clusteranalysen, sortiert nach Versuchspersonen, dargestellt:

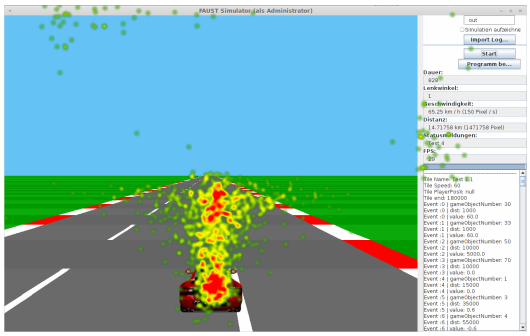

Abbildung 3.13.: Akkumulierende Heatmap von Versuchsperson 1

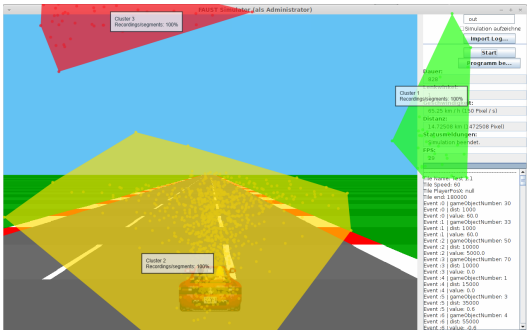

Abbildung 3.14.: Clusteranalyse von Versuchsperson 1

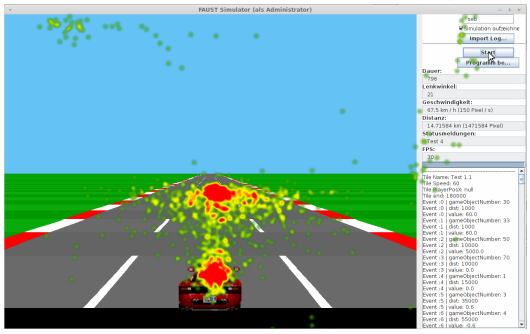

Abbildung 3.15.: Akkumulierende Heatmap von Versuchsperson 2

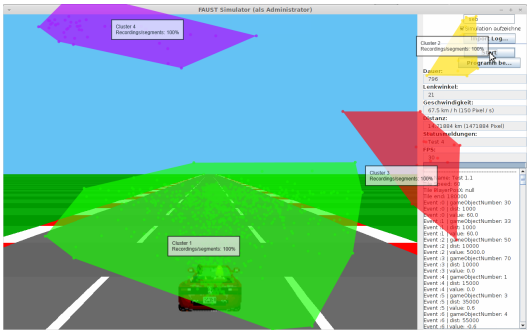

Abbildung 3.16.: Clusteranalyse von Versuchsperson 2

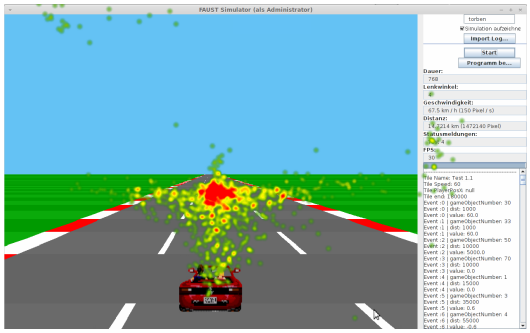

Abbildung 3.17.: Akkumulierende Heatmap von Versuchsperson 3

Abbildung 3.18.: Clusteranalyse von Versuchsperson 3

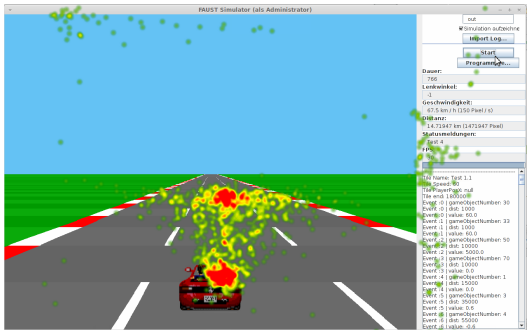

Abbildung 3.19.: Akkumulierende Heatmap von Versuchsperson 4

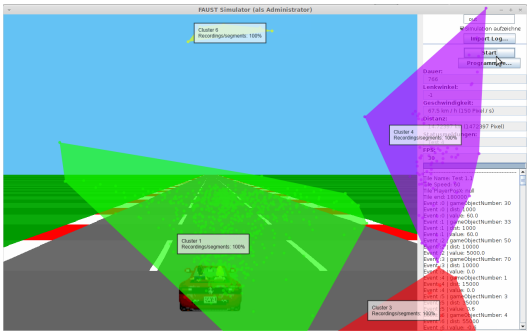

Abbildung 3.20.: Clusteranalyse von Versuchsperson 4

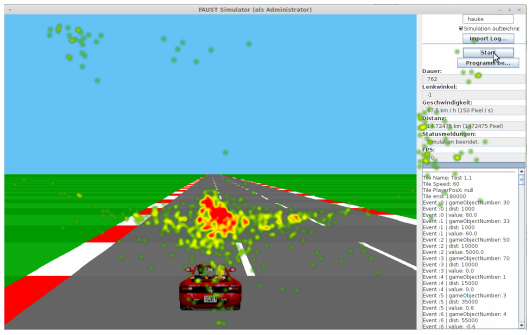

Abbildung 3.21.: Akkumulierende Heatmap von Versuchsperson 5

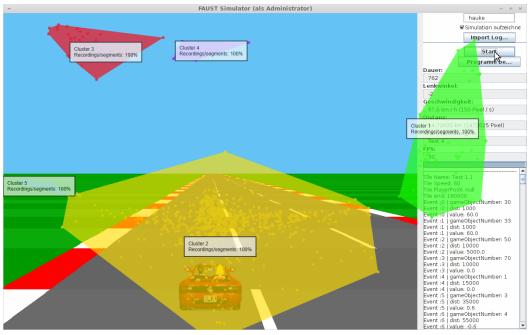

Abbildung 3.22.: Clusteranalyse von Versuchsperson 5

Abbildung 3.23.: Akkumulierende Heatmap von Versuchsperson 6

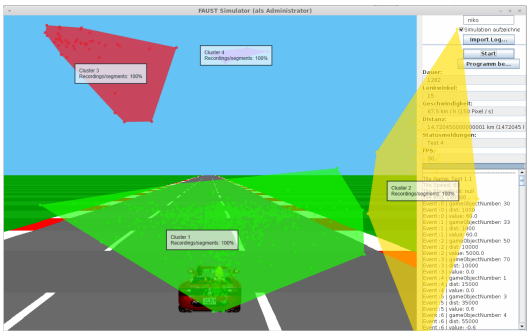

Abbildung 3.24.: Clusteranalyse von Versuchsperson 6

Insgesamt lässt sich bei der Betrachtung der Heatmaps sagen, dass alle sechs Versuchspersonen ein sehr ähnliches Verhalten aufweisen. Weiter ist auffällig, dass alle Versuchspersonen ihr Hauptaugenmerk auf den Bereich direkt vor dem Auto gelegt haben.

Die Form der Heatmap ähnelt einem stark abstrahierten *T*, wobei die Länge des Schafts, ausgehend vom Fahrzeug und in Fahrtrichtung liegend, stark variiert. Die Ausprägung des Querstrichs ist ebenfalls sehr unterschiedlich. Den Versuchspersonen wurde vor Beginn des Tests Auskunft über die Länge einer weißen Fahrbahntrennungslinie (6 Meter) und der Distanz zwischen Selbigen (6 Meter) gegeben. Während die Versuchspersonen 1 - 4 primär den Bereich in einer Entfernung von 6-7 weißen Linien (72 bis 84 Meter) vor dem Fahrzeug am intensivsten betrachtet haben, haben die Versuchspersonen 5 und 6 den Bereich in einer Entfernung von 2-3 weißen Linien (24 bis 36 Meter) gewählt.
Die von der Software generierten Cluster zeigen, in welche Bereiche die Software aus Anwendersicht eingeteilt ist und wie groß diese sind. Mit dem Wissen, dass im oberen Simulationsbereich Anweisungen eingeblendet werden, erklärt sich dieser Cluster. Weiter gibt es eine variierende Anzahl von Clustern im Menü- und Informationsbereich (rechts). Der größte Cluster befindet sich bei allen Versuchspersonen auf der simulierten Fahrbahn.

Für beide Analyseansätze kann eine Region of Interest bestimmt werden:

Abbildung 3.25.: Heatmap-basierte Region of Interest

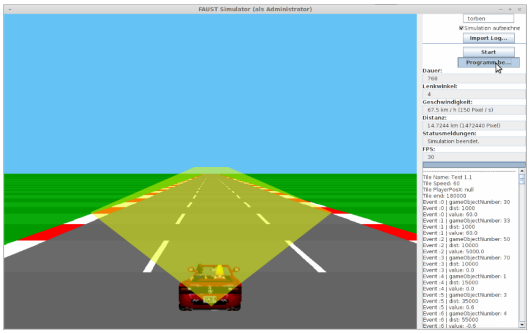

Abbildung 3.26.: Clusteranalyse-basierte Region of Interest

Während in Abbildung 3.25 die Positionierung und Form einer möglichen Region of Interest auf Grundlage der Heatmap hergeleitet wurde, ist dieses in Abbildung 3.26 auf Basis der Clusteranalyse durchgeführt worden.

3.5.2. Messung der Sprungantwort

Zur Messung der Sprungantwort wurden die Logfiles von Testszenario 2 (siehe Kapitel 3.4.2 ausgewertet. Das Ziel in diesem Test bestand darin das Fahrzeug in der Mitte der Fahrbahn zu halten. Hierzu wurden die horizontale Position des Fahrzeugs auf der Fahrbahn und die Lenkradnullstellung verändert. Das nachfolgend dargestellte Diagramm trägt horizontal die zurückgelegte Distanz (in Pixeln) relativ zum Beginn des Tests gegen die horizontale Position

(oben, entsprechend Definition 3.20) beziehungsweise den Lenkwinkel (unten, in Prozent) aller sechs Versuchspersonen auf:

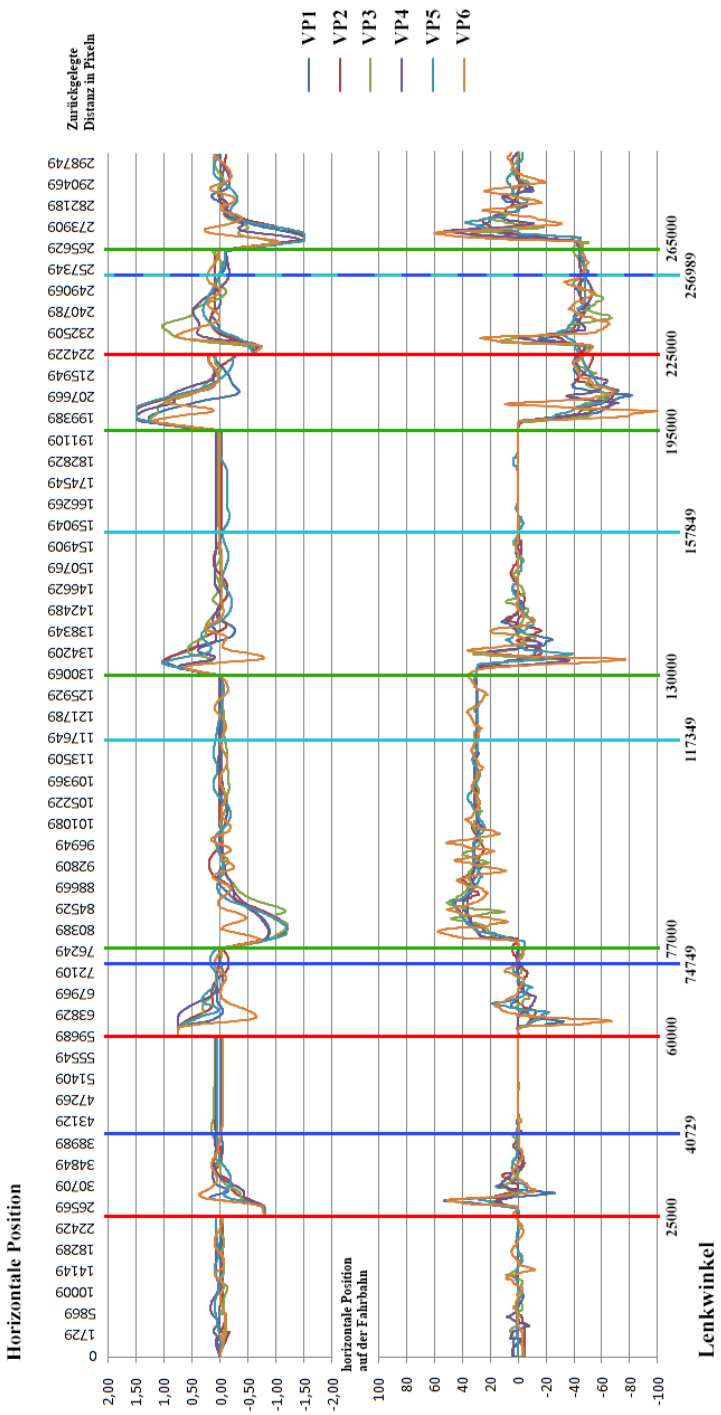

Abbildung 3.27.: Diagramm, mit horizontal aufgetragener zurückgelegter Distanz und vertikal aufgetragener Position auf der Fahrbahn (oben) beziehungsweise des prozentualen Lenkwinkels (unten)

Die roten vertikalen Linien markieren den Moment, in dem die horizontale Position des Fahrzeugs verändert wird. Die dunkelblauen Linien markieren den Moment, zu dem 83,3% der Versuchspersonen das Fahrzeug wieder stabil und zentral in der Mitte der Fahrbahn (Maximum der Amplitude größer als -0,1 und kleiner als +0,1) halten. Die vertikalen grünen Linien markieren Änderungen der Lenkradnullstellung und dazugehörige hellblaue Linien markieren den eingeschwungenen Zustand in der Mitte der Fahrbahn.

Die Geschwindigkeit beträgt zu Beginn des Tests 60 Pixel pro Sample, mit Überschreiten der 160.000 Pixel Grenze, verdoppelt sich die Geschwindigkeit auf 120 Pixel pro Sample. Auf Grundlage dieses Wissens und den Momenten, zu denen die Teilnehmer wieder zur Mitte der Fahrbahn zurückgekehrt sind, kann die Dauer des Einschwingvorgangs beim Versetzen des Fahrzeugs bestimmt werden. Hierfür ergibt sich bei 30 Frames pro Sekunde die mittlere benötigte Zeit des Einschwingvorgangs:

$$40729 - 25000 = 15729 \tag{3.26}$$

$$\frac{15729}{30 * 60} \approx 8,74 Sekunden \tag{3.27}$$

$$74749 - 60000 = 14749 \tag{3.28}$$

$$\frac{14749}{30 * 60} \approx 8,19 Sekunden \tag{3.29}$$

$$256989 - 225000 = 31989 \tag{3.30}$$

$$\frac{15729}{30 * 120} \approx 8,89 Sekunden \tag{3.31}$$

In allen drei Fällen benötigen die Versuchspersonen zwischen acht und neun Sekunden zur Korrektur der Position, was darauf schließen lässt, dass der Regelalgorithmus des Menschen unabhängig von der Geschwindigkeit ist. Während die ersten beiden Korrekturen bei 27 km/h und ohne Lenkoffset durchzuführen waren, betrug die Geschwindigkeit bei der dritten Korrektur 54 km/h und der Lenkoffset -30°. In Kapitel 4.1 wird darauf eingegangen, einen Regelalgorithmus zu entwerfen, der dieses Verhalten nachbildet.

Weiter kann durch Auswertung der Zeit bis zur ersten Reaktion auf die Verschiebung des Fahrzeugs eine Aussage über die mittlere Reaktionszeit der Versuchspersonen gemacht werden:

$$26029 - 25000 = 1029 \tag{3.32}$$

$$\frac{1029}{30 * 60} \approx 0,57 Sekunden \tag{3.33}$$

$$61009 - 60000 = 1009 \tag{3.34}$$

$$\frac{1009}{30 * 60} \approx 0,56 Sekunden \tag{3.35}$$

$$226869 - 225000 = 1869 \tag{3.36}$$

$$\frac{1869}{30 * 120} \approx 0,52 Sekunden \tag{3.37}$$

Mit einer Reaktionszeit von rund 0,5 Sekunden entspricht das Messergebnis in etwa den 550 - 630 ms die von Triggs und Harris [37] als Reaktionszeit des Menschen im Straßenverkehr angegeben wurden.

3.5.3. Ermittlung der Ausweichtrajektorie

Zur Ermittlung der Ausweichtrajektorien bei statischen und dynamischen Hindernissen, wurden zunächst die aufgezeichneten Videosequenzen des Eyetrackers dahingehend untersucht, zu welchem Zeitpunkt ein Hindernis für die Versuchspersonen relevant wird.
Es hat sich gezeigt, dass 84% aller Hindernisse (auch außerhalb der tatsächlich gefahrenen Trajektorie) erstmalig in einer Entfernung von 6 - 7 weißen Fahrbahntrennungslinie fokussiert werden. Diese Entfernung entspricht der in 3.5.1 angesprochenen Distanz, der vor dem Auto die größte Aufmerksamkeit gewidmet wird. Dies gilt sowohl in Kurven, als auch auf Geraden.
Alle Versuchspersonen haben den Ausweichvorgang möglichst früh eingeleitet (siehe Abbildung 3.28). Die dabei abgefahrene Trajektorie war, sehr seicht. Ausnahmefälle bestanden dabei in Reaktionen auf Kollisionen oder einem anderen Verhalten des Fahrzeugs als durch die Versuchsperson angestrebt war.
Der Unterschied zwischen statischen und dynamischen Hindernissen war immens. Während die Anforderung an die Versuchsperson bei statischen Hindernissen eher gering war (bezogen auf die Anzahl der Fixationen auf das Hindernis), waren bewegte Hindernisse deutlich anspruchsvoller und wurden durchschnittlich sechs Mal häufiger fixiert. Bis auf einige wenige Fälle war das Vorbeifahren an beiden Typen von Hindernissen kein Problem und gelang beim ersten Versuch zu über 90%.

Die nachfolgende Visualisierung zeigt die horizontale Position des Fahrzeugs auf der Fahrbahn, während des ersten Tests. Die roten Punkte markieren die Positionen von Hindernissen auf der Fahrbahn, unterhalb des Diagramms werden die genauen Positionen den Hindernisse zugeordnet:

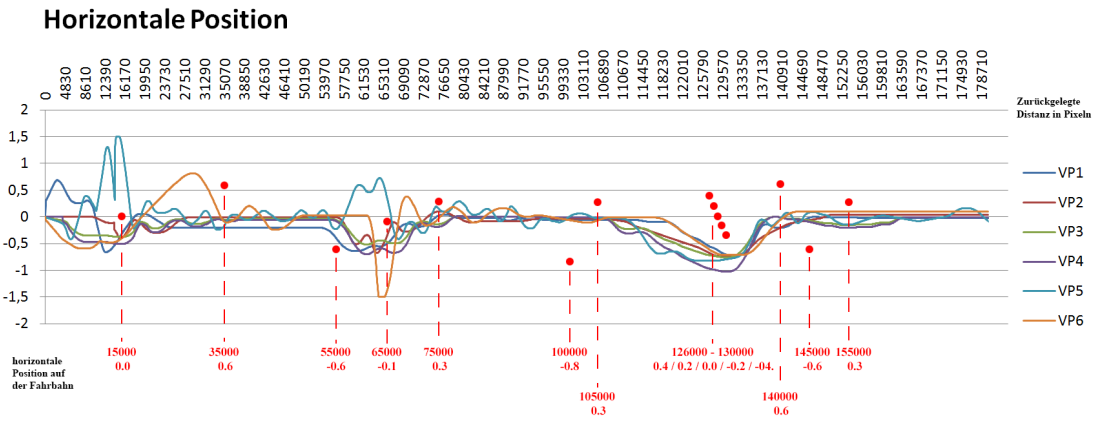

Abbildung 3.28.: Verlauf der horizontalen Position des Fahrzeugs auf der Fahrbahn beim Vorbeifahren an Hindernissen (rote Punkte)

In dem Diagramm ist zu erkennen, dass sich die Versuchspersonen bei den ersten beiden Hindernissen zunächst an die Steuerung des Fahrzeugs gewöhnen mussten und sehr extreme Lenkmanöver durchführen, um nicht mit den Hindernissen (bei 15000 Pixel und 65000 Pixel) zu kollidieren. Insgesamt sind auf diesem Teilstück drei Hindernisse, denen die Versuchspersonen ausweichen müssen, da sonst eine Kollision droht. Die übrigen Hindernisse dienten dem Zweck, die Aufmerksamkeit der Versuchspersonen auf sich zu ziehen. Beim dritten Hindernis (126000 Pixel), das eine Änderung der Position auf der Straße notwendig macht, handelt es sich um eine Kette aus fünf Hindernissen, die einen großen Teil des befahrbaren Bereichs belegen. Alle Versuchspersonen haben bereits frühzeitig mit der Korrektur der Position begonnen. Während die erste Korrektur von *VP4* bereits bei einer Pixeldistanz von 106650 Pixel eingeleitet wurde, hat *VP6* hiermit bis 117030 Pixel gewartet. Die durchschnittliche Distanz bis zur Reaktion auf dieses Hindernis liegt bei 110290. Somit ergibt

sich als mittlere Ausweichzeit vor Erreichen des Hindernisses für eine Geschwindigkeit von 27 km/h:

$$126000 - 110290 = 15710 \tag{3.38}$$

$$\frac{15710}{30 * 60} \approx 8,73\ Sekunden \tag{3.39}$$

Ein Zusammenhang zwischen diesem Ergebnis und den Zeiten aus Kapitel 3.5.2 konnte trotz der gleichen Größenordnung nicht hergestellt werden. Fakt ist allerdings, dass sich der Fixationspunkt entsprechend dem angepeilten Ziel verschiebt, wie es in der nachfolgenden Abbildung exemplarisch zu sehen ist und von Land und Lee in *Where we look when we steer* [24] beschrieben wird.

Abbildung 3.29.: Der Fixationspunkt verschiebt sich entsprechend der Trajektorie

Durch die Anpassung der Lenkradstellung ändert sich die horizontale Position auf der Fahrbahn. Der Autofahrer hat genau wie die Versuchsperson in der Simulation die Möglichkeit selbst zu entscheiden, wie intensiv und wann die Änderung der horizontalen Position durchgeführt wird. Von den Versuchspersonen wurden zwei verschiedene Trajektorien für die Ausweichvorgänge gewählt:

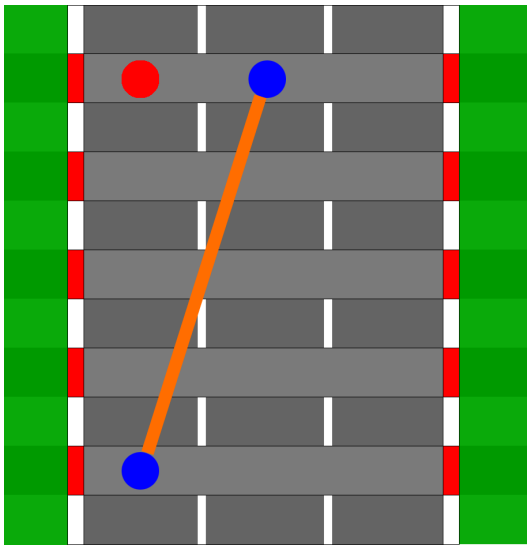

Abbildung 3.30.: Ausweichen mit konstanter Veränderung der horizontalen Position (bekannt als Follow the Carrot [25])

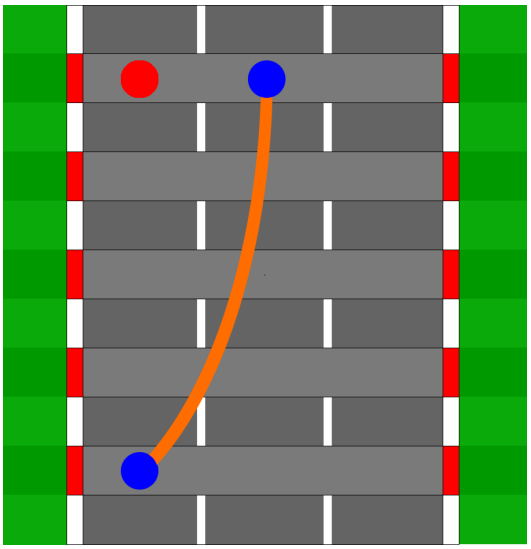

Abbildung 3.31.: Degressives Ausweichen mit stetig schwächer werdendem Lenkwinkel

In blau ist die aktuelle Position (unten) und die angestrebte Position (oben) eingezeichnet. Das Ziel besteht darin, dem roten Hindernis auszuweichen.

Neben diesen gibt es weitere Verfahren zur Bestimmung von Trajektorien aus dem Forschungsbereich der Bahnplanung. Zu den bekanntesten biologisch motivierten Algorithmen gehört Pure Pursuit [11]. Hier wird eine Kreisbahn vom aktuellen Standpunkt zum Zielpunkt berechnet, die abgefahren wird:

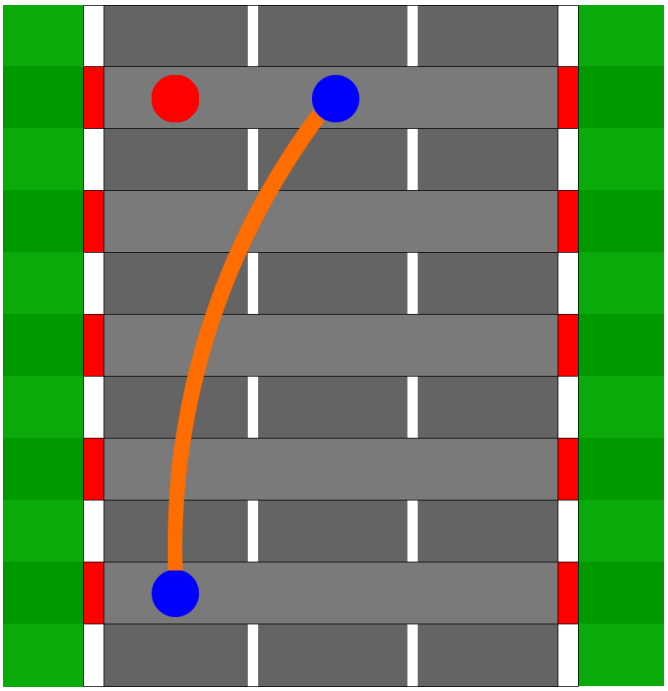

Abbildung 3.32.: Ausweichen mit Pure Pursuit nach Coulter [11]

3.5.4. Ermittlung des Beschleunigungsverhaltens

Alle Versuchspersonen strebten im gesamten Verlauf des vierten Tests die Maximalgeschwindigkeit an. Erst in Situationen mit erhöhtem Anspruch (sehr breite Hindernisse oder gehäuftes Auftreten von Hindernissen in Kurven) wurde die Geschwindigkeit von allen Versuchspersonen kurzzeitig verringert.
Auf dieser Grundlage können schwerlich weitere Informationen über den menschlichen Regler extrahiert werden, als dass eine Verringerung der Geschwindigkeit mit zunehmender Komplexität einer Situation notwendig ist.

3.5.5. Statistische Auswertung

Während der Tests mit dem Eyetracker konnten statistische Informationen gesammelt werden, die nachfolgend dargestellt werden. Die Dauer der Tests und die Rate erfolgreich gemessener Augenpositionen durch den Eyetracker variiert stark. Die Gründe hierfür bestehen darin, dass neben Kollisionen und unterschiedlicher Geschwindigkeit im letzten Abschnitt der Simulation auch die Zeit der Kalibrierung, sowie die Zeit zur Klärung finaler Fragen vor Beginn des Tests

mit in die Statistik eingehen. Die Diskrepanz bei den erfolgreichen Samples durch den Eyetracker ist durch zu starke Bewegungen der Versuchsperson, Blinzeln oder die Änderungen der Sitzposition zu erklären. Die nachfolgende Tabelle ordnet die zuvor beschriebenen Faktoren den Versuchspersonen zu:

Versuchsperson	Dauer des Tests	erfolgreiche Samples
VP1	12:59	80%
VP2	13:02	88%
VP3	12:20	86%
VP4	12:06	91%
VP5	12:14	96%
VP6	12:34	96%

Alle nachfolgenden Diagramme wurden auf Grundlage der Daten des Eyetrackers erzeugt und basieren darauf, dass zwei verschiedene AOIs (Area of Interest) definiert worden sind. Wobei *VP1-Alles* (dunkelblau) den gesamten Bereich des Bildschirms und *VP1* (cyan) den Bereich der Fahrbahn abdeckt, der laut Clusteranalyse (siehe 3.5) von primärem Interesse ist. Die nachfolgende Abbildung visualisiert den zuvor beschriebenen Sachverhalt:

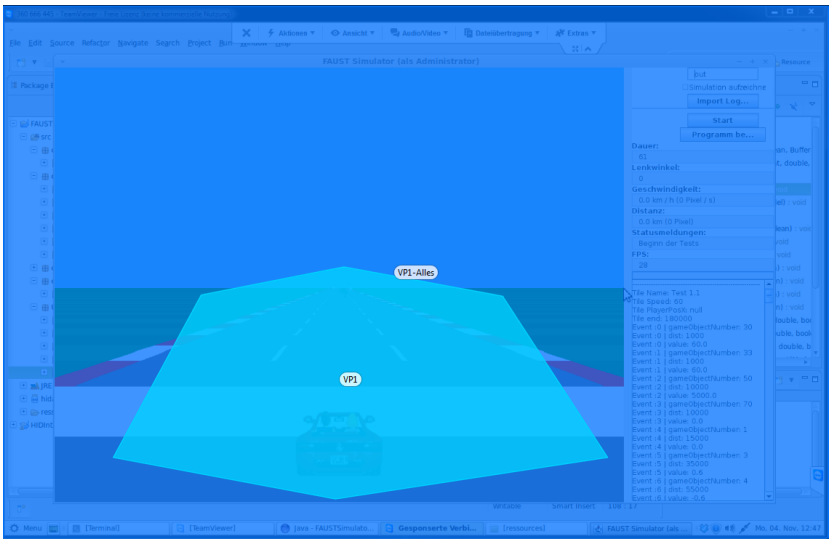

Abbildung 3.33.: **A**re **o**f **I**nterest

Die Diagramme 3.34 und 3.35 visualisieren die durchschnittliche und maximale Fixationszeit der Versuchspersonen:

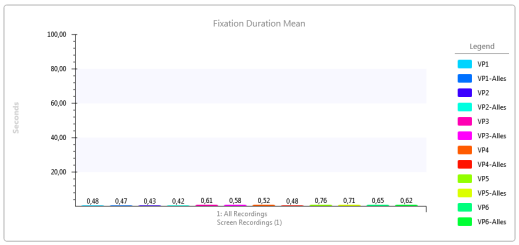

Abbildung 3.34.: Durchschnittliche Fixationszeit

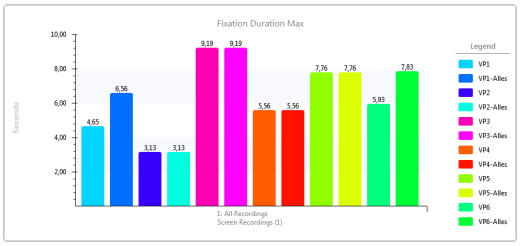

Abbildung 3.35.: Maximale Fixationszeit

Die minimale Fixationszeit konnte aufgrund der geringen Samplerate des verwendeten Eyetrackers nicht bestimmt werden. Diagramm 3.36 zeigt, wie lange die jeweilige Area of Interest insgesamt fixiert worden ist.

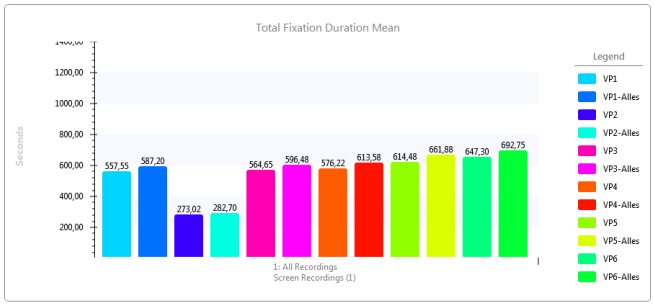

Abbildung 3.36.: Gesamte Fixationszeit

Das letzte Diagramm zeigt, welche Versuchsperson wie häufig die jeweilige Area of Interest fixiert hat:

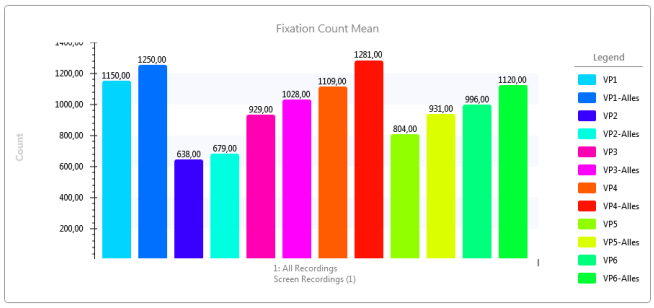

Abbildung 3.37.: Gesamtanzahl Fixationen

Bei der Betrachtung aller Diagramme sollte im Hinterkopf behalten werden, dass gilt:

$$VPn \subseteq VPn - Alles \tag{3.40}$$

$$mit\ n := \{n \in \mathbb{N}\,|1 \leq n \leq 6\} \tag{3.41}$$

4. Reglerentwurf

Dieses Kapitel beschäftigt sich in Abschnitt 4.1 mit dem Entwurf, der Parametrierung und Implementierung von Regelalgorithmen für einen Spezialfall der Simulation. In Abschnitt 4.2 wird auf die Implementierung eines neuen Informationsextraktion-Algorithmus zur Steuerung der Carolo-Cup Plattform des FAUST Projekts eingegangen. Das entwickelte Verfahren (nachfolgend als *Tracing* bezeichnet) wurde auf Grundlage des aus Kapitel 3.5 erlangten Wissens implementiert.

4.1. Reglerentwurf

In Abbildung 3.27 aus Kapitel 3.5.2 ist das Schwingungsverhalten des menschlichen Reglers abgebildet. Zu sehen ist die von den Versuchspersonen durchgeführte Korrektur der horizontalen Verschiebungen des Fahrzeugs auf der Fahrbahn und die Reaktion auf eine Änderung der Lenkradnullstellung. Auf Basis dieses Diagramms sollen verschiedene Regelalgorithmen entwickelt werden, um die horizontale Position des Fahrzeugs zu korrigieren. Das Ziel besteht hier nicht darin, einen perfekten Regler zu entwickeln, sondern das Schwingungsverhalten möglichst nahe an den Ergebnissen der durchgeführten Tests auszurichten.

Ein ähnliches Schwingungsverhalten konnte für PID- und PT2-Regler in *Grundlagen der Regelungstechnik* [19] gefunden werden. An dieser Stelle soll nicht weiter auf die Begründung dieser Auswahl oder die Eigenschaften der Regler eingegangen werden, da dieses kein Bestandteil dieser Masterarbeit sind. Im nachfolgenden Bild ist das dazugehörige Simulink Schaltbild abgebildet:

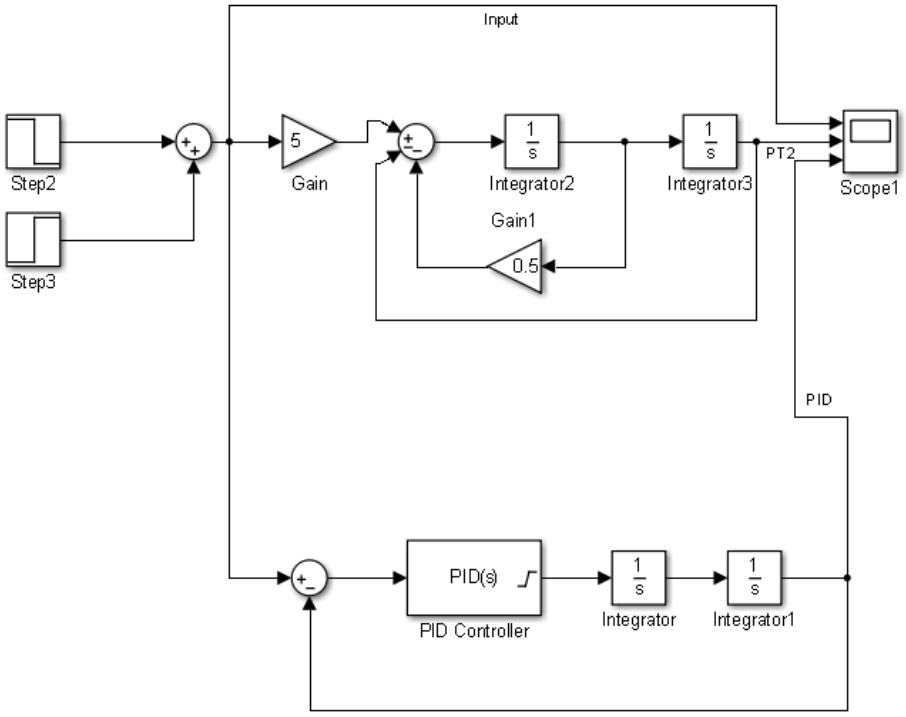

Abbildung 4.1.: Simulink-Schaltbild einer PT2- und einer PID-Regelstrecke

Der Input ist ein 1 Takt langer Peak auf Wert von *-0.8*, was genau der Störung entspricht, die bei der ersten roten Linie in Abbildung 3.27 wirkt. Der PID-Regler wurde um zwei zusätzliche Integratoren erweitert, um das Schwingungsverhalten anzupassen. In der folgenden Tabelle sind die Parameter des PID-Reglers abgebildet:

Parameter	Wert
Proportional (P)	2.5
Integral (I)	0.5
Derivative (D)	1
Filter coefficient (N)	2

Das Ergebnis der Simulation ist im nachfolgenden Scopeplot zu sehen:

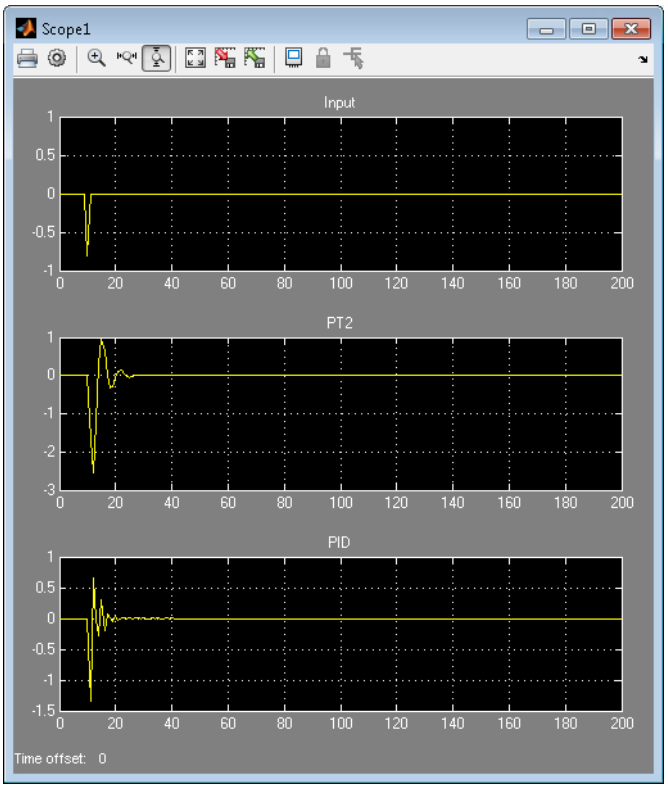

Abbildung 4.2.: Ergebnis der parametrierten PT2- und PID-Regler im Simulink-Scopeplot

Die Regler wurden gegen ein spezielles Interface implementiert, welches es ermöglicht, Geschwindigkeit und Lenkwinkel für jedes Sample vorzugeben. Das Interface stellt relevante Informationen, wie die aktuelle Geschwindigkeit, das im Simulationsfenster dargestellte Bild und die für diesen Fall essentielle horizontale Position des Fahrzeugs auf der Fahrbahn zur Verfügung.

4.1.1. Evaluation

Das Ergebnis der beiden in Java implementierten und an das Interface der Simulation angeschlossenen Regelalgorithmen ist in den Abbildungen 4.3 und 4.4 zu sehen.

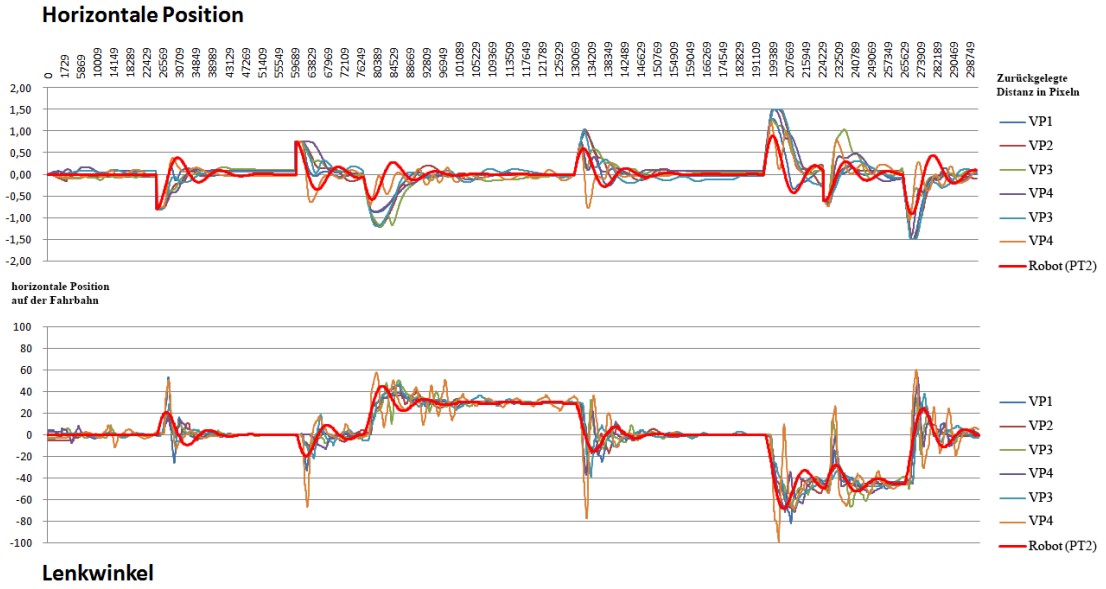

Abbildung 4.3.: In rot die vom PT2-Regler gefahrene Trajektorie über der der Versuchspersonen

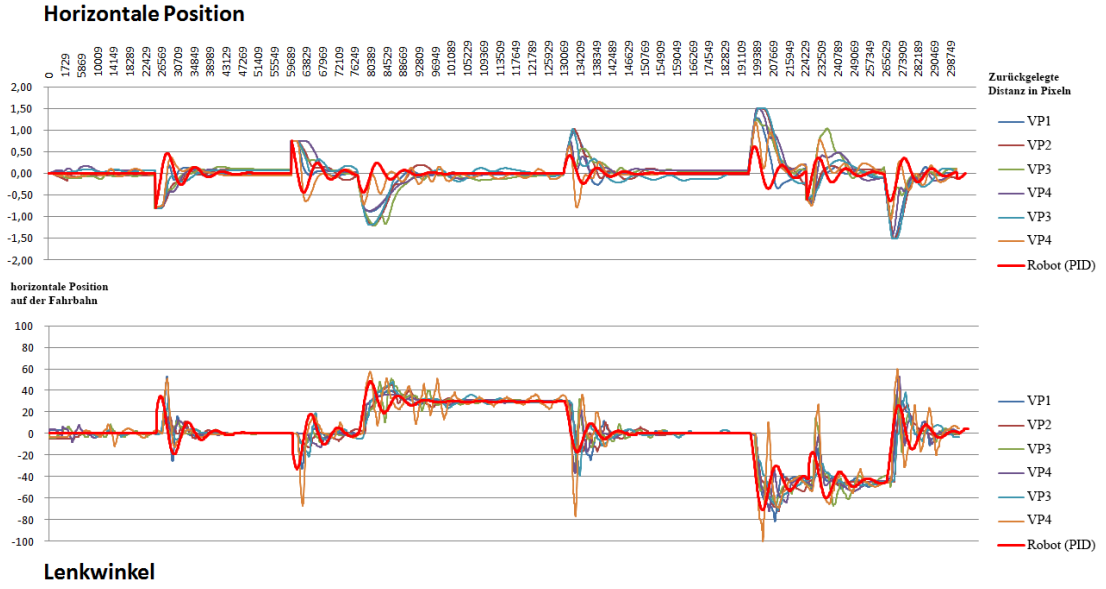

Abbildung 4.4.: In rot die vom PID-Regler gefahrene Trajektorie über der der Versuchspersonen

Beide Regler bilden ein sehr ähnliches Verhalten ab, welches dem der Versuchspersonen weitestgehend entspricht. Der größte Unterschied besteht darin, dass die Regelalgorithmen sofort auf den Einheitssprung reagieren, während die Versuchspersonen rund eine halbe

Sekunde bis zur ersten Reaktion benötigen (siehe Kapitel 3.5.2). Auch hier gilt das System als eingeschwungen, wenn das Maximum der Amplitude (Änderung der horizontalen Position) größer als -0,1 und kleiner als +0,1 ist.

Die beiden zuvor beschriebenen Regler sind für einen Spezialfall entwickelt worden. Die Anforderungen an diese Regelalgorithmen waren die gleichen, wie die mit denen die Versuchspersonen in Testszenario 2 (siehe Kapitel 3.4.2) konfrontiert wurden. Wenn die gesamte Simulation von einem Agenten befahren werden soll, sind viele weitere Dinge notwendig. Dazu zählen Programmabschnitte, die die Geschwindigkeitsregelung für das vierte Testszenario übernehmen oder eine Hinderniserkennung und die dazugehörige automatische Bestimmung von Ausweichtrajektorien. Die Implementierung eines solchen Gesamtsystems übersteigt den Umfang dieser Arbeit, ist allerdings auf Grundlage der vorhandenen Forschungsergebnisse umsetzbar.

4.2. Tracing-Algorithmus

Der nachfolgend beschriebene Algorithmus ist ein Teil der Software, welche zur Regelung der Carolo-Cup Plattform (siehe Kapitel 2.1) eingesetzt werden kann. Bei der verwendeten Subsumption-Architektur [3][4] des FAUSTcore, werden Plugins mit verschiedenen Prioritäten geladen. Die Prioritäten dieser Plugins geben die Abarbeitungsreihenfolge an. Die so geschaffene Möglichkeit, Abhängigkeiten zwischen verschiedenen Tasks zu modellieren wird zusätzlich durch die Option der Interaktion von Tasks untereinander durch die Verwendung der vom FAUSTcore bereitgestellten Datencontainer unterstützt.
Das in diesem Projekt überarbeitete Modul bekommt in der Regel die höchste Priorität, da die meisten anderen Module auf die Daten der Fahrspurerkennung (Polaris [14]) zugreifen und diese für eigene Zwecke nutzen, wie es beispielsweise die Hindernis- und Kreuzungserkennung [44] macht.

Nachfolgend werden die fünf Schritte des implementierten Algorithmus detailliert beschrieben. Hierfür gelten zunächst folgende Bedingungen:

Die Kamera stellt ein Bild *B* als Matrix mit *M* Zeilen und *N* Spalten, also *M* * *N* Pixel zur Verfügung.

$$B = [B_{x_1,x_2}] \quad mit\ x_1 \mid 0 \leq x_1 \leq M \tag{4.1}$$

$$und\ x_2 \mid 0 \leq x_2 \leq N \tag{4.2}$$

$$wobei\ M = 479 \tag{4.3}$$

$$N = 751 \tag{4.4}$$

Wobei jeder Bildpunkt einen Grauwert *g* hat.

$$g(B_{x_1,x_2}) \ \in \ \{0...255\} \tag{4.5}$$

Durch Kopplung der projektiven Transformation T mit der Verzeichnungskorrektur V für das verwendete Objektiv kann eine Transformation aus dem Bildkoordinatensystem K_B in das Weltkoordinatensystem K_W durchgeführt werden. Eine Rücktransformation T^{-1} mit anschließender Verzeichnung V^{-1} transformiert entsprechend von K_W nach K_B. So kann durch Angabe von Weltkoordinaten in nachfolgender Gleichung auf den Grauwert eines Pixel des Bildes mit den Koordinaten y_1, y_2 zugegriffen werden:

$$g(V^{-1}(T^{-1}(y_1, y_2))) \tag{4.6}$$

Abbildung 4.5 zeigt eine Übersicht der zuvor beschrieben Zusammenhänge anhand des kompletten Kamerabildes (links) und des relevanten Ausschnitts des in Weltkoordinaten transformierten Bildes (rechts). Die Maßeinheit des linken Koordinatensystems ist ganze Pixel, die des rechten Koordinatensystems ist cm.

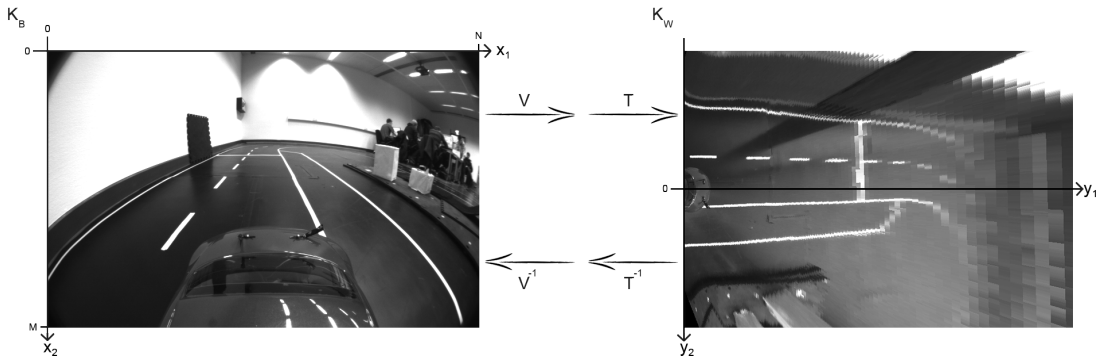

Abbildung 4.5.: Nomenklatur und Transformationsübersicht für Bild- und Weltkoordinatensystem

Schritt 1: Profile suchen

Sobald das Modul vom Scheduler des FAUSTcore gestartet wird, besteht der erste Handlungsschritt für den Algorithmus darin, nach Profilen zu suchen. Als Profil wird nachfolgend ein Kandidat für eine Fahrbahnbegrenzung bezeichnet. Ein Profil ist eine weiße 1 Pixel hohe Linie in K_B, die nach links und rechts durch signifikante Unterschiede $\geq\ threshold = 35$ in $g()$ zweier benachbarter Pixel begrenzt wird.
Es wird zunächst unterschieden, ob es zuvor einen erfolgreichen Tracing-Versuch (Verfolgung von Fahrbahnbegrenzungen) gegeben hat oder nicht. Wenn nicht, werden in K_W Start- und Endpunkt einer Linie wie nachfolgend beschrieben bestimmt und nach K_B transformiert:

$$P_1 = (15, -95) \tag{4.7}$$

$$P_2 = (15, \quad 35) \tag{4.8}$$

$$Q_1 = V^{-1}(T^{-1}(P_1)) \tag{4.9}$$

$$Q_2 = V^{-1}(T^{-1}(P_2)) \tag{4.10}$$

Die fest gewählten Werte von -95 cm und 35 cm wurden durch Tests an maximal steilen Kurven ermittelt. Sollte es zuvor einen erfolgreichen Tracing-Versuch gegeben haben, wird für

jedes valide Profil, also jede korrekt erkannte Fahrbahnmarkierung des vorhergehenden Zyklus, das zuvor beschriebene Verfahren für jedes Profil wie folgt abgewandelt:

$$P_{alt} = (p_{y_1}, p_{y_2}) \tag{4.11}$$

$$\tau = 10 \tag{4.12}$$

$$P_1 = (15, (p_{y_2} - \tau)) \tag{4.13}$$

$$P_2 = (15, (p_{y_2} + \tau)) \tag{4.14}$$

$$Q_1 = V^{-1}(T^{-1}(P_1)) \tag{4.15}$$

$$Q_2 = V^{-1}(T^{-1}(P_2)) \tag{4.16}$$

Hierfür wird der erste Punkt P_{alt} jedes Profils herangezogen. Die zweite Komponente p_{y2} dieses Punktes wird um τ inkrementiert beziehungsweise dekrementiert. Der Wert von 10 cm für τ hat sich als gut erwiesen, weil sich bei korrektem Betrieb die Fahrbahnbegrenzung niemals um mehr als 10 cm verschiebt. Des Weiteren ist diese Toleranz bei einer Fahrspurbreite von 40 cm und 30 cm breitem Parkstreifen unkritisch. Abbildungen 4.6 und 4.7 visualisieren das zuvor geschilderte:

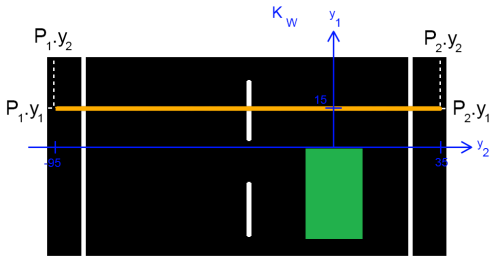

Abbildung 4.6.: Schritt 1 - Initialzustand

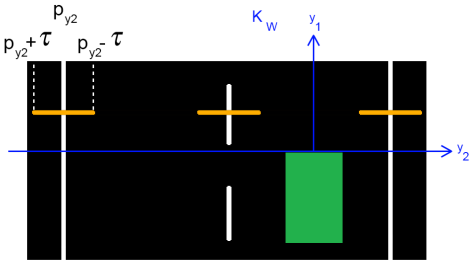

Abbildung 4.7.: Schritt 1 - Positionierung der Profilsuchlinien nach vorherigem erfolgreichen Durchlauf

Zwischen den Punkten Q_1 und Q_2 wird in K_B mit nachfolgendem in Pseudocode beschriebenen Verfahren nach Profilen gesucht:

Data : Q_1, Q_2
Result : Profilliste
threshold = 35;
for $i = Q_1.y_2 \rightarrow Q_2.y_2$ **do**
  if $abs(g(Q_1.y_1, i) - g(Q_1.y_1, i+1)) \geq threshold$ **then**
    for $j = i+1 \rightarrow Q_2.y_2$ **do**
      if $abs(g(Q_1.y_1, i) - g(Q_1.y_1, i+1)) \geq threshold$ **then**
        center = $(i+1) + ((j-(i+1))/2)$;
        Profilliste.pushback($Point_{Bild}$($Q_1.y_1$,center));
        break;
      end
      j+1;
    end
  end
  i+1;
end

Pseudocode 1 : Profilsuche

Der Pseudocode 1 wird für jedes Punktepaar Q_1, Q_2 durchgeführt und liefert eine Liste von möglichen Profilen.
Wird aufgrund schlechter Bedingungen für die Kamera (Reflexionen, schlechte Belichtung,...), an Fehlstellen im Parcours oder an Kreuzungen kein Profil gefunden, so wird die erste Komponente p_{y1} in 5 cm Schritten inkrementiert bis das Ende des erkennbaren Bereichs von 130 cm erreicht ist, dann wird der Vorgang für diesen Durchlauf abgebrochen. Abbildung 4.8 zeigt die um 75cm nach vorne geschobene Linie zwischen Q_1 und Q_2 (untere schwarze Linie). Der Bereich zwischen den schwarzen Linie ist die Region of Interest.

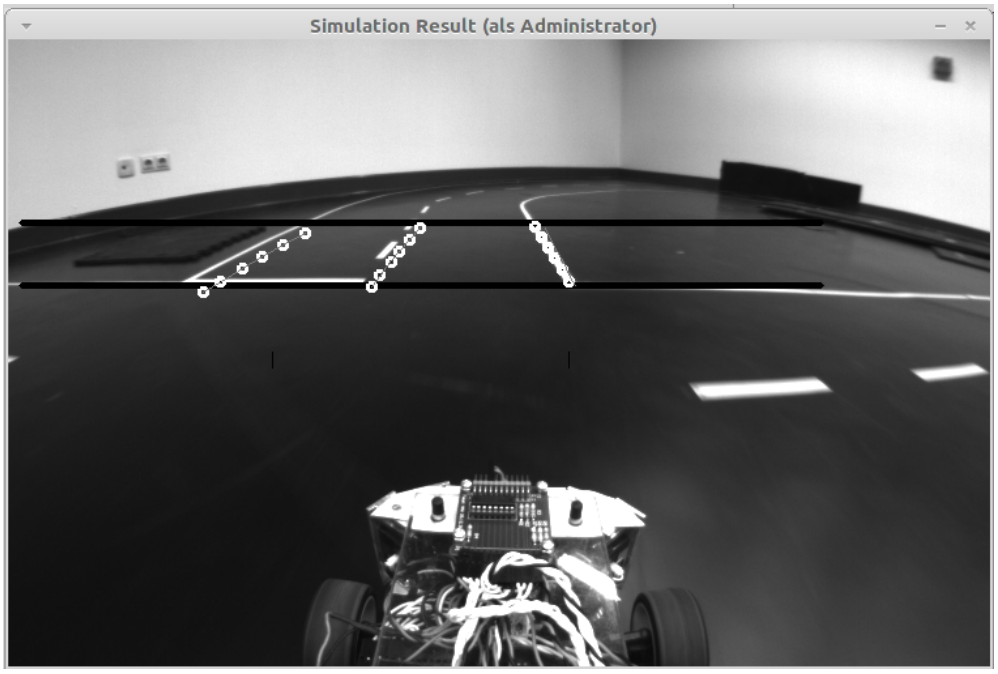

Abbildung 4.8.: Aufgrund mangelnder Profile verkleinerte Region of Interest

Schritt 2: Gefundene Profile validieren

Alle gefundenen Profile werden hinsichtlich ihrer Validität untersucht. Hierfür werden eine Reihe von Plausibilitätsregeln (beispielsweise Abstände zueinander, Lage in Bezug auf das Auto oder die Anzahl) verwendet, die nachfolgend erläutert werden. Der erste Schritt der Validierung besteht darin, die gefundenen Profile aus Schritt 1 auf ihre Größe hin zu untersuchen. Tests haben ergeben, dass eine Fahrbahnbegrenzung in K_B zwischen Q_1 und Q_2 niemals schmaler als $\phi = 4$ Pixel ist. Unter dieser Voraussetzung können fälschlich erkannte Profile, die häufig durch Reflexionen entstehen, gefiltert werden. Hierzu wird, wie im folgenden Pseudocode beschrieben, vorgegangen:

Data : Profilliste, ϕ
Result : Profilliste
foreach $Profil\ \ p\ :\ Profilliste$ **do**
  from = 0;
  to = 0;
  for $i = p.y_2, break = false \rightarrow break = true$ **do**
    if $abs(g(p.y_1, i)\ -\ g(p.y_1, i+1)) \geq threshold$ **then**
      break = true;
      to = i;
    end
    else i++;
  end
  for $i = p.y_2, break = false \rightarrow break = true$ **do**
    if $abs(g(p.y_1, i)\ -\ g(p.y_1, i-1)) \geq threshold$ **then**
      break = true;
      from = i;
    end
    else i−− ;
  end
  if $(to - from) < \phi$ **then**
    Profilliste.remove(p);
  end
end

Pseudocode 2 : Heuristik 1 - Schmale Profile entfernen

Die zweite angewandte Heuristik ordnet die übrigen Profile den drei Fahrbahnbegrenzungen zu. Hierzu wird eine Toleranz $\theta = 7$ definiert, die durch Subtraktion von den idealisierten Positionen P_{H2} der Fahrbahnbegrenzungen in K_W den Punkt $P1_{H2}$ und durch Addition den Punkt $P2_{H2}$ ergibt. Die Profile befinden sich in K_W bei $y_2 = \{-60, -20, +20\}$. Begründet ist die Wahl von y_2 durch die Breite der Fahrspuren von jeweils 40 cm und der Position des Fahrzeugs in der Mitte der rechten Spur. Abbildung 4.9 zeigt die zuvor genannte idealisierte Lage der Fahrbahnbegrenzungen und der Toleranz θ.

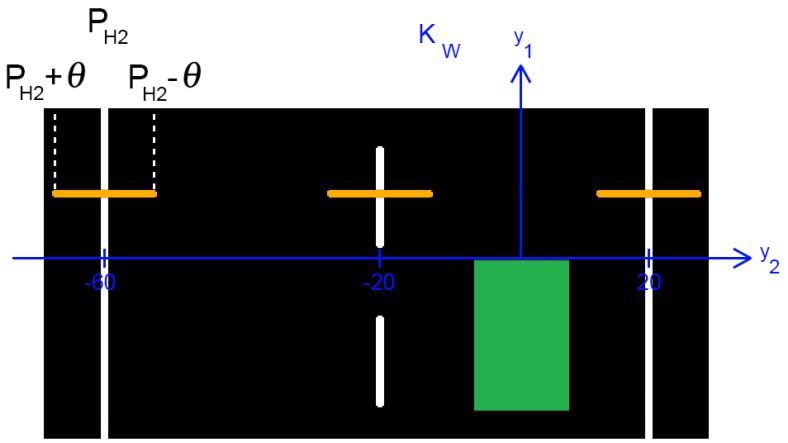

Abbildung 4.9.: Heuristik 2 - Sondieren gefundener Profile und Fahrbahnbegrenzungszuordnung

Pseudocode 3 wird für jede der drei Fahrspurbegrenzungen ausgeführt. Profile, die nicht zugeordnet werden können, werden verworfen und nicht weiter betrachtet. Sollte einer Fahrspur mehr als ein Profil zugeordnet werden, wird θ solange verkleinert, bis nur noch ein Profil vorhanden ist. Sollten zwei Profile den gleichen Abstand zur idealisierten Position in K_W haben, wird das erste gewählt. Sollte für eine Fahrspurbegrenzung kein Profil zurückgegeben werden, wird davon ausgegangen, dass sich in K_W zwischen $P1_{H2}$ und $P2_{H2}$ dieser Fahrspurbegrenzung kein Profil befindet. Übergeben wird die Profilliste, das initiale θ und die Position der idealisierten Fahrspur P_{H2}.

```
Data : Profilliste,θ, P_H2
Result : Fahrspurprofilliste
done = false;
Backupprofil = null;
while !done do
    foreach Profil p : Profilliste do
        if p.y_2 > P_H2 − θ && p.y_2 < P_H2 + θ then
            Fahrspurprofilliste.pushback(p);
        end
    end
    if Fahrspurprofilliste.size() > 1 then
        θ − −;
        BackupProfil = Fahrspurprofilliste.getFirstElement();
        Fahrspurprofilliste.clear();
    end
    else if Fahrspurprofilliste.size() == 0 && BackupProfil != null then
        Fahrspurprofilliste.pushback(BackupProfil);
    end
    else
        done = true;
    end
end
```

Pseudocode 3 : Heuristik 2 - Fahrbahnbegrenzungszuordnung

In Tests hat sich gezeigt, dass dieses Verfahren sehr gut funktioniert, sofern die Position des Fahrzeugs auf der Strecke, die Kalibrierung der projektiven Transformation T sowie die Verzeichnungskorrektur V ausreichend gut ist. Je größer die Zuordnungstoleranz θ, desto wahrscheinlicher können Stellungsfehler des Fahrzeugs und Fehler in den Transformationen ausgeglichen werden, allerdings erhöht sich dadurch auch die Chance, dass fälschlich erkannte Profile statt der eigentlichen Fahrbahnbegrenzungen erkannt werden.

Schritt 3: Tracing der Fahrbahnbegrenzungen

Der dritte Schritt im Programmablauf besteht darin, den Verlauf der zuvor als valide markierten und den jeweiligen Fahrbahnbegrenzungen zugeordneten Profilen zu ermitteln (Tracing). Dazu wurde ein Verfahren gewählt, das dem des Menschen beim Steuern eines Fahrzeugs sehr ähnlich ist. Wie sich aus den Testvideos, die in der Veranstaltung *Projekt 1* (vgl.

[16]) mit verschiedenen Versuchspersonen erstellt wurden, ergeben hat, wird dem Verlauf der Fahrbahn intuitiv gefolgt. Hierbei versucht der menschliche Fahrer eines Vehikels stets einen Punkt in einer der Geschwindigkeit angepassten Entfernung vor dem Fahrzeug zu betrachten, um frühzeitig eine Korrektur der Trajektorie des Fahrzeugs in Richtung dieses Punktes vornehmen zu können. Dieses Verhalten wurde in Software umgesetzt und hat sich als sehr stabil erwiesen und bildet den Kern dieses Tracing-Algorithmus.

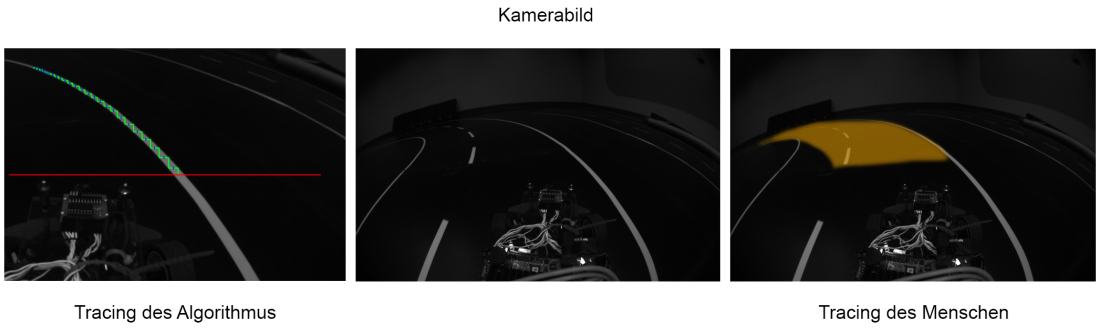

Abbildung 4.10.: Vergleich des menschlichen und algorithmischen Tracings

In Abbildung 4.10 wird veranschaulicht, wie der Algorithmus konkret einer Fahrbahnbegrenzung folgt und im Vergleich dazu das menschliche Verhalten beim Betrachten des Straßenverlaufs. Der Algorithmus benötigt eine valide Profillinie aus Schritt 2. Es wird bereits in Pseudocode 1 darauf geachtet, dass der Mittelpunkt des Profils zurückgegeben wird, wie in der Abbildung dargestellt ist. Pseudocode 4 beschreibt das in Abbildung 4.10 dargestellte treppenstufenartige Absuchen der Fahrbahnbegrenzung.

```
Data : Profilliste, threshold, scanRangeFarestPixel
Result : L_B
foreach Profil p : Profilliste do
    xTemp = 0;
    y_2 = p.y_2;
    for i = p.y_1, → i > scanRangeFarestPixel do
        if abs(g(i, y_2) − g(i − 3, y_2)) > threshold then
            if abs(g(i, y_2) − g(i, y_2 + 3)) > threshold then
                xTemp = y_2;
                for j = y_2, → j ≥ 0 do
                    if abs(g(i, j) − g(i, j − 3)) ≥ threshold then
                        Q.y_1 = i;
                        Q.y_2 = (j + xTemp)/2;
                        L_B.pushback(Q);
                        break;
                    end
                    j − −;
                end
            end
            else if abs(g(i, y_2) − g(i, y_2 − 3)) > threshold then
                xTemp = y_2;
                for j = y_2, → j ≤ (N − 1) do
                    if abs(g(i, j) − g(i, j + 3)) > threshold then
                        Q.y_1 = i;
                        Q.y_2 = (j + xTemp)/2;
                        L_B.pushback(Q) ;
                        break;
                    end
                    j + +;
                end
            end
        end
        i − −;
    end
end
```

Pseudocode 4 : Kernalgorithmus - Tracing einer Fahrspurbegrenzung

Das Ergebnis ist eine Liste L_B mit m Punkten Q aus dem Koordinatensystem K_B.

$$L_B = [Q_m] \quad mit\ m \in \{\mathbb{N}^+\} \tag{4.17}$$

Sollte die Linie enden, wie beispielsweise an einer Fehlstelle im Parcours oder einer Mittellinie, so terminiert dieses Verfahren, da weder oben, noch links oder rechts weiße Pixel zu finden sind. Des Weiteren terminiert das Suchverfahren, wenn der Grauwert $g(x_1, x_2)$ des aktuell untersuchten Pixels aus B $<threshold * 2$ des ersten Pixels ist. Auf diese Weise kann sichergestellt werden, dass wirklich nur helle weiße Teile des Bildes untersucht werden. Durch die Begrenzung der Region of Interest auf die Straße, der Bereich zwischen den beiden roten Linien in Abbildung 4.11, werden weitere Fehlerquellen bereits im Vorhinein ausgeschlossen.

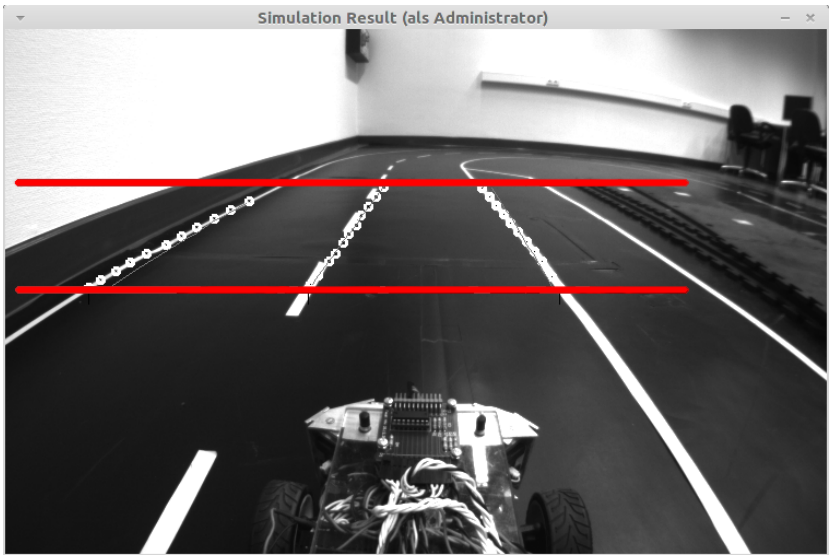

Abbildung 4.11.: Region of Interest

Gewählt werden diese beiden Begrenzungen nach Bedarf. Die untere liegt direkt vor dem Auto, also bei etwa 35 cm und die obere bei maximal 130 cm, dem Ende der erkennbaren Fahrbahn. Häufig ist es so, dass die Stufen, die der implementierte Algorithmus bildet, in Kurven mit zunehmender Entfernung immer kleiner werden. Ist entweder die Breite oder die Höhe einer Stufe kleiner als 3 Pixel, wird ebenfalls terminiert. Ein häufiges Szenario stellen Reflexionen auf dem Straßenbelag dar. Je nach Zustand, in dem sich der Algorithmus befindet, also ob gerade vertikal oder horizontal gesucht wird, verschiebt sich auch die Lage des nächsten in L_B abgespeicherten Punktes. Um starke Ausreißer zu filtern, wird das Verhältnis der Breite und Höhe jeder Treppenstufe untersucht. Sollten sich Höhe und Breite stärker als

den Faktor 15 unterscheiden, wird abgebrochen. Dieser Wert muss so groß sein, damit Geraden keine Probleme darstellen.
Start- und Ziellinien sowie Kreuzungen sind besondere Situationen, da bei diesen nicht terminiert werden darf und diese von anderen Störungen unterschieden werden müssen. Auch wenn diese nur lokal gehäuft auftreten stellen sie trotz alledem mit zunehmender Nähe ein Problem dar. In den Abbildungen 4.12 und 4.13 wird die Problematik und der dazugehörige Lösungsansatz verdeutlicht:

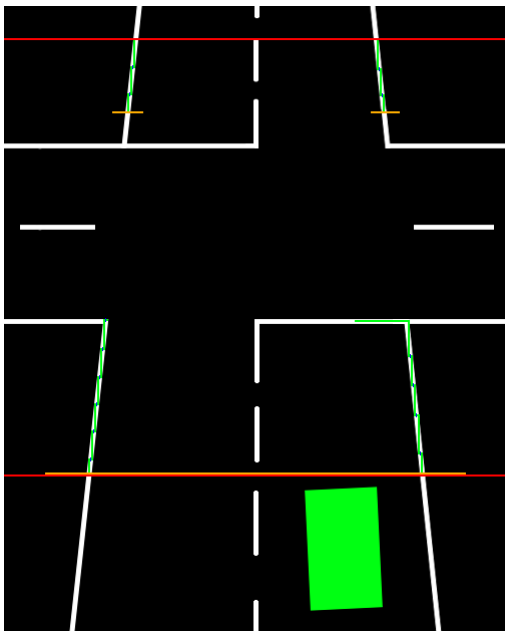

Abbildung 4.12.: Schritt 3 - Tracing an einer Kreuzung

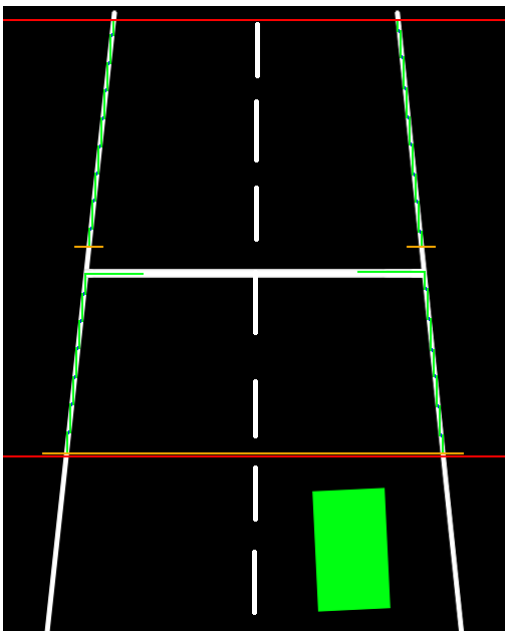

Abbildung 4.13.: Schritt 3 - Tracing an einer Start-/Ziellinie

Die gewählte Perspektive ist die der Kamera des Fahrzeug, die Position ist allerdings oberhalb des Fahrzeugs. Die roten horizontalen Linien bilden die Region of Interest. Die gelben Linien wurden zuvor bereits in Schritt 1 (siehe Abbildungen 4.6 und 4.7) detailliert beschrieben. Auf den Abbildungen sind im oberen Bildbereich zwei kleine gelbe Suchlinien dargestellt. Die Position dieser wird durch eine Verschiebung in K_W des letzten gespeicherten Punktes in vertikaler Richtung ermittelt:

Data : L_B
Result : L_B
$P.y1 = V(T(L_B.getLast().x2))$;
$P.y2 = V(T(L_B.getLast().x1))$;
$\gamma = 10$ **if** *isKreuzung* **then**
  $P.y1+ = 2 * Spurbreite + \gamma$;
  $L_B.pushback(V^{-1}(T^{-1}(P.y1, P.y2)))$;
end
else if *isStartZiellinie* **then**
  $P.y1+ = \gamma$;
  $L_B.pushback(V^{-1}(T^{-1}(P.y1, P.y2)))$;
end
else
  $terminate()$;
end

Pseudocode 5 : Verschiebung des letzten Punktes aus L_B vor die Start-/ Ziellinie beziehungsweise Kreuzung

Durch $\gamma = 10cm$ wird sichergestellt, dass die Suchlinie vor und nicht auf der jeweiligen querenden Linie platziert wird. Wird im Bereich der Suchlinie ein markantes Profil gefunden (vergleiche Schritt 1), wird die dazugehörige Linie entsprechend Schritt 3 verfolgen. Die gewählte Verschiebung ist davon abhängig, ob es sich um die Haltelinie einer Kreuzung oder um eine Start-/Ziellinie handelt. In grün ist die Linienverfolgung aus Schritt 3 (siehe 4) eingezeichnet worden. Die blauen vertikalen Linien markieren die Punkte Q, die beim Tracing gefundenen wurden und zur Weiterverarbeitung in L_B abgespeichert werden.

Polaris benötigt für die Polynomapproximation Koordinaten aus K_W. Diese werden für jede Fahrbahn in einer Liste L_W mit n Elementen gespeichert. Jedes Element ist ein Punkt P, der durch verkettete Anwendung von V und T auf einen Punkt Q aus K_B entstanden ist und zwei Koordinaten y_1 und y_2 aus K_W enthält. Q entstammt dabei einer Liste L_B aus dem vorhergehenden Schritt:

$$P.y1 = V(T(Q.x2)) \tag{4.18}$$

$$P.y2 = V(T(Q.x1)) \tag{4.19}$$

$$L_W = [P_n] \quad mit\ n \in \{\mathbb{N}^+\} \tag{4.20}$$

$$L_W = V(T(L_B)) \tag{4.21}$$

Nachfolgend sind nur noch die Listen L_W aus K_W relevant.

Schritt 4: Listen vervollständigen
Es gibt Fälle, in denen Profile nicht weiter verfolgt werden können, beispielsweise am Ende eines Mittelstrichs, an Fehlstellen oder Kreuzungen. In so einem Fall wird die euklidische Distanz d zwischen dem ersten Punkt P_1 und dem letzten Punkt P_n, der in Schritt 3 für eine Fahrspurbegrenzung gefunden wurde, in K_W berechnet:

$$d(P_1, P_n) = \sqrt{(P_1.x - P_n.x)^2 + (P_1.y - P_n.y)^2} \tag{4.22}$$

Ist $d \leq 20cm$ werden die Punkte P_2 bis P_n aus der Liste L_W entfernt und durch Approximation vervollständigt. Konnte einer Fahrspurbegrenzung kein Profil zugewiesen werden und folglich auch keine Liste von Punkten in Schritt 3 für diese erzeugt werden, wird versucht, diese auf Basis vorhandener Stützpunktlisten zu approximieren. Beispielhaft wird in Pseudocode **??** die Approximation der Mittellinie basierend auf erfolgreich gefundenen äußeren Fahrspurbegrenzungen und in 7 eine Approximation der mittleren und linken Fahrspurbegrenzung gezeigt:

Data : $L_{W_{links}}, L_{W_{rechts}}$
Result : $L_{W_{mitte}}$
if $isValid(L_{W_{links}})\&\&isValid(L_{W_{rechts}})$ **then**
  for $i = 0 \ \rightarrow \ ((i < L_{W_{links}}.size()) \ \&\& \ (i < L_{W_{rechts}}.size()))$ **do**
    $y_1 = L_{W_{rechts}}.get(i).y1;$
    $y_2 = abs(L_{W_{rechts}}.get(i).y2) - abs(L_{W_{links}}.get(i).y2);$
    $L_{W_{mitte}}.pushback(Punkt(y_1, y_2));$
  end
end
Pseudocode 6 : Approximation der Stützpunktliste der mittleren Fahrspurbegrenzung

Data : $L_{W_{rechts}}$
Result : $L_{W_{links}}, L_{W_{mitte}}$
Spurbreite = 40;
if $isValid(L_{W_{rechts}})$ **then**
  for $i = 0 \rightarrow i < L_{W_{rechts}}.size()$ **do**
    $y_1 = L_{W_{rechts}}.get(i).y1$;
    $y_2 = L_{W_{rechts}}.get(i).y2 -$ Spurbreite;
    $L_{W_{mitte}}.pushback(Punkt(y_1, y_2))$;
    $y_2 = L_{W_{rechts}}.get(i).y2 - 2*$ Spurbreite;
    $L_{W_{links}}.pushback(Punkt(y_1, y_2))$;
  end
end

Pseudocode 7 : Approximation der Stützpunktlisten der linken und mittleren Fahrspurbegrenzungen

Abbildung 4.14 zeigt das Ergebnis der zuvor beschriebenen Approximation der mittleren Fahrspurbegrenzung (6) und Abbildung 4.15 das Ergebnis von Pseudocode 7.

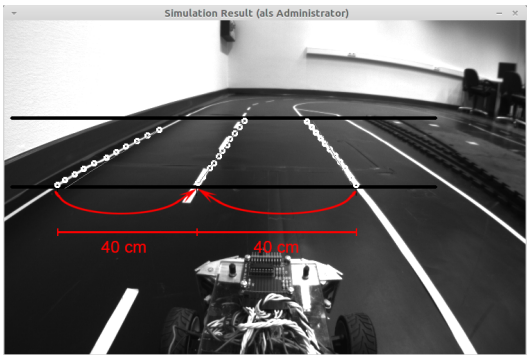

Abbildung 4.14.: Approximation der mittleren Fahrspurbegrenzung

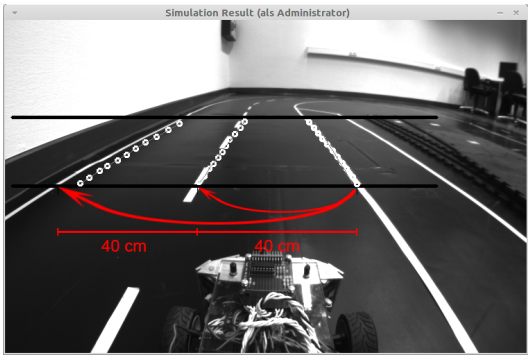

Abbildung 4.15.: Approximation der linken und mittleren Fahrspurbegrenzung

Abbildung 4.16 zeigt die Approximation der linken und mittleren Spurbegrenzung am Anfang einer Linkskurve.

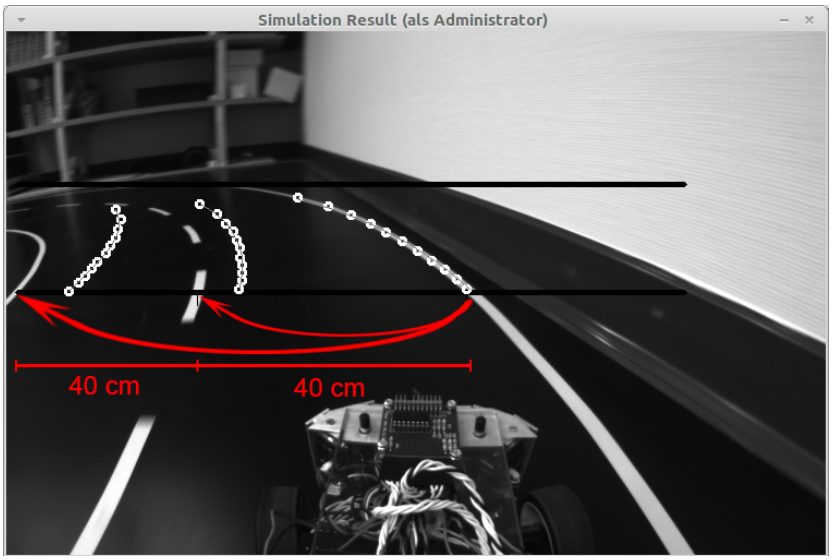

Abbildung 4.16.: Approximation der linken und mittleren Fahrspurbegrenzung

Besonders in den Abbildungen 4.15 und 4.16 ist der Fehler der Transformation T^{-1} und der Verzeichnung V^{-1} deutlich zu erkennen. Die roten Pfeilen zeigen von der Approximationsgrundlage auf die angestrebte Position. So schlecht das Ergebnis auch erscheint, liegt der Fehler beim Einzeichnen in das Bild bei unter 15 cm und das auch lediglich auf der äußeren linken Fahrspurbegrenzung die beim Befahren der rechten Spur nicht zur Steuerung verwendet wird. Aus Sicht des Fahrzeugs ist der Fehler deutlich kleiner, da alle Algorithmen in K_W berechnet werden und die starken Abweichungen beim Einzeichnen, genauer durch die Anwendung von V^{-1} entstehen, was auf ein leicht verstelltes Objektiv zum Zeitpunkt der Aufnahme zurückgeführt werden kann. Ist $20 \leq d(L_{W_1}) \leq 50$ und $d(L_{W_2}) \geq 75$, wird L_{W_1} auf Grundlage von L_{W_2} und des Abstands ihrer ersten Punkte, also der y_2 Komponente, verlängert. Nachfolgend am Beispiel einer kurzen linken und längeren rechten Fahrspurbegrenzung:

Data : $L_{W_{links}}, L_{W_{rechts}}$
Result : $L^*_{W_{links}}$
Spurbreite = 40;
if $d(L_{W_{links}}) \geq 20 \;\; \&\& \;\; d(L_{W_{links}}) \leq 50 \;\; \&\& \;\; d(L_{W_{rechts}}) \geq 75$ **then**
    $L^*_{W_{links}}.copyOf(L_{W_{links}})$;
    $dist = abs(abs(L_{W_{rechts}}.get(0).y_2) - abs(L_{W_{links}}.get(0).y_2))$;
    for $i = 0 \;\; \rightarrow \;\; i < L_{W_{rechts}}.size()$ **do**
        if $L_{W_{rechts}}.get(i).y1 > 50$ **then**
            $y_1 = L_{W_{rechts}}.get(i).y1$;
            $y_2 = L_{W_{rechts}}.get(i).y2 - dist$;
            $L^*_{W_{links}}.pushback(Punkt(y_1, y_2))$;
            $i++$;
        end
    end
end

Pseudocode 8 : Vervollständigung der Stützpunktliste der linken Fahrspurbegrenzung

Die hier verwendeten Operationen sind ressourcensparend, da keine Transformationen T^{-1} oder Verzeichnungen V^{-1} durchgeführt wird. Es werden lediglich Additionen und Subtraktionen zur Vervollständigung der Stützpunktlisten aus Schritt 3 genutzt.

Schritt 5: Fallback für den Fall, dass keine Fahrbahnbegrenzung gefunden wurden
Sollte der Fall eintreten, dass keine Fahrbahnbegrenzung gefunden werden konnte, wie es Abbildung 4.17 zeigt, werden die Punkte des letzten Durchgangs verwendet, bei dem mindestens eine valide Stützpunktliste vorhanden ist. Zwischen Kurven und Geraden wird nicht unterschieden. Dieses Verfahren hat sich als sehr praktikabel erwiesen und ist definitiv besser, als den Lenkwinkel auf $0°$ einzustellen, wie es in dem Fall geschieht, wenn keine Polynome gebildet werden können, weil keine Stützpunkte für die Polynombildung in den Punktlisten vorhanden sind.

Abbildung 4.17.: Fallback an einer Kreuzung

5. Zusammenfassung

In dieser Masterarbeit wurde eine Simulationssoftware zur Erforschung menschlichen Verhaltens beim Steuern eines Fahrzeugs entwickelt. Während der Tests wurden die Versuchspersonen mit verschiedenen Szenarien konfrontiert, die auf Basis einer Passantenbefragung ausgearbeitet worden sind. Durch die Verwendung eines Eyetrackers konnten die Fixationspunkte und Sakkaden während des Tests aufgezeichnet werden. Die Auswertung der dabei entstandenen Daten wurde mit Hinblick auf die zuvor festgelegten Szenarien durchgeführt. Auf Basis des extrahierten Wissens konnten für einen Spezialfall zwei Regelalgorithmen in Matlab Simulink entwickelt, parametriert und getestet werden. Die Implementierung und der Anschluss dieser an das speziell entwickelte Interface der Simulationssoftware ermöglicht die Evaluation der beiden Regelalgorithmen.
Des Weiteren wurde aufbauend auf den Testergebnissen ein Algorithmus zur Merkmalsextraktion von relevanten Informationen für die Spurführung entwickelt, die zur Steuerung der Carolo-Cup Plattform benötigt werden.

5.1. Bewertung

Der Ansatz, biologisch motivierte Algorithmen zu implementieren, wird bereits seit vielen Jahren sehr erfolgreich genutzt. Zwar halten sich die meisten Technologiehersteller mit der Entwickelung von biologisch motivierten Algorithmen zurück, trotzdem werden Algorithmen, die dieser Klassifizierung entsprechen, früher oder später Einzug in den Alltag halten. Seien es Roboter, die Mimik und Gestik [39][31][10] verstehen, Systeme mit der Fähigkeit die menschliche Sprache zu erlernen [26][34] oder autonome Systeme die Teile ihrer Logik in menschliche Adaption und Abstraktion verlagern.
Autonome Systeme sind aus der heutigen Welt nicht mehr wegzudenken. Sie übernehmen in der Regel begrenzte, klar definierte und sich, wenn auch im weitesten Sinne, wiederholende Aufgaben und handeln in Bezug auf diese selbstständig und unabhängig, also autonom. In diesem Zusammenhang wird der Begriff des Systems als Verbund aus in Wechselwirkung stehenden Elementen, die als Aufgaben-gebundene abgegrenzte Einheiten angesehen werden können, definiert.

Wird von diesen klar definierten und begrenzten Aufgaben abgewichen, steigt die Komplexität der zu lösenden Aufgabe für die Maschine stark an und kann von konventionellen starren Algorithmen nur schwer gelöst werden. Diese Problematik kann in einigen Fällen durch Anwendung verschiedener Verfahren der künstlichen Intelligenz in Kombination mit entsprechenden Trainingsdaten umgangen werden. Durch Abstraktion dieser Trainingsdaten beziehungsweise Hintergrundinformationen wird die Grundlage für die Lösung von Aufgaben einer bestimmten Klasse geschaffen, was den direkten Schluss zulässt, dass die gewählten Hintergrundinformationen essentiellen Einfluss auf das problemlösende Verhalten von autonomen Systemen haben.
Auch wenn hier eine Empfehlung für biologisch motivierte Algorithmen ausgesprochen wird, ist die Reduktion der Abtastrate außerhalb von Laborbedingungen in der Regel nicht sinnvoll, da den Systemen so die Fähigkeit genommen wird, schnell und präzise zu reagieren.

Das Ziel sollte eine Kombination aus der menschlichen Flexibilität und der technischen Reaktionsfähigkeit und Präzision sein.

5.2. Ausblick

Der nächste Schritt bestünde darin, situationsbedingte Regel- und Handlungsvorschriften zu entwickeln. So ist es denkbar, dass Algorithmen wie Pure Pursuit entsprechend dem Fixationspunkt der Versuchspersonen das Fahrzeug regeln, um beispielsweise geplante Ausweichvorgänge oder Spurwechsel durchzuführen. Doch um dieses tun zu können, muss zunächst ein Algorithmus entwickelt werden, der die Fahrbahn, entsprechend dem menschlichen Vorbild, nach Hindernissen absucht. Die vorhandenen Regelalgorithmen können den neuen Situationen leicht angepasst werden.

Der nächste Schritt besteht darin, ein Plugin für den FAUSTcore zu entwickeln, welches dieses Verfahren abbilden kann. Hier sollte darauf geachtet werden, dass zunächst eine rudimentäre Funktionalität geschaffen wird, das Fahrzeug auf der Straße zu halten.

Abbildungsverzeichnis

1.1. Unfälle mit Personenschaden, verursacht durch Fahrerfehler nach Unfallursachen [33] 1

2.1. Übersicht der FAUST Fahrzeugplattformen 7
2.2. Diamond - 2. Generation der Carolo-Cup Plattform 8
2.3. Abbildungsschärfe auf der Netzhaut des linken Auges [15] 10
2.4. Visualisierung der unterschiedlichen Bereiche des Gesichtsfeldes; von innen nach außen: foveal, para-foveal, peripher [18] 10

3.1. Screenshot der Simulationssoftware aus [16] 14
3.2. VCM-Architektur der Simulationssoftware 15
3.3. Klassendiagramm der Simulationssoftware 16
3.4. Vorschau der Simulationssoftware 17
3.5. Straßengeometrie 18
3.6. Kurvenbildung bei verwendeter Straßengeometrie [17] 20
3.7. Projektionsgeometrie zur Ermittlung der vertikalen Position eines Straßensegments 21
3.8. Übersicht der Einzelschritte zur Erzeugung der Simulation 23
3.9. Logitech Momo Racing 24
3.10. Objekte zur Platzierung auf der Fahrbahn 28
3.11. Positionsangaben zur Platzierungen von Objekten auf der Fahrbahn 29
3.12. Übersicht der Testumgebung 35
3.13. Akkumulierende Heatmap von Versuchsperson 1 39
3.14. Clusteranalyse von Versuchsperson 1 39
3.15. Akkumulierende Heatmap von Versuchsperson 2 39
3.16. Clusteranalyse von Versuchsperson 2 39
3.17. Akkumulierende Heatmap von Versuchsperson 3 39
3.18. Clusteranalyse von Versuchsperson 3 39
3.19. Akkumulierende Heatmap von Versuchsperson 4 40

3.20. Clusteranalyse von Versuchsperson 4 . . . 40
3.21. Akkumulierende Heatmap von Versuchsperson 5 . . . 40
3.22. Clusteranalyse von Versuchsperson 5 . . . 40
3.23. Akkumulierende Heatmap von Versuchsperson 6 . . . 40
3.24. Clusteranalyse von Versuchsperson 6 . . . 40
3.25. Heatmap-basierte Region of Interest . . . 41
3.26. Clusteranalyse-basierte Region of Interest . . . 41
3.27. Diagramm, mit horizontal aufgetragener zurückgelegter Distanz und vertikal aufgetragener Position auf der Fahrbahn (oben) beziehungsweise des prozentualen Lenkwinkels (unten) . . . 43
3.28. Verlauf der horizontalen Position des Fahrzeugs auf der Fahrbahn beim Vorbeifahren an Hindernissen (rote Punkte) . . . 46
3.29. Der Fixationspunkt verschiebt sich entsprechend der Trajektorie . . . 47
3.30. Ausweichen mit konstanter Veränderung der horizontalen Position (bekannt als Follow the Carrot [25]) . . . 48
3.31. Degressives Ausweichen mit stetig schwächer werdendem Lenkwinkel . . . 48
3.32. Ausweichen mit Pure Pursuit nach Coulter [11] . . . 49
3.33. **A**re **o**f **I**nterest . . . 50
3.34. Durchschnittliche Fixationszeit . . . 51
3.35. Maximale Fixationszeit . . . 51
3.36. Gesamte Fixationszeit . . . 51
3.37. Gesamtanzahl Fixationen . . . 51

4.1. Simulink-Schaltbild einer PT2- und einer PID-Regelstrecke . . . 54
4.2. Ergebnis der parametrierten PT2- und PID-Regler im Simulink-Scopeplot . . . 55
4.3. In rot die vom PT2-Regler gefahrene Trajektorie über der der Versuchspersonen 56
4.4. In rot die vom PID-Regler gefahrene Trajektorie über der der Versuchspersonen 56
4.5. Nomenklatur und Transformationsübersicht für Bild- und Weltkoordinatensystem 59
4.6. Schritt 1 - Initialzustand . . . 60
4.7. Schritt 1 - Positionierung der Profilsuchlinien nach vorherigem erfolgreichen Durchlauf . . . 60
4.8. Aufgrund mangelnder Profile verkleinerte Region of Interest . . . 62
4.9. Heuristik 2 - Sondieren gefundener Profile und Fahrbahnbegrenzungszuordnung 64
4.10. Vergleich des menschlichen und algorithmischen Tracings . . . 66
4.11. Region of Interest . . . 68
4.12. Schritt 3 - Tracing an einer Kreuzung . . . 69

4.13. Schritt 3 - Tracing an einer Start-/Ziellinie 69
4.14. Approximation der mittleren Fahrspurbegrenzung 72
4.15. Approximation der linken und mittleren Fahrspurbegrenzung 72
4.16. Approximation der linken und mittleren Fahrspurbegrenzung 73
4.17. Fallback an einer Kreuzung 75

Listings

3.1. Beispielkonfiguration eines Tiles ohne Events 27
3.2. Auflistung und Konfiguration verfügbarer Objekte 29
3.3. Konfiguration eines bewegten Objekts 30
3.4. Konfiguration einer Geschwindigkeitsänderung 30
3.5. Konfiguration einer Positionsänderung des Fahrzeugs 30
3.6. Konfiguration einer Geschwindigkeitsbegrenzung 30
3.7. Konfiguration der Geschwindigsfixierung 31
3.8. Konfiguration der Lenkrad-Nullstellungsänderung 31
3.9. Konfiguration der Pausierung von Tests 31
3.10. Konfiguration der Texteinblendung 31
3.11. Konfiguration möglicher Screenüberblendungen 32
3.12. Vollständiges Beispiel einer 1-Tile Konfiguration 32

A.1. Konfigurationsdatei zur Parametrierung der Tests 82

A. Konfigurationsdatei der Simulation

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <fps>30</fps>
  <speed>0</speed>
  <track>
  <tile>
    <name>Test 1.1</name>
    <speed>60</speed>
    <playerposx>null</playerposx>
    <end>180000</end>
    <events>
      <!-- Geschwindigkeit -->
      <object gameObjectNumber="30" distance="1000" value="60" />
      <!-- konstante Geschwindigkeit -->
      <object gameObjectNumber="33" distance="1000" value="60" />
      <!-- Pause und Nachricht an die Versuchsperson -->
      <object gameObjectNumber="50" distance="10000" value="5000" />
      <object gameObjectNumber="70" distance="10000"
        value="Test 1.1 beginnt." />
      <!-- Baum -->
      <object gameObjectNumber="1" distance="15000" value="0.0" />
      <!-- FAUST Schild -->
      <object gameObjectNumber="3" distance="35000" value="0.6" />
      <!-- HAW Schild -->
      <object gameObjectNumber="4" distance="55000" value="-0.6" />
      <!-- HAW Schild -->
      <object gameObjectNumber="4" distance="65000" value="-0.1" />
      <!-- Baum -->
      <object gameObjectNumber="1" distance="75000" value="0.3" />
      <!-- Felsen -->
      <object gameObjectNumber="2" distance="100000" value="-0.8" />
      <!-- HAW Schild -->
      <object gameObjectNumber="4" distance="105000" value="0.3" />
      <!-- Wald -->
      <object gameObjectNumber="1" distance="126000" value="0.4" />
```

```
36        <object gameObjectNumber="1" distance="127000" value="0.2" />
37        <object gameObjectNumber="1" distance="128000" value="0.0" />
38        <object gameObjectNumber="1" distance="129000" value="-0.2" />
39        <object gameObjectNumber="1" distance="130000" value="-0.4" />
40        <!-- FAUST Schild -->
41        <object gameObjectNumber="3" distance="140000" value="0.6" />
42        <!-- HAW Schild -->
43        <object gameObjectNumber="4" distance="145000" value="-0.6" />
44        <!-- Baum -->
45        <object gameObjectNumber="1" distance="155000" value="0.3" />
46        <!-- 1.1 zuende bei 180.000 -->
47      </events>
48    </tile>
49    <tile>
50      <name>Test 1.2</name>
51      <speed>60</speed>
52      <playerposx>null</playerposx>
53      <end>360000</end>
54      <events>
55        <!-- Geschwindigkeit -->
56        <object gameObjectNumber="30" distance="180001" value="60" />
57        <!-- konstante Geschwindigkeit -->
58        <object gameObjectNumber="33" distance="180001" value="60" />
59        <!-- Pause und Nachricht an die Versuchsperson -->
60        <object gameObjectNumber="50" distance="185000" value="5000" />
61        <object gameObjectNumber="70" distance="185000"
62          value="Test 1.2 beginnt." />
63        <!-- Baum -->
64        <object gameObjectNumber="1" distance="200000" value="0.0" />
65        <!-- FAUST Schild -->
66        <object gameObjectNumber="3" distance="215000" value="0.6" />
67        <!-- HAW Schild -->
68        <object gameObjectNumber="4" distance="230000" value="-0.6" />
69        <!-- HAW Schild -->
70        <object gameObjectNumber="4" distance="245000" value="-0.1" />
71        <!-- Baum -->
72        <object gameObjectNumber="1" distance="255000" value="0.3" />
73        <!-- Felsen -->
74        <object gameObjectNumber="2" distance="290000" value="-0.8" />
75        <!-- HAW Schild -->
76        <object gameObjectNumber="4" distance="291000" value="0.3" />
77        <!-- Wald -->
78        <object gameObjectNumber="1" distance="316000" value="0.4" />
```

```
79        <object gameObjectNumber="1" distance="317000" value="0.2" />
80        <object gameObjectNumber="1" distance="318000" value="0.0" />
81        <object gameObjectNumber="1" distance="319000" value="-0.2" />
82        <object gameObjectNumber="1" distance="320000" value="-0.4" />
83        <!-- FAUST Schild -->
84        <object gameObjectNumber="3" distance="335000" value="0.6" />
85        <!-- HAW Schild -->
86        <object gameObjectNumber="4" distance="340000" value="-0.6" />
87        <!-- Baum -->
88        <object gameObjectNumber="1" distance="350000" value="0.3" />
89        <!-- Felsen -->
90        <object gameObjectNumber="2" distance="355000" value="-0.6" />
91        <object gameObjectNumber="2" distance="356000" value="-0.4" />
92        <object gameObjectNumber="2" distance="357000" value="-0.2" />
93        <object gameObjectNumber="2" distance="358000" value="0" />
94        <object gameObjectNumber="2" distance="359000" value="0.20" />
95        <object gameObjectNumber="2" distance="360000" value="0.40" />
96        <!-- 1.2 zuende bei 360.000 -->
97      </events>
98    </tile>
99    <tile>
100     <name>Test 1.3</name>
101     <speed>60</speed>
102     <playerposx>null</playerposx>
103     <end>480000</end>
104     <events>
105       <!-- Pause und Nachricht an die Versuchsperson -->
106       <object gameObjectNumber="50" distance="361000" value="5000" />
107       <object gameObjectNumber="70" distance="361000"
108         value="Test 1.3 beginnt." />
109       <object gameObjectNumber="5" distance="400000" value="0.5" />
110       <object gameObjectNumber="5" distance="435000" value="-0.25" />
111       <object gameObjectNumber="5" distance="450000" value="0.6" />
112     </events>
113   </tile>
114   <tile>
115     <name>Test 1.4</name>
116     <speed>60</speed>
117     <playerposx>null</playerposx>
118     <end>620000</end>
119     <events>
120       <!-- Pause und Nachricht an die Versuchsperson -->
121       <object gameObjectNumber="50" distance="521000" value="5000" />
```

```
122       <object gameObjectNumber="70" distance="521000"
123         value="Test 1.4 beginnt." />
124       <object gameObjectNumber="5" distance="550000" value="0.5" />
125       <object gameObjectNumber="5" distance="585000" value="-0.25" />
126       <object gameObjectNumber="5" distance="600000" value="0.6" />
127     </events>
128   </tile>
129   <tile>
130     <name>Test 2</name>
131     <speed>60</speed>
132     <playerposx>null</playerposx>
133     <end>910000</end>
134     <events>
135       <!-- Pause und Nachricht an die Versuchsperson; Zentrierung-->
136       <object gameObjectNumber="50" distance="621000" value="5000" />
137       <object gameObjectNumber="70" distance="621000"
138         value="Test 2 beginnt. Fahrzeug zentral auf der mittleren
139                Spur halten." />
140       <object gameObjectNumber="31" distance="621000" value="0.0" />
141       <!-- Fahrzeug nach links verschieben -->
142       <object gameObjectNumber="31" distance="645000" value="-0.8" />
143       <!-- Fahrzeug nach rechts verschieben -->
144       <object gameObjectNumber="31" distance="680000" value="0.75" />
145       <!-- Steer Offset negativ -->
146       <object gameObjectNumber="35" distance="697000" value="-30" />
147       <!-- Steer Offset rückgängig -->
148       <object gameObjectNumber="35" distance="750000" value="0" />
149       <!-- Geschwindigkeit anpassen -->
150       <object gameObjectNumber="30" distance="780000" value="100" />
151       <object gameObjectNumber="32" distance="780000" value="120" />
152       <object gameObjectNumber="30" distance="780500" value="120" />
153       <!-- Steer Offset positiv -->
154       <object gameObjectNumber="35" distance="815000" value="45" />
155       <!-- Fahrzeug nach rechts verschieben -->
156       <object gameObjectNumber="31" distance="845000" value="-0.6" />
157       <!-- Steer Offset rückgängig -->
158       <object gameObjectNumber="35" distance="885000" value="0" />
159     </events>
160   </tile>
161   <tile>
162     <name>Test 3.1</name>
163     <speed>60</speed>
164     <playerposx>null</playerposx>
```

```
165     <end>1030000</end>
166     <events>
167       <!-- Pause und Nachricht an die Versuchsperson -->
168       <object gameObjectNumber="35" distance="890000" value="0" />
169       <object gameObjectNumber="50" distance="890000" value="5000" />
170       <object gameObjectNumber="70" distance="890000"
171         value="Test 3.1 beginnt." />
172       <!-- Schranke 3 -->
173       <object gameObjectNumber="10" distance="920000" value="-0.22" />
174       <object gameObjectNumber="30" distance="921000" value="60" />
175       <!-- Schranke 3 doppelt -->
176       <object gameObjectNumber="10" distance="960000" value="-0.575" />
177       <object gameObjectNumber="10" distance="960000" value="0.165" />
178       <object gameObjectNumber="32" distance="961000" value="100" />
179       <object gameObjectNumber="30" distance="961000" value="100" />
180       <!-- Schranke 3 -->
181       <object gameObjectNumber="9" distance="966000" value="-0.16" />
182       <!-- Schranke 4 -->
183       <object gameObjectNumber="11" distance="1005000" value="-0.145" />
184     </events>
185   </tile>
186   <tile>
187     <name>Test 3.2</name>
188     <speed>60</speed>
189     <playerposx>null</playerposx>
190     <end>1170000</end>
191     <events>
192       <!-- Pause und Nachricht an die Versuchsperson -->
193       <object gameObjectNumber="35" distance="1042000" value="0" />
194       <object gameObjectNumber="50" distance="1042000" value="5000" />
195       <object gameObjectNumber="70" distance="1042000"
196         value="Test 3.2 beginnt." />
197       <!-- Schranke 3 -->
198       <object gameObjectNumber="10" distance="1053000" value="-0.22" />
199       <object gameObjectNumber="30" distance="1053000" value="60" />
200       <!-- Schranke 3 doppelt -->
201       <object gameObjectNumber="10" distance="1088000" value="-0.575" />
202       <object gameObjectNumber="10" distance="1088000" value="0.165" />
203       <object gameObjectNumber="32" distance="1089000" value="100" />
204       <object gameObjectNumber="30" distance="1089000" value="100" />
205       <!-- Schranke 3 -->
206       <object gameObjectNumber="9" distance="1093000" value="-0.16" />
207       <!-- Schranke 4 -->
```

```
208       <object gameObjectNumber="11" distance="1120000" value="-0.145" />
209       <!-- Schranke 4 -->
210       <object gameObjectNumber="11" distance="1126000" value="-0.35" />
211     </events>
212   </tile>
213   <tile>
214     <name>Test 4</name>
215     <speed>0</speed>
216     <playerposx>null</playerposx>
217     <end>1490000</end>
218     <events>
219       <!-- Pause und Nachricht an die Versuchsperson -->
220       <object gameObjectNumber="34" distance="1160000" value="0" />
221       <object gameObjectNumber="50" distance="1160000" value="5000" />
222       <object gameObjectNumber="70" distance="1160000"
223         value="Test 4 beginnt. Pedale benutzen." />
224       <object gameObjectNumber="32" distance="1160000" value="150" />
225       <object gameObjectNumber="5" distance="1175000" value="0.5" />
226       <object gameObjectNumber="5" distance="1195000" value="-0.25" />
227       <object gameObjectNumber="5" distance="1215000" value="0.6" />
228       <object gameObjectNumber="5" distance="1295000" value="0.5" />
229       <object gameObjectNumber="5" distance="1355000" value="-0.25" />
230       <object gameObjectNumber="5" distance="1358000" value="0.6" />
231       <object gameObjectNumber="5" distance="1360000" value="0.5" />
232       <object gameObjectNumber="5" distance="1405000" value="-0.25" />
233       <object gameObjectNumber="5" distance="1435000" value="0.6" />
234     </events>
235   </tile>
236   </track>
237 </configuration>
```

XML-Konfiguration A.1: Konfigurationsdatei zur Parametrierung der Tests

Literaturverzeichnis

[1] Andrew B. Watson: *Handbook of Perception and Human Performance.* Wiley, 1986

[2] Bellet T. and Bailly B. and Mayenobe P. and and Georgeon O.: *Cognitive modelling and computational simulation of drivers mental activities. P.C. Cacciabue (Ed.), In : Modelling Driver Behaviour in Automotive Environment: Critical Issues in Driver Interactions with Intelligent Transport Systems,.* Springer Verlag, pp 315-343, 2007

[3] Brooks, R.A.: A robust layered control system for a mobile robot. In: *Robotics and Automation, IEEE Journal of* 2 (1986), Nr. 1, S. 14–23. – ISSN 0882-4967

[4] Brooks, R.A.: A robust layered control system for a mobile robot. In: *SPIE Proceedings* 0727 (1987), S. 77–84

[5] Carolo-Cup: *Carolo-Cup.* – URL http://www.carolo-cup.de/. – (Abgerufen am 08.12.2013)

[6] Carolo-Cup: *Carolo-Cup 2013 - Ergebnisse.* 2013. – URL https://wiki.ifr.ing.tu-bs.de/carolocup/system/files/Ergebnisse_2013.pdf. – (Abgerufen am 06.09.2013)

[7] Carolo-Cup: *Carolo-Cup Regelwerk 2013.* 2013. – URL https://wiki.ifr.ing.tu-bs.de/carolocup/system/files/Carolo-Cup_Regelwerk_2013.pdf

[8] Claude Elwood Shannon: Communication in the Presence of Noise. In: *Proc. IRE, Vol. 37, No. 1 (Jan. 1949), Nachdruck in: Proc. IEEE, Vol. 86, No. 2, (Feb. 1998)* (1949)

[9] Codeminders: *Java HID API.* – URL https://code.google.com/p/javahidapi/. – (Abgerufen am 04.09.2013)

[10] Cooney, M.D. ; Nishio, S. ; Ishiguro, H.: Recognizing affection for a touch-based interaction with a humanoid robot. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, S. 1420–1427. – ISSN 2153-0858

[11] Coulter, R. C.: Implementation of the Pure Pursuit Path Tracking Algorithm / Robotics Institute. Pittsburgh, PA, January 1992 (CMU-RI-TR-92-01). – Forschungsbericht

[12] Das europäische Parlament und der Rat der europäischen Union: *Verordnung (EG) Nr. 661/2009.* 13. Juli 2009. – URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:2009R0661:20110501:DE:PDF. – (Abgerufen am 15.12.2013)

[13] Dipl.-Ing. Sebastian Rauch and M.Sc. Michael Aeberhard and Dipl.-Ing. Michael Ardelt and Dr.-Ing. Nico Kämpchen: *Autonomes Fahren auf der Autobahn – Eine Potentialstudie für zukünftige Fahrerassistenzsysteme.* 2012

[14] Eike Jenning: Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit / Hochschule für Angewandte Wissenschaften Hamburg. 2008. – Forschungsbericht

[15] Hans-Werner Hunziker: *Im Auge des Lesers: Foveale und periphere Wahrnehmung: Vom Buchstabieren zur Lesefreude.* Stäubli; Auflage: 1., Aufl. (1. Januar 2007), 01. Januar 2007. – ISBN 372660068X

[16] Hauke Schröder: Projektbericht 1: Datensammlung zur Untersuchung der Augenaktorik bei der Steuerung eines Vehikels / Hochschule für Angewandte Wissenschaften Hamburg. Dezember 2012. – Forschungsbericht

[17] Jake Gordon: *How to build a racing game.* 2012. – URL http://codeincomplete.com/posts/2012/6/22/javascript_racer/. – (Abgerufen am 04.07.2013)

[18] Jean-Charles Bornard and Thierry Bellet and Pierre Mayenobe and Dominique Gruyer and Bernard Claverie: A perception module for car drivers' visual strategies modeling and visual distraction effect simulation. In: *Proceedings of the 1st international Symposium on Digital Human Modelling (IEA-DHM), France* (2011)

[19] K.H. Kraft: Gundlagen der Regelungstechnik / Fachhochschule Braunschweig/Wolfenbüttel. – Forschungsbericht

[20] Klein, M. and Kiehl, P.: *Einführung in die DIN-Normen.* Teubner, 2001. – URL http://books.google.de/books?id=r3irIpteJZIC. – ISBN 9783519263012

[21] Kowarschik, Wolfgang: *Multimedia Programmierung*. 2012. – URL http://glossar.hs-augsburg.de/Model-View-Controller-Paradigma#VCM-Paradigma. – (Abgerufen am 06.08.2013)

[22] Logitech: *Logitech Momo Racing Steering Wheel*. – URL http://www.logitech.com/de-de/support/320?osid=14&bit=64. – (Abgerufen am 24.11.2013)

[23] Louis Gorenfeld: *Lou's Pseudo 3d Page*. 2012. – URL http://www.extentofthejam.com/pseudo/. – (Abgerufen am 04.07.2013)

[24] M. F. Land and D. N. Lee: Where we look when we steer. In: *Nature* 369 (1994)

[25] Martin Lundgren: Path Tracking for a Miniature Robot / Department of Computing Science, Umeå University, Sweden. 2003. – Forschungsbericht

[26] Park, Kiyoung ; Lee, Sung J. ; Jung, Ho-Young ; Lee, Yunkeun: Human-robot interface using robust speech recognition and user localization based on noise separation device. In: *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, 2009, S. 328–333. – ISSN 1944-9445

[27] Paul Read and Mark-Paul Meyer: *Restoration of Motion Picture Film (Butterworth-Heinemann Series in Conservation and Museology)* . Butterworth Heinemann, 01. September 2000. – ISBN 075062793X

[28] Pressestelle TU Braunschweig: Weltweit erstes automatisches Fahren im realen Stadtverkehr. 133 (2010). – URL https://www.tu-braunschweig.de/presse/medien/presseinformationen?year=2010&pinr=133. – (Abgerufen am 03.12.2013)

[29] RealVNC: *RealVNC*. – URL https://www.realvnc.com/. – (Abgerufen am 02.09.2013)

[30] Schahram Dustdar and Harald Gall and Manfred Hauswirth: *Software-Architekturen für Verteilte Systeme: Prinzipien, Bausteine und Standardarchitekturen für moderne Software (Xpert.press)*. Springer, 15. Juli 2003. – ISBN 3540430881

[31] Sigalas, M. ; Baltzakis, H. ; Trahanias, P.: Gesture recognition based on arm tracking for human-robot interaction. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010, S. 5424–5429. – ISSN 2153-0858

[32] Signal 11 Software: *HID API*. – URL http://www.signal11.us/oss/hidapi. – (Abgerufen am 04.09.2013)

[33] Statistisches Bundesamt: *Verkehrsunfälle*. 2013. – URL https://www.destatis.de/DE/Publikationen/Thematisch/TransportVerkehr/Verkehrsunfaelle/VerkehrsunfaelleMonat/VerkehrsunfaelleM2080700131074.pdf?__blob=publicationFile. – (Abgerufen am 08.11.2013)

[34] Stückler, Jörg ; Droeschel, David ; Gräve, Kathrin ; Holz, Dirk ; Kläss, Jochen ; Schreiber, Michael ; Steffens, Ricarda ; Behnke, Sven: Towards Robust Mobility, Flexible Object Manipulation, and Intuitive Multimodal Interaction for Domestic Service Robots. In: Röfer, Thomas (Hrsg.) ; Mayer, N.Michael (Hrsg.) ; Savage, Jesus (Hrsg.) ; Saranli, Uluc? (Hrsg.): *RoboCup 2011: Robot Soccer World Cup XV* Bd. 7416. Springer Berlin Heidelberg, 2012, S. 51–62. – URL http://dx.doi.org/10.1007/978-3-642-32060-6_5. – ISBN 978-3-642-32059-0

[35] Teamviewer: *Teamviewer*. – URL http://www.teamviewer.com/de/index.aspx. – (Abgerufen am 02.09.2013)

[36] Theodor Axenfeld and Hans Pau: *Lehrbuch und Atlas der Augenheilkunde*. Elsevier, München; Auflage: 12. völlig neubearb. Aufl. (1980), 1980. – ISBN 3437002554

[37] Thomas J. Triggs and Walter G. Harris: Reaction Time of Drivers to Road Stimuli. In: *Human Factors Report No. HFR-12* (1982)

[38] Tobii Technology: *Eye tracking software - Tobii Studio*. – URL http://www.tobii.com/en/eye-tracking-research/global/products/software/tobii-studio-analysis-software/. – (Abgerufen am 1.08.2013)

[39] Triesch, Jochen ; Malsburg, Christoph Von D.: *Robotic Gesture Recognition*. 1997

[40] TUM Phoenix Robotics: *TUM Phoenix Robotics*. – URL http://www.phoenix.stud.tum.de/auto/. – (Abgerufen am 07.11.2013)

[41] USB Implementers' Forum: Universal Serial Bus (USB) - Device Class Definition for Human Interface Devices (HID) / Firmware Specification. 27. Juni 2001. – Forschungsbericht

[42] VLC media player: *VLC media player*. – URL http://www.videolan.org/vlc/. – (Abgerufen am 02.09.2013)

[43] VolksBot: *entwickelt vom Fraunhofer Institut IAIS*. – URL http://www.volksbot.de/. – (Abgerufen am 23.08.2013)

[44] Waldemar Heck: Hindernis- und Kreuzungserkennung auf autonomen Fahrzeugen durch Kamera und Infrarotsensorik / Hochschule für Angewandte Wissenschaften Hamburg. 2013. – Forschungsbericht

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 20.12.2013 Hauke Schröder