Inhaltsverzeichnis

Tabellen und Abbildungsverzeichnis

Abkürzungsverzeichnis

ESP	Elektronisches Stabilitätsprogramm
ASR	Anti Schlupf Regelung
ABS	Anti Blockier System
XML	Extensible Markup Language
ALU	Aluminium
Leg.	Legierung
DGL	Differentialgleichung
HTWG	Fachhochschule Konstanz
S1	Strecke 1
S2	Strecke 2
S3	Strecke 3
S4	Strecke 4
MKS	Mehr-Körper-System
FEM	Finite-Elemente-Model
KDE	K Desktop Environment
gcc	GNU Compiler Collection
gdb	GNU Debugger
SVN	Subversion
CVS	Concurrent Versions System
Qt	Quasar Toolkit
URL	Uniform Resource Locator
GNU	GNU's Not UNIX
LGPL	Library / Lesser General Public License
GPL	General Public License
usw	Und so weiter
z.B.	zum Beispiel
DIN	Deutsche Industrienorm
GUI	Grafical user interface (grafische Benutzeroberfläche)
API	Application programming interface (Programierschnittstelle)

Verwendete Formelzeichen

$\vec{v}$	[m/s]	Geschwindigkeit
$\vec{v}_0$	[m/s]	Anfangsgeschwindigkeit
$\vec{a}$	[m/s^2]	Beschleunigung
s	[m]	Strecke/Weg
t	[s]	Zeit
F	[N]	Resultierende Kraft
$\vec{F}_N$	[N]	Normalkraft
$\vec{F}_H$	[N]	Haftreibungskraft
$\vec{F}_K$	[N]	Gleitreibungskraft
$\vec{F}_{GH}$	[N]	Hangabtriebskraft
$\vec{F}$	[N]	Kraft
μ_H	[-]	Haftreibungskoeffizient
μ_R	[-]	Rollreibungskoeffizient
μ	[-]	Gleitreibungskoeffizient
$\vec{g}$	[m/s^2]	Beschleunigung im Gravitationsfeld der Erde
F_{SV}	[N]	Seitenführungskraft am Vorderrad
F_{SH}	[N]	Seitenführungskraft am Hinterrad
a_y	[m/s^2]	Querbeschleunigung
a_x	[m/s^2]	Horizontalbeschleunigung
F_X	[N]	Resultierende Kraft in X-Richtung
F_Y	[N]	Resultierende Kraft in Y-Richtung
F_Z	[N]	Resultierende Kraft in Z-Richtung
J_X	[kg/m^2]	Trägheitsmoment der Wankbewegung
J_Y	[kg/m^2]	Trägheitsmoment der Nickbewegung
J_Z	[kg/m^2]	Trägheitsmoment der Gierbewegung
M_X	[Nm]	Wankmoment
M_Y	[Nm]	Nickmoment
M_Z	[Nm]	Giermoment
ψ	[Grad]	Gierwinkel
ϕ	[Grad]	Wankwinkel
Θ	[Grad]	Nickwinkel
l_v	[m]	Abstand vom Schwerpunkt zum Vorderrad
l_h	[m]	Abstand vom Schwerpunkt zum Hinterrad
l	[m]	Spurabstand
l_a	[m]	Länge des Autos
b_a	[m]	Breite des Autos
c_a	[m]	Höhe des Autos
r	[m]	Kurvenradius
α_v	[Grad]	Schräglaufwinkel vorn
α_h	[Grad]	Schräglaufwinkel hinten
c_v	[N/rad]	Schräglaufsteifigkeit vorn
c_h	[N/rad]	Schräglaufsteifigkeit hinten
δ	[Grad]	Lenkwinkel
δ_v	[Grad]	Lenkwinkel vorn

δ_h	[Grad]	Lenkwinkel hinten
β	[Grad]	Schwimmwinkel
ω	[rad/s]	Winkelgeschwindigkeit
α	[rad/s^2]	Winkelbeschleunigung
φ	[Grad]	Drehwinkel
ρ	[kg/m^3]	Dichte
a_T	[m/s^2]	Tangentialbeschleunigung
a_R	[m/s^2]	Radialbeschleunigung
a_Z	[m/s^2]	Zentripetalbeschleunigung
F_Z	[N]	Zentripetalkraft
F_{XVL}	[N]	Kraft in X-Richtung am linken Vorderrad
F_{XVR}	[N]	Kraft in X-Richtung am rechten Vorderrad
F_{XHR}	[N]	Kraft in X-Richtung am rechten Hinterrad
F_{XHL}	[N]	Kraft in X-Richtung am linken Hinterrad
F_{ZVL}	[N]	Kraft in Z-Richtung am linken Vorderrad
F_{ZVR}	[N]	Kraft in Z-Richtung am rechten Vorderrad
F_{ZHL}	[N]	Kraft in Z-Richtung am linken Hinterrad
F_{ZHR}	[N]	Kraft in Z-Richtung am rechten Hinterrad
F_{YVL}	[N]	Kraft in Y-Richtung am linken Vorderrad
F_{YVR}	[N]	Kraft in Y-Richtung am rechten Vorderrad
F_{YHL}	[N]	Kraft in Y-Richtung am linken Hinterrad
F_{YHR}	[N]	Kraft in Y-Richtung am rechten Hinterrad
F_{AH}	[N]	Auftriebskraft hinten
F_{AV}	[M]	Auftriebskraft vorn
b_V	[m]	Abstand der beiden Vorderräder
b_H	[m]	Abstand der beiden Hinterräder
l_V	[m]	Abstand vom Schwerpunkt zur Vorderachse
l_H	[m]	Abstand vom Schwerpunkt zur Hinterachse
h_{SP}	[m]	Höhe des Schwerpunkts
h_{WA}	[m]	Höhe der Wankachse
h_{NA}	[m]	Höhe der Nickachse
S_X	[-]	Umfangsschlupf
μ_X	[-]	Umfangskraftbeiwert
μ_y	[-]	Seitenkraftbeiwert

Zusammenfassung (Abstract)

Thema:	**Carrerabahn - Implementierung einer Simulation zur Verifizierung von Echtzeitprogrammen für die Steuerung von Hardware**
Diplomand:	Marco Fahr
Firma:	HTWG Konstanz
Betreuer:	Prof. Dr. M. Mächtel
Abgabedatum:	07.05.2008
Schlagworte:	Carrerabahn, Echtzeit, Sensoren, Simulation, Signalverarbeitung,

Das Ziel dieser Diplomarbeit besteht darin mittels einer Simulation eine Testumgebung zu schaffen, in der Programme, welche die Aufgabe haben Autos auf einer Carrerabahn zu steuern, auf ihre Zuverlässigkeit und Fehlerfreiheit getestet werden können. Dazu ist es nötig, dass die zu testenden Programme keinen Unterschied während ihrer Ausführung feststellen, ob sie mit der Simulation oder mit der echten Hardware kommunizieren. Aus diesem Grund verwendet die Simulation den Treiber für die Carrerabahn genau in der gleichen Weise wie die reale Carrerabahn dies im Betrieb auch macht. Die Simulation versucht die reale Carrerabahn mit allen ihren Sensoren möglichst exakt nachzubilden.

Kapitel 1

1. Einführung

1.1 Motivation und Zielsetzung

Ziel dieser Arbeit ist die Erstellung einer Simulation in enger Zusammenarbeit mit dem Treiber der simulierten Hardware. Ziel dieser Arbeit ist unter es gängige Lösungen für Simulationsprobleme aus dem Bereich der Fahrdynamik dahingehend zu beurteilen, ob sie eine mögliche Lösung für auftretende Probleme in der Simulation bieten. Insbesondere die Realisierung des Driftsensors kann als eines der Hauptprobleme betrachtet werden.

1.2 Aufbau der Arbeit

Die Diplomarbeit ist in sechs Kapitel gegliedert. Das erste Kapitel besteht aus der Einführung. Im zweiten Kapitel wird auf die der Simulation zu Grunde liegende Physik und Mathematik eingegangen. Die verschiedenen physikalischen Gesetze werden in diesem Kapitel näher betrachtet und erklärt, insbesondere die Gesetze der Bewegungslehre.

Im dritten Kapitel werden der Aufbau und das Layout der realen Carrerabahn beschrieben und erklärt. Es werden die verschiedenen Sensoren in der Bahn besprochen und die Aufteilung der Bahn in verschiedene Abschnitte.

Kapitel vier beschäftigt sich mit dem Treiber und den nötigen Modifikationen um diesen mit der Carrerabahn der HTWG Konstanz zu benutzen. Dazu gehören die Erweiterung auf den Vierspurbetrieb, das Umstellen auf eine neue Strecke und die Verwendung von anderen Hardware und Benutzer Statuswörtern.

Der aktuelle Stand der Technik im Bezug auf die Simulation von nicht schienengebundenen Kraftfahrzeugen wird im fünften Kapitel beleuchtet. Insbesondere Ansätze zur Bildung von Reifen und Fahrzeugmodellen für die Simulation werden hier vorgestellt. Die Modelle werden insbesondere unter dem Aspekt, sie für die Simulation des Driftsensors zu gebrauchen vorgestellt. Des Weiteren werden eigene Ansätze zur Realisierung des Driftsensors vorgestellt.

Im sechsten und letzten Kapitel wird die Entwicklungsumgebung, die benutzten Werkzeuge und Programmiersprachen und Techniken und der Aufbau und Ablauf der Simulation

vorgestellt. Es wird auf die Realisierung und die Konzepte hinter der Simulation eingegangen und wie die einzelnen Aspekte der Simulation wie z.B. Aufbau der Stecke, Modellierung des Autos und der Sensoren verwirklicht wurden.

Kapitel 2

2. Physikalische Grundlagen

In den Folgenden Kapiteln wird an manchen Stellen von Kinematik, Kinetik und Dynamik gesprochen, weshalb ich an dieser Stelle Folgende Begriffsdefinition anführen möchte.

Kinetik

„Die Kinematik beschreibt die Bewegung der Körper, ohne dabei auf die Kräfte als Ursache für die Verschiedenen Bewegungsarten einzugehen.“ [DoKrVo07]

Kinematik

„Die Lehre von den Kräften und Bewegungen nennt man Kinetik. Sie arbeitet mit allen Grundgrößen der Mechanik, mit der Länge, der Kraft und der Zeit“ [Ass04]

Dynamik

„Wenn man nur vom Wortstamm ausgeht (griechisch = Kraft), dann ist die Dynamik die Lehre von den Kräften. So gesehen ist sie ein Oberbegriff für Statik und Kinetik.“ [Ass04]

2.1 Gleichförmige Bewegung

Eine der Aufgaben der Simulation besteht darin, die möglichen Bewegungen der Autos auf der Bahn zu simulieren. Eine dieser Bewegungen ist die gleichförmige Bewegung mit einer konstanten Geschwindigkeit v.

Man unterscheidet zwei Arten von gleichförmigen Bewegungen:

1. Die gleichförmige geradlinige Bewegung
2. Die gleichförmige krummlinige Bewegung

In diesem Abschnitt möchte ich nur auf die erste der gleichförmigen Bewegungen eingehen, da die zweite in einem der Folgenden Abschnitte über die Kreisbewegung behandelt wird.

Folgende drei Gesetzte gelten für die gleichförmige geradlinige Bewegung:

- **Das Weg-Zeit-Gesetzt**

$$s(t) = \vec{v}_0 + \vec{v} * t \tag{2.1}$$

- **Das Geschwindigkeits-Zeit-Gesetzt**

$$\vec{v}(t) = \vec{v} = konstant \tag{2.2}$$

- **Das Beschleunigungs-Zeit-Gesetzt**

$$\vec{a}(t) = 0 \tag{2.3}$$

Es gelten weiterhin noch diese Beziehungen untereinander:

- $\dot{s}(t) = \frac{ds}{dt} = \vec{v}$ (2.4)
- $\ddot{s}(t) = \dot{v}(t) = \frac{dv}{dt} = \frac{d^2s}{dt^2} = \vec{a}$ (2.5)

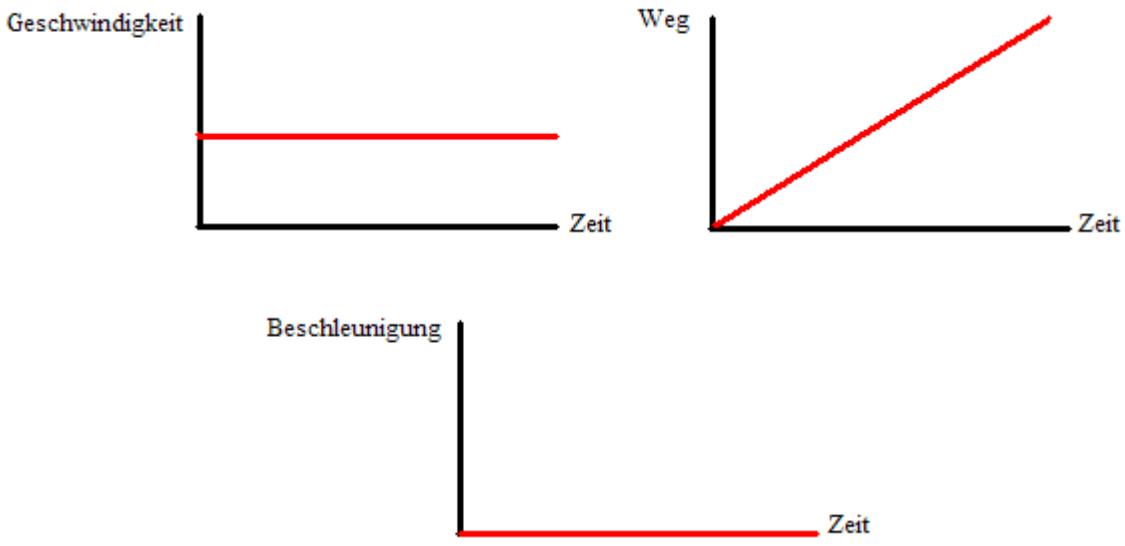

Abbildung 1: Gleichförmige Bewegung

2.2 Gleichförmig beschleunigte Bewegung

Eine weitere Bewegungsart welche durch die Simulation dargestellt werden muss ist die gleichförmig beschleunigte Bewegung. Diese Bewegung tritt immer dann auf wenn das Auto auf der Carrerabahn einen neuen Geschwindigkeitswert übermittelt bekommt. Es ist eine Bewegung mit einer konstanten Beschleunigung$\vec{a}$. Das Auto besitzt im Moment der Geschwindigkeitszuweisung eine Anfangsgeschwindigkeit v_0 und wird durch die Zuweisung auf die neue Geschwindigkeit $\vec{v}$ beschleunigt. Diese Bewegung tritt auch beim Anfahren auf. In diesem Fall hat die Anfangsgeschwindigkeit $\vec{v}_0$ den Wert 0. Auch das abbremsen auf eine niedrigere Geschwindigkeit als die Momentane oder bis zum Stillstand ist eine beschleunigte Bewegung. In diesem Fall hat die Beschleunigung einen negativen Betrag.

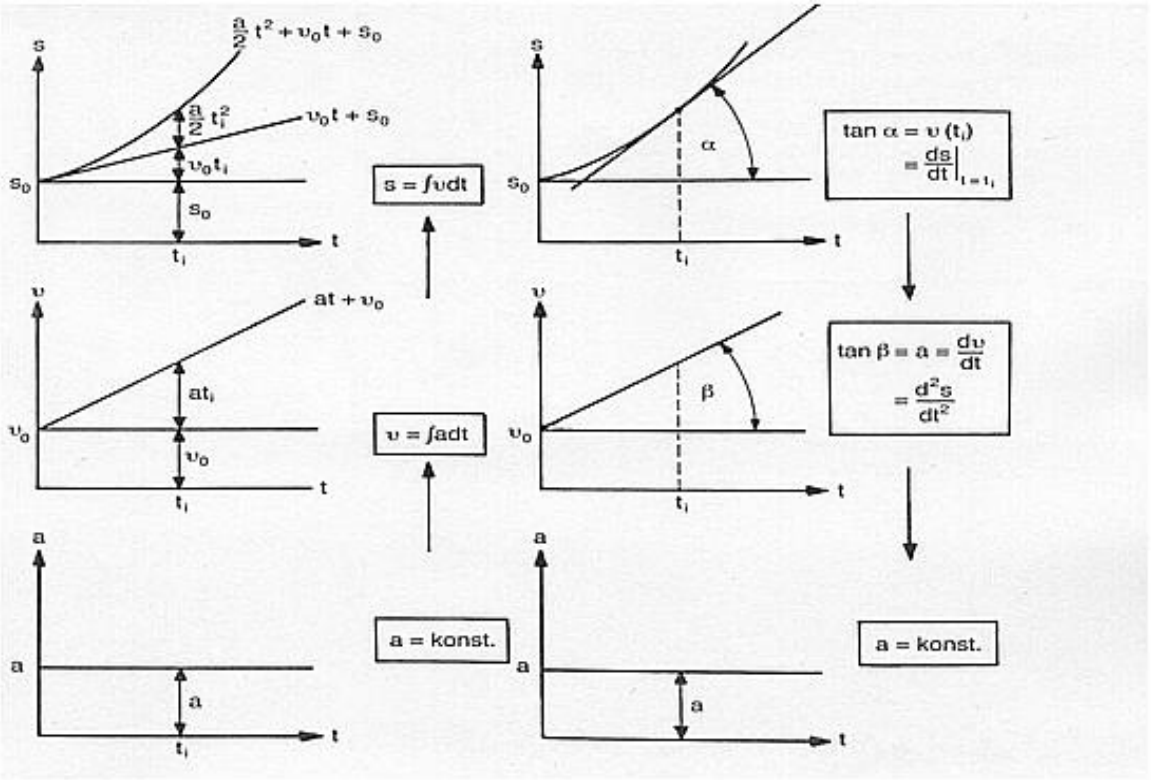

Abbildung 2: Gleichförmig beschleunigte Bewegungen **Quelle [www_erl]**

Folgende drei Gesetzte gelten für die gleichförmige beschleunigte Bewegung:

- **Das Weg-Zeit-Gesetzt**

$$s(t) = s_0 + \vec{v}_0 * t \pm \frac{1}{2} * \vec{a} * t^2 \qquad (2.6)$$

- **Das Geschwindigkeits-Zeit-Gesetzt**

$$\vec{\mathrm{v}}(\mathrm{t}) = \mathrm{v}_0 \pm \vec{\mathrm{a}} * \mathrm{t} \tag{2.7}$$

- **Das Beschleunigungs-Zeit-Gesetzt**

$$\vec{a}(t) = konstant \tag{2.8}$$

Weiterhin gelten noch folgende Beziehungen untereinander:

- $\dot{s}(t) = \frac{ds}{dt} = \vec{v}$
- $\ddot{s}(t) = \dot{v}(t) = \frac{dv}{dt} = \frac{d^2s}{dt^2} = \vec{a}$

2.3 Kreisbewegung und Zentripetalkraft

Bei der Kreisbewegung handelt es sich bei der Anfangs schon erwähnten gleichförmig krummlinigen Bewegung. Jede Kurvenfahrt die von einem Auto auf der Carrerabahn unternommen wird ist eine Kreisbewegung. Bei der konstanten Kreisbewegung werden in gleichen Zeitabständen Δt gleiche Kreisbogenabschnitte Δs durchlaufen. Die Geschwindigkeit $\vec{v}$ ist dabei konstant.

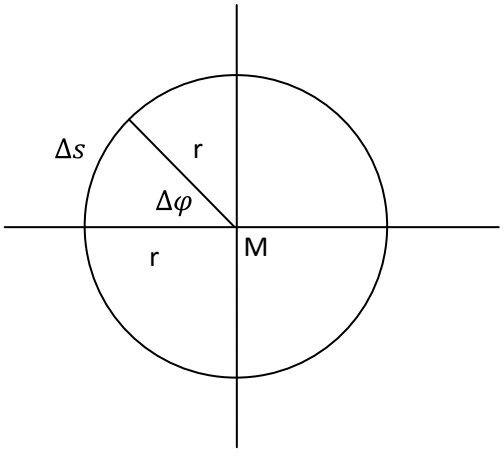

Abbildung 3: Konstante Kreisbewegung

Es gelten die Folgenden Beziehungen:

$$\vec{v} = \frac{\Delta s}{\Delta t} \tag{2.9}$$

$$\vec{v} = \frac{\Delta \varphi}{\Delta \mathrm{t}} * r \tag{2.10}$$

$$\omega = \frac{d\varphi}{dt} = \dot{\varphi} \tag{2.11}$$

(2.11) in (2.10) eingesetzt führt dann zu:

$$\vec{v} = \omega * r = \dot{\varphi} * r \tag{2.12}$$

$$\varphi = \varphi_0 + \omega * t \tag{2.13}$$

Die radiale Beschleunigung ist bei dieser Bewegung immer zum Kreismittelpunkt gerichtet und vom Betrag konstant. Sie lautet:

$$a_R = a_Z = \omega^2 * r = \frac{v^2}{r} \tag{2.14}$$

Unter Berücksichtigung des zweiten newtonschen Axioms $F = m * a$ ergibt sich die Zentripetalkraft der gleichförmigen Kreisbewegung zu:

$$F_Z = \frac{m*v^2}{r} \tag{2.15}$$

Bei der gleichförmig beschleunigte Kreisbewegung gilt für die Winkelbeschleunigung:

$$\vec{\alpha} = \dot{\vec{\omega}} = \ddot{\vec{\varphi}} = \frac{a_T}{r} \tag{2.16}$$

Für den Drehwinkel gilt, analog zu gleichförmig beschleunigter Bewegung, die folgende Beziehung:

$$\varphi_{(t)} = \varphi_0 + \omega_0 * t + \frac{1}{2}\vec{\alpha} * t^2 \qquad (2.17)$$

2.4 Reibung

Wenn sich zwei Körper an einer Stelle berühren und es wird eine horizontale Kraft auf diese Kontaktstelle ausgeübt, so tritt an dieser Stelle Reibung zwischen diesen Objekten auf. Wenn ein z.B. ein Objekt auf einer Straße aufliegt so wird dieses mit der Kraft F_N auf den Untergrund gedrückt. Die Kraft F_N ist der Anteil der Gewichtskraft des Objektes der im rechten Winkel (senkrecht) an der Auflagefläche angreift. Wird nun versucht das Objekt über den Untergrund zu ziehen oder zu drücken, so entsteht die Reibkraft welcher der ziehenden oder drückenden Kraft entgegen wirkt. Diese Kraft verhindert ein Rutschen des Körpers über den Untergrund. Dies nennt man Haftreibung. Wird die Kraft um den Körper in Bewegung zu versetzen erhöht so wird irgendwann eine Schwelle erreicht bei der die Gegenkraft der Haftreibung kleiner ist als die anliegende Kraft. Das Objekt beginnt über den Untergrund zu gleiten und die Haftreibung geht in Gleitreibung über. Der Wiederstand der dem Körper durch das Gleiten auf dem Untergrund entgegengebracht wird ist die Reibkraft.

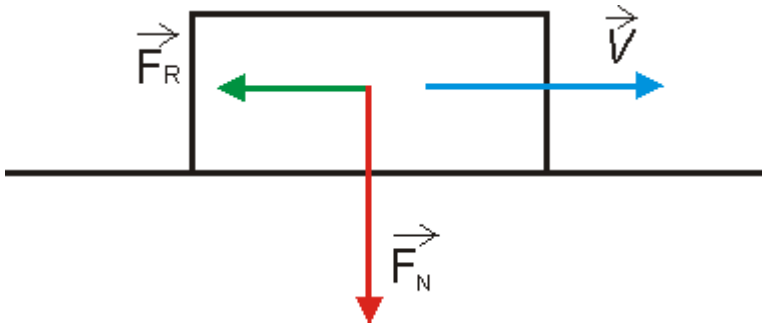

Abbildung 4: Reibung **Quelle [www_wü]**

$$\vec{F}_H \leq \mu_H * \vec{F}_N \qquad (2.18)$$

$$\vec{F}_{Hmax} = \mu_H * \vec{F}_N \qquad (2.19)$$

$$\vec{F}_K = \mu * \vec{F}_N \quad (2.20)$$

μ_H ist der Haftreibungskoeffizient und er beschreibt das Reibungsverhalten zwischen zwei Objekten die nicht in Bewegung sind. μ ist der Gleitreibungskoeffizient welcher das Verhalten der Reibung zwischen zwei Objekten beschreibt die an einer Fläche aneinander gleiten. Damit sich ein Objekt bewegt muss eine Kraft aufgebracht werden die größer als die Haftreibungskraft ist. Ist der Körper in Bewegung so muss nur noch die Gleitreibungskraft aufgebracht werden um den Körper in Bewegung zu halten. Aus diesem Grund gilt immer $\mu \leq \mu_H$.

Die Folgende Tabelle wurde aus [Rob04] entnommen:

Material	Haftreibungskoeffizient μ_H	Gleitreibungskoeffizient μ
Eisen – Eisen	0,45 – 0,80	1,0
Kupfer – Kupfer	0,45 – 0,80	0,60 – 1,0
Stahl – Stahl	0,45 – 0,80	0,40 – 0,70
Chrom – Chrom	0,45 – 0,80	0,41
Nickel – Nickel	0,45 – 0,80	0,39 – 0,70
ALU Leg. – ALU Leg.	0,45 – 0,80	0,15 – 0,60
Stahl – Kupfer	0,45 – 0,80	0,23 – 0,29
Stahl – Weißmetall	0,45 – 0,80	0,21
Stahl – Grauguss	0,18 – 0,24	0,17 – 0,24
Bremsbelag – Stahl		0,50 – 0,60
Leder - Metall	0,60	0,20 – 0,25
Polyamid - Stahl		0,32 – 0,45
PTFE – Stahl		0,04 – 0,22
Eis - Stahl	0,027	0,014

Abbildung 5: Reibungskoeffizienten

2.5 Die schiefe Ebene

Auch die schiefe Eben spielt in der Simulation eine Rolle. Da es auf der Carrerabahn je einen Abschnitt mit einer Bergfahrt und einen Abschnitt mit einer Talfahrt gibt, ist es notwendig sich mit den Gesetzen an der schiefen Ebene vertraut zu machen.

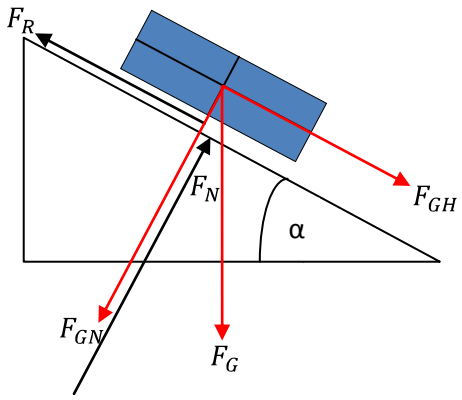

Abbildung 6: Kräfte an der schiefen Ebene

Ein Körper der die schiefe Eben mit einer konstanten Geschwindigkeit hinauffährt beschreibt eine gleichförmig verzögerte Bewegung. Ein Körper der die schiefe Eben hinunterfährt beschreibt eine gleichförmig beschleunigte Bewegung. Die Bewegung kommt dadurch zu Stande weil eine Komponente, die Hangabtriebskomponente $\vec{F}_{GH}$, welche parallel zur schiefen Ebene verläuft den Körper beschleunigt oder abbremst. Die Komponente welche senkrecht zur schiefen Ebene verläuft, die Normalkomponente $\vec{F}_{GN}$ bzw. $\vec{F}_N$ist verantwortlich für Reibungskraft $\vec{F}_R$ die zwischen den Auflageflächen entsteht.

$$\vec{F}_{GN} = \vec{F}_G * \cos\alpha \tag{2.21}$$

$$\vec{F}_{GH} = \vec{F}_G * \sin\alpha \tag{2.22}$$

$$\vec{F}_R = \mu * \vec{F}_{GN} \tag{2.23}$$

Bei der Abwärtsbewegung ist die Gleitreibung der Hangabtriebskraft entgegengesetzt. Bei der Aufwärtsbewegung addieren sich Hangabtriebskraft und Gleitreibungskraft.

Die jeweilige Beschleunigung ergibt sich zu:

- **Aufwärts**

$$\vec{a} = \vec{g}\,(\sin\alpha + \mu * \cos\alpha) \tag{2.24}$$

- **Abwärts**

$$\vec{a} = \vec{g}\,(\sin\alpha - \mu * \cos\alpha) \tag{2.25}$$

Diese Beschleunigung verzögert oder beschleunigt eine gleichförmige Bewegung an der schiefen Ebene. Bei gleichförmig beschleunigten oder verzögerten Bewegungen unterstützt diese Beschleunigungskomponente die Bewegung.

2.6 Trägheitsmoment

Der Trägheitsmoment, auch Massenträgheitsmoment genannt, ist die physikalische Größe die angibt wie sich ein starrer Körper bei einer Veränderung der eigenen Rotation um eine Achse verhält.

Das Trägheitsmoment wird durch die Folgende Summe berechnet, $r_1$ist der senkrechte Abstand des i - ten Teilchens von der Drehachse:

$$J = \sum_i m_i * r_i^2 \tag{2.26}$$

Da in der Simulation der Körper des Autos zu einem Quader vereinfacht wird, ist in diesem Fall nur das Trägheitsmoment der z-Achse von Bedeutung ergibt sich die Formel zu:

$$J = \sum_i m_i * (l_a^2 + b_a^2) \tag{2.27}$$

Hieraus folgt mit $dm = \rho\, dl_a\, db_a\, dh_a$:

$$J_z = \int_m (l_a^2 + b_a^2)\, dm \tag{2.28}$$

$$J_z = \rho \int_{-\frac{h_a}{2}}^{\frac{h_a}{2}} \int_{-\frac{b_a}{2}}^{\frac{b_a}{2}} \int_{-\frac{l_a}{2}}^{\frac{l_a}{2}} (l_a^2 + b_a^2)\, dl_a db_a dh_a \tag{2.29}$$

$$J_z = \rho \int_{-\frac{h_a}{2}}^{\frac{h_a}{2}} \int_{-\frac{b_a}{2}}^{\frac{b_a}{2}} (l_a^3 + b_a^2 * l_a)\ db_a dh_a \tag{2.30}$$

$$J_z = \rho \int_{-\frac{h_a}{2}}^{\frac{h_a}{2}} \left(\frac{1}{12} l_a^3 * b_a + \frac{1}{12} b_a^3 * l_a\right) dh_a \tag{2.31}$$

$$J_z = \rho \left(\frac{1}{12} l_a^3 * b_a * h_a + \frac{1}{12} b_a^3 * l_a * h_a\right) \tag{2.32}$$

$$J_z = \frac{1}{12} \rho * l_a * b_a * h_a (l_a^2 + b_a^2) \tag{2.33}$$

Mit $m = \rho * l_a * b_a * h_a$ ergibt sich das Trägheitsmoment um die z-Achse des Autos zu:

$$J_z = \frac{1}{12} * m * (l_a^2 + b_a^2) \tag{2.34}$$

Dieses Moment wird bei der späteren Betrachtung der dynamischen Fahrzeugmodelle in Kapitel 5 benötigt.

2.7 Kontinuierliche Simulation

Das Ziel einer kontinuierlichen Simulation ist es den zeitlichen Verlauf der Zustandsgrößen eines Modells möglichst exakt zu beschreiben. Als Zustandsgrößen bezeichnet man diejenigen Größen, die durch Aufsummierung eines kleinen Teilbetrags zum aktuellen Wert, innerhalb eines Zeitintervalls, berechnet werden können. Neben diesen Größen existieren noch die Funktionalen Größen des Modells. Diese lassen sich durch Gleichungen mit vorhandenen Werten sofort bestimmen.

Für die Bestimmung des zeitlichen Verlaufs werden in der Regel Differentialgleichungen verwendet. Diese Gleichungen versuchen das dynamische Verhalten mathematisch auszudrücken. Mit diesen Gleichungen wird das physikalische Verhalten der Simulation modelliert und im Rechner abgebildet. Der Zustand der Simulation wird durch Integration der abgeleiteten Zustandsgrößen ermittelt.

Der Funktionsverlauf der Simulation ist nur zu bestimmten Zeitpunkten genau gegeben und muss deshalb zwischen diesen Zeitpunkten durch Funktionen ermittelt werden. Mit Hilfe dieser Funktionen ist es möglich den Simulationsablauf kontinuierlich zu berechnen. Dies geschieht durch Integration dieser Funktionen. Ausgehend von einem Startzustand kann so das Modell der Simulation das Verhalten zu jedem beliebigen Zeitpunkt berechnen. Modellgrößen, wie z.B. Geschwindigkeiten, Beschleunigungen, Drehmomente oder Kräfte die die Veränderung des Modells bewirken, werden während der Simulation zwischen zwei Zeitpunkten als konstant angenommen, da eine genauere Bestimmung in der Regel nicht möglich ist.

2.8 Numerische Integration nach Euler[1]

Integrale müssen auf dem Rechner numerisch gelöst werden. Das einfachste und auch bekannteste Verfahren ist die Integration nach Euler. Dieses Verfahren ist auch als Polygonzug- oder Euler-Cauchy-Verfahren bekannt. Von einem Startwert ausgehend werden in einem bekannten Zeitintervall, für eine definierte Schrittweite, Stützstellen berechnet und die Funktion zwischen diesen Stützstellen linear interpoliert. Die so berechneten Werte sind nur Nährungen, je kleiner jedoch die Schrittweite gewählt wird, desto genauer liegen diese Werte bei dem simulierten realen System.
Wird einem Körper mit der Masse m, der Position s und der Geschwindigkeit v eine Kraft F über einen Zeitraum t zugeführt, so erfährt dieser Körper eine Beschleunigung die durch die folgende Gleichung gegeben ist:

$$F_{(t)} = m * a_t = a_t = \frac{F_{(t)}}{m} \tag{2.35}$$

Mit den Gleichungen

$$v_{(t)} = v_0 + a * t \equiv v_{(t+\Delta t)} = v_{(t)} + \int_t^{t+\Delta t} a_{(t)} dt \tag{2.36}$$

und

$$s_{(t)} = s_0 + v * t \equiv s_{(t+\Delta t)} = s_{(t)} + \int_t^{t+\Delta t} v_{(t)} dt \tag{2.37}$$

[1] Vgl. [Her04]

sind die Geschwindigkeit und die Position des Körpers, in Abhängigkeit von der Beschleunigung, zu jedem Zeitpunkt exakt bestimmt. Auf Grund der Tatsache, dass der Verlauf der Beschleunigung über der Zeit nicht bekannt ist, sind die Integrale nicht exakt bestimmbar. Die Beschleunigung kann nur zum Zeitpunkt t und den vorherigen Zeitpunkten bestimmt werden. Durch das Integrationsverfahren wird nun versucht sich dem wirklichen Verlauf der Beschleunigung möglichst exakt anzunähern. Hier kommen die numerischen Integrationsverfahren ins Spiel. Eines der einfachsten ist das Euler-Verfahren. Mit dessen Hilfe wird nun der Verlauf der Beschleunigung zwischen den berechneten Stützpunkten linear interpoliert.

Die lineare Interpolation ist durch die Folgenden zwei Gleichungen gegeben:

$$v_{(t+\Delta t)} = v_{(t)} + \frac{a_{(t)} - \frac{a_{(t+\Delta t)} - a_{(t)}}{2}}{\Delta t} \tag{2.38}$$

$$s_{(t+\Delta t)} = s_{(t)} + \frac{v_{(t)} - \frac{v_{(t+\Delta t)} - v_{(t)}}{2}}{\Delta t} \tag{2.39}$$

Die Qualität des Ergebnisses, sprich die Größe des gemachten Fehlers durch die lineare Interpolation, hängt von der Größe der gewählten Schrittweite Δt ab. Je kleiner die Schrittweite gewählt wird, desto kleiner der Fehler.

Kapitel 3

3. Carrerabahn Aufbau

3.1 Segmente

Bei der aufgebauten Rennbahn im Labor für Echtzeitsysteme der HTWG Konstanz handelt es sich um das Modell Exklusiv der Firma Carrera. Auf der Bahn können Fahrzeuge im Maßstab 1:24 und 1:32 betrieben werden. Ein Element der Bahn ist 198 mm breit und der Spurabstand beträgt 100 mm. Die Bahn ist mit vier Spuren aufgebaut. Die Gesamtbreite der Carrerabahn summiert sich dadurch auf 396 mm plus eventuell angebrachte Seitenstreifen für die Leitplanken. Die Carrerabahn ist in insgesamt neun Segmente aufgeteilt. Zwei Segmente beinhalten je eine Kreuzung. In einem Segment befindet sich eine Steigung, im anschließenden Segment das Gefälle. Die Bahn verfügt über insgesamt zehn Kurven und eine Brücke.

Das erste Segment besteht nur aus einer langen Geraden. Es hat seinen Anfang nach dem Start/Ziel Element und endet an der ersten Kurve. Es besteht aus zwei 1/3 Geraden mit je 115mm Länge und sechs Standardgeraden mit je 345 mm Länge. Diese Bauteile werden für die Spuren eins, zwei, drei und vier verwendet.

Das zweite Segment besteht aus einer 210° Rechtskurve. Die Spuren eins und zwei sind aus sieben 2/30° Kurvenelementen, die Spuren drei und vier aus drei 1/60° und einem 1/30° Kurvenelement aufgebaut.

Segment drei besteht aus einer weiten 120° Linkskurve gefolgt von einer kurzen 30° Rechtskurve. Spur eins und zwei bestehen aus zwei 1/3 Geraden mit je 115 mm Länge, vier 2/30° und einem 3/30° Kurvenelement, Spur drei und vier auch aus zwei 1/3 Geraden, vier 3/30° und einem 2/30° Kurvenelement.

Das vierte Segment besteht aus einem weiten 360° Linkskurve mit zusätzlicher Steigung. Spur eins und zwei sind aus einer 1/3 Geraden und zwölf 2/30° Kurvenelementen aufgebaut. Die Spuren drei und vier aus einer 1/3 Geraden und zwölf 3/30° Kurvenelementen.

Segment fünf beinhaltet eine Kreuzung und eine kurze 30° Linkskurve. Die Spuren eins und zwei sind aus zwei 1/3 Geraden, einer Kreuzung mit 345 mm Länge, einem 3/30° Kurvenelement und einer 345 mm langen Standardgeraden aufgebaut. Spur drei und vier bestehen auch aus zwei 1/3 Geraden, einer Kreuzung, einem 4/15° Kurvenelement und einer Standardgeraden.

Segment sechs besteht aus einer 240° Rechtskurve mit Gefälle und einer 30° Linkskurve. Die erste und die zweite Spur sind aus acht 4/15° Kurvenelementen, zwei Standardgeraden, einem 3/30° Kurvenelement und einem 1/4 Geradenelement mit 00 mm Länge zusammengebaut. Spur drei und vier aus acht 3/30° Kurvenelementen, zwei Standardgeraden, einem 4/15° Kurvenelement und einem 1/4 Geradenelement.

Das siebte Segment ist eine 180° Linkskurve. Spur eins und zwei bestehen aus einer Standardgeraden und drei 1/60° Kurvenelementen. Die Spuren drei und vier aus einer Standardgeraden und sechs 2/30° Kurvenelementen.

Im achten Segment befindet sich wieder eine Kreuzung und eine 120° Rechtskurve. Die erste und zweite Spur bestehen aus einer Standardgeraden, einer Kreuzung, vier 2/30° Kurvenelementen und einer 1/3 Geraden. Die dritte und vierte Spur aus einer Standardgeraden, einer Kreuzung, zwei 1/60° Kurvenelementen und einem 1/3 Geradenelement.

Im neunten und letzten Segment befindet sich eine 120° Rechtskurve. Die Spuren eins und zwei bestehen aus einer Standardgeraden, vier 3/30° Kurvenelementen, zwei Standardgeraden und einem Start/Ziel-Element mit 345 mm Länge. Die Spuren drei und vier aus einer Standardgeraden, vier 2/30° Kurvenelementen, zwei Standardgeraden und einem Start/Ziel-Element.

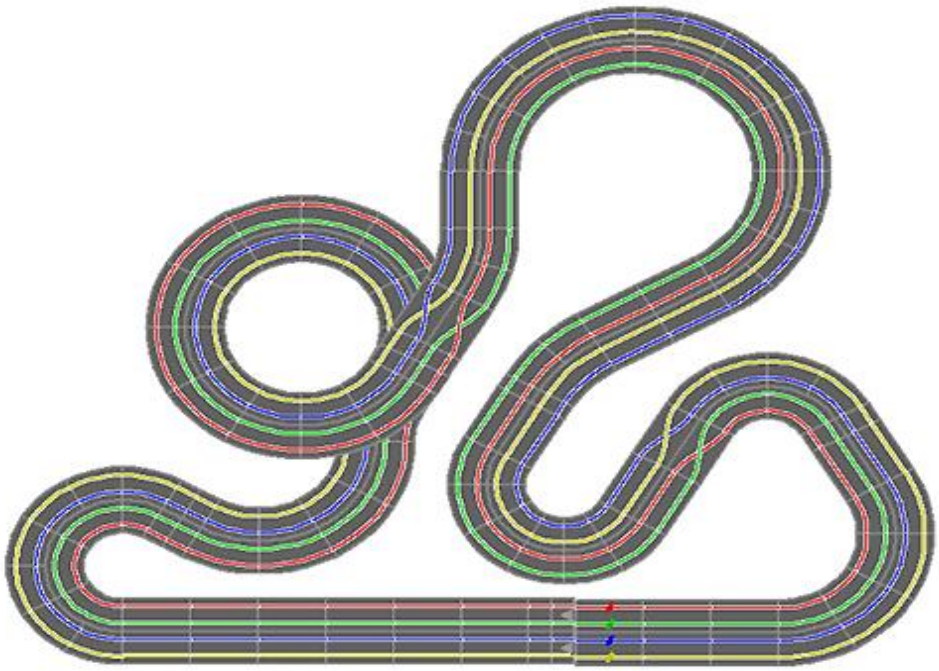

Abbildung 7: Aufbauplan der Carrerabahn

3.2 Sensoren

Die Carrerabahn ist mit zwei Arten von Sensoren ausgestattet. Beim ersten Sensortyp handelt es sich um eine Lichtschranke. Die Lichtschranken sind immer am Anfang vor dem jeweiligen Segment angebracht. Mit ihnen ist es möglich die Position und die Geschwindigkeit eines Autos auf der Bahn zu bestimmen. Die Segmente der Carrerabahn werden durch die Position der Sensoren vorgegeben. Immer zwischen zwei Sensoren ist ein Segment. Es gibt insgesamt neun Sensoren und die Bahn ist auch in neun Segmente aufgeteilt. Der erste Sensor wurde direkt nach dem Start in der Mitte der ersten 1/3 Geraden angebracht. Der zweite Sensor wurde in der Standartgeraden vor der ersten Kurve angebracht. Sein Abstand beträgt 86 mm vom Anfang der Standartgeraden. Der dritte Sensor wurde in der Mitte der ersten 1/3 Geraden nach der ersten Kurve angebracht. Die vierte Lichtschranke wurde in der Mitte der 1/3 Geraden vor der dritten Kurve, dem Vollkreis, angebracht. Lichtschranke Nummer fünf wurde in der Mitte der 1/3 Geraden nach dem Vollkreis angebracht. Die sechste Lichtschranke wurde in der Standartgeraden vor der vierten Kurve angebracht. Der Abstand beträt 86 mm vom Kurvenanfang in die Standartgerade hinein. Der siebte Sensor wurde in der Standartgeraden vor der fünften Kurve angebracht. Sein Abstand beträgt 86 mm vom Kurvenanfang in die Standartgerade hinein. Sensor Nummer acht wurde in die Standartgerade vor der Kreuzung des achten Segments angebracht. Hier beträgt der Abstand auch 86 mm vom Anfang der Kreuzung in die Standartgerade hinein. Die letzte Lichtschranke wurde 86 mm vor der letzten Kurve in der Standartgeraden angebracht.

Der zweite Sensortyp ist ein sogenannter Driftsensor. Auf Grund der Tatsache, dass während der Erstellung der Diplomarbeit noch keine Driftsensoren installiert waren, wurde in Absprache mit Prof. Mächtel die Annahme getroffen, dass die Sensoren immer bei 40 Grad in einer Kurve angebracht sind. Ein solcher Driftsensor ermittelt die Auslenkung des Autos beim driften in der Kurve. Sein Wertebereich liegt zwischen eins und vierzehn. Gemessen wird mit diesem Sensor nur die Auslenkung nach außen. Auf der fertigen Bahn sind insgesamt sieben Driftsensoren verbaut. Wird ein Driftsensor ausgelöst, so werden zwei Statuswörter von der Carrerabahn an den Treiber übermittelt. Das erste Statuswort enthält Informationen über das aktuelle Segment, ob eine Lichtschranke ausgelöst wurde, ob ein Driftsensor ausgelöst wurde und vieles mehr. Das zweite Hardware Statuswort wird nur übertragen wenn ein Driftsensor ausgelöst wurde. In diesem Statuswort ist die Information über welchen Sensor es sich handelt und welchen Wert dieser Sensor angenommen hat kodiert.

Kapitel 4

4. Treiber

4.1 Anpassungen

Eine der ersten Tätigkeiten während der Diplomarbeit bestand im studieren der Funktionsweise des Treiber, welcher mir von Prof. Mächtel zur Verfügung gestellt wurde. Die Version musste um einige Funktionalität erweitert werden, da sie für eine andere Rennbahn gedacht war. In der Ursprünglichen Version war der Treiber für eine Rennbahn mit nur zwei Spuren geschrieben worden. Es mussten also Gerätedateien für zwei zusätzliche Spuren angelegt werden. Diese Änderung betraf die Gerätedateien für die Steuerprogramme und die Gerätedateien für die Simulation. Es wurden also zusätzliche vier neue Gerätedateien eingebaut. Im Treiber wie auch in der Simulation wird eine Spur durch eine Struktur dargestellt. Diese Struktur wurde so erweitert, dass eine Spur nicht nur Informationen über den direkten Gegner in Form seines Hardware Statuswortes hat, sondern von allen an einem Rennen beteiligten Spuren. Diese Struktur musste auch um das zweite Hardware Statuswort für den Driftsensor erweitert werden. In diesem Zuge wurde auch die Routine die für den Austausch der Statuswörter zuständig ist so überarbeitet, dass der Austausch der Hardware Statuswörter zwischen den Spuren auch mit der neue Carrerabahn mit vier Spuren funktioniert.

Im nächsten Schritt musste dem Treiber der Aufbau der Carrerabahn an der HTWG bekannt gemacht werden. Die Bahn wurde als ein Feld mit so vielen Elementen wie es Segmente auf der Bahn gibt realisiert. Jedem Segment wurde ein hexadezimaler Wert gegeben der mit den Werten der Hardware Statuswörtern der Segmente in der Simulation übereinstimmt. So ist gesichert wenn ein Statuswort von der Simulation an den Treiber geschickt wird, dass der Treiber herausfinden kann in welchem Segment das Auto sich gerade befindet. Im nächsten Schritt mussten die Routinen angepasst werden welche die Statuswörter die von der Simulation an den Treiber geschrieben werden auswerten. Da das 32 Bit Hardware Statuswort eine andere Belegung hat wie die für den der Treiber Ursprünglich geschrieben wurde.

Da bei der Übertragung der Driftsensorwerte der HTWG-Carrerabahn zwei Statuswörter benötigt werden, wurde der Treiber so umgeschrieben, dass er mit einem Feld von Statuswörtern umgehen kann. Der Treiber kann dann auf das zweite Element des Feldes zugreifen wenn er das zweite Statuswort benötigt. Auf diese Weise ist Sichergestellt, dass beide Statuswörter mit einem write()- Aufruf der Simulation gleichzeitig beide Statuswörter

übertragen werden und die Simulation nicht zweimal den write()-Aufruf hintereinander ausführen muss. Es wurde noch ein Fehler behoben, der es nicht erlaubte den Treiber ein zweites Mal zu laden nachdem er schon einmal vom Betriebssystem geladen wurde. Durch den Fehler konnten beim erneuten Laden keine Gerätedateien für die Simulation und die Steuerprogramme angelegt werden, da beim löschen zuvor die Nummern der Geräte nicht richtig freigegeben wurden.

Der Treiber kommuniziert mit dem Steuerprogramm eines Autos über ein sogenanntes Benutzer Statuswort. Immer wenn von der Bahn ein Hardware Statuswort an den Treiber geschickt wird, weil sich z.B. das aktuelle Segment geändert hat oder ein Sensor ausgelöst wurde, wird daraufhin vom Treiber ein Benutzer Statuswort generiert und auf die zuständige Gerätedatei geschrieben. Die Routine die dieses Benutzer Statuswort generiert musste auch an die neue Syntax, welche die HTWG-Bahn und ihre Steuerprogramme verwendet angepasst werden.

Kapitel 5

5. Modell- und Simulationslösungen

In diesem Kapitel sollen verschiedene Modelle die als Lösung für die Implementierung des Driftsensors geeignet sind diskutiert werden. Die Modelle werden heute in allen Bereichen der Fahrdynamik und der Simulation von Fahrzeugen angewendet und die meisten sind stellen den heutigen Stand der Technik dar. Fahrassistenzsysteme wie ESP, ASR und DSC basieren im Wesentlichen auf diesen Modellen. Mit diesen Modellen ist eine Grundlage geschaffen worden, Kraftfahrzeuge physikalisch korrekt im Computer abzubilden. Es wird in diesem Kapitel jedoch nur auf Modelle mit maximal zwei Freiheitsgraden eingegangen. Modelle die mehr Freiheitsgrade bieten, wie z.B. die MKS-Modelle oder die FEM-Modelle werden nicht vorgestellt, da diese mehr bieten als für die Lösung des Simulationsproblems benötigt wird.

Für die Simulation ist besonders die Querdynamik dieser Modelle entscheidend, da diese die Kräfte und Bewegungen beschreiben welche an Autos beim Driftvorgang in der Carrerabahn vorherrschen.

Die meisten der diskutierten Modelle beschreiben leider nur das Verhalten des Autos bis in den kritischen Bereich jedoch nicht über den Grenzbereich hinaus.

5.1 Radumfangsschlupf[2]

Im Allgemeinen bedeutet Schlupf das Verhalten eines mit festgelegter Drehzahl rotierenden Körpers zu einem oder mehreren anderen Bauteilen nicht ganz synchron zu rotieren. Eines der besten Beispiele hierfür wäre ein Riemenantrieb. Der Riemen rotiert mit einer geringfügig anderen Drehzahl wie die Welle die ihn antreibt. Die Kraft die von einem Reifen übertragen werden kann hängt direkt vom sogenannten Reifenschlupf ab. Dieser Schlupf ist nach DIN 70000 folgendermaßen definiert:

$$S_X = \frac{\omega - \omega_0}{\omega_0} \tag{5.1}$$

[2] Vgl. [HeEr07] S. 36 - 37

In der Formel bezeichnet $\omega_0$die Drehgeschwindigkeit eines geradeaus frei rollenden Rades und ω bezeichnet den Schlupf eines Rades.

Nach DIN 7000 sind der Umfangs- und Seitenkraftbeiwert folgendermaßen definiert:

$$\mu_X = \frac{F_X}{F_N} \tag{5.2}$$

$$\mu_X = \frac{F_Y}{F_F} \tag{5.3}$$

5.2 Lineares Reifenmodell

Das einfachste der bekannten Reifenmodelle zur Berechnung der Reifenseitenkraft, ist das lineare Reifenmodell. Es behält seine Gültigkeit bei kleinen Schräglaufwinkeln. Dem Modell liegt die Vereinfachung zu Grunde, dass die Seitenführungskraft des Reifens durch die Folgende Beziehung gegeben ist:

$$F_y = c_\alpha * \alpha \tag{5.4}$$

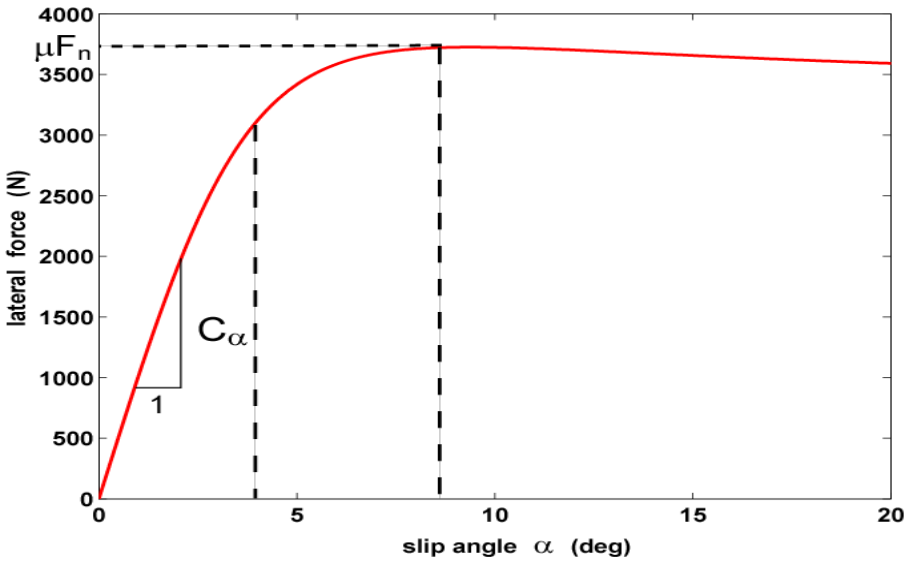

Abbildung 8: Seitenführungskraft in Abhängigkeit des Schwimmwinkels Quelle [JudGer05]

Bei diesem Modell ist steht c_α für den Seitenkraftbeiwert und α für den Schwimmwinkel. Der Seitenkraftbeiwert hängt vom Radumfangsschlupf S_X des jeweiligen Rades ab. Das bedeutet der Seitenumfangsschlupf hat dieselbe Steigung wie der lineare Anteil des Radumfangsschlupf. Da dieser lineare Bereich nur begrenzt eingesetzt werden kann sind die Möglichkeiten mit diesem Modell sehr begrenzt. Das Verhalten eines Autos das den Grenzbereich überschritten hat lässt sich diesem Modell nicht simulieren. Sein Haupteinsatzgebiet liegt beim linearen Einspurmodell.

5.3 Reifenmodell nach Pacejka

Ein Ansatz der auf trigonometrischen Funktionen basiert ist die sogenannte „Magic Formula“ von Pacejka.

$$y = D * \sin[C * \tan^{-1}(B * x - \tan^{-1} B * x)] \tag{5.5}$$

Die Formel wird hier nur der Vollständigkeit halber vorgestellt, da sie ein häufig genutztes Modell in der Simulation von Kraftfahrzeugen ist. Sie scheidet aber als Möglichkeit für die Simulation aus. Je nach dem was mit der Formel berechnet wird, entweder die lateralen Reifenkräfte oder die longitudinalen Reifenkräfte, werden die einzelnen Faktoren D, C und B entweder aus fünfzehn oder elf Reifenkennwerten berechnet. Die Kennwerte werden durch die Reifenhersteller empirisch in Versuchen ermittelt. Da man die Vollgummireifen eines Carreraauto schlecht mit den luftgefüllten Reifen eines Kraftfahrzeugs vergleichen kann, wird dieses Modell nicht weiter betrachtet. Der interessierte Leser sei auf [Bec01] verwiesen. In den Kapiteln 21, 22 und 29 wird besonders auf die Verwendung der Magic Formula eingegangen

5.4 Lineares Einspurmodell[3]

Das am häufigsten bei der Realisierung von Simulationen verwendete Modell für ein Fahrzeug ist das lineare Einspurmodell. Dieses Modell besitzt einen Freiheitsgrad welcher durch die Längsdynamik gegeben ist. Es entstand durch die Arbeit von Riekert und Schunk 1939 [RieSchu39].

Dieses Modell ist Stand der Technik im Bereich der linearen Theorie der Fahrzeugtechnik. Es ist ein stark vereinfachtes Modell bei dem der Schwerpunkt des Autos in die Fahrbahnebene gelegt wird. Die Räder einer jeden Achse werden in der Mitte zu einem Rad zusammengefasst, daher auch die Bezeichnung Einspurmodell. Dieses Modell behält

[3] Vgl. [HeEr07] S. 90 - 95

grundsätzlich seine Gültigkeit solange Querbeschleunigungen von ca. 4 m/s^2 nicht überschritten werden. Desweiteren behält es nur seine Gültigkeit wenn die Bewegungsgleichungen linearisiert werden, so dass angenommen werden kann es gilt $\sin(\alpha) \approx \alpha$ und $\cos(\alpha) \approx 1$. Darüber hinaus behält es seine Gültigkeit nur bei trockener Fahrbahn. Es treten keine Beschleunigungen in Längsrichtung. Hub-, Nick- und Wankbewegungen werden nicht berücksichtigt. Die Radlasten an den Vorder- und Hinterachsen bleiben konstant, es tritt also kein Lastwechsel auf. Für eine eventuelle Verwendung in der Simulation ist das dynamische Verhalten des linearen Einspurmodells von Bedeutung. Hier ist die Betrachtung des dynamischen Gleichgewichts in der Querrichtung und um die Hochachse maßgebend.

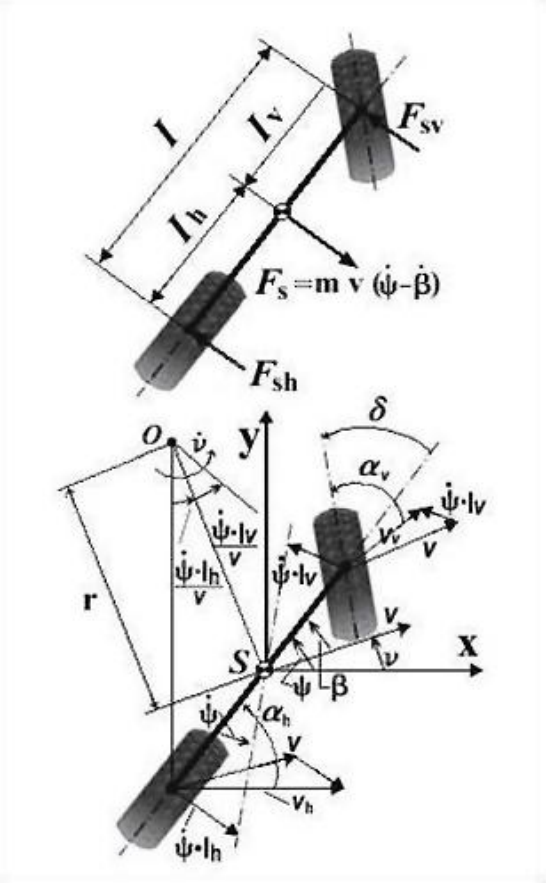

Abbildung 9: Lineares Einspurmodell **Quelle [HeEr07]**

Im ersten Schritt werden die Gleichgewichtsbedingungen des linearen Einspurmodells aufgestellt. Kräftegleichgewicht der Längsachse:

$$m * a_y = F_{SV} + F_{SH} \tag{5.6}$$

Momentengleichgewicht:

$$J_z * \ddot{\psi} = F_{SV} * l_v + F_{SH} * l_h \tag{5.7}$$

Die Näherung der Schräglaufwinkel für kleine Lenkwinkel und Giergeschwindigkeiten:

$$\alpha_v = \beta + \delta - l_v * \frac{\dot{\psi}}{v} \tag{5.8}$$

$$\alpha_h = \beta + l_h * \frac{\dot{\psi}}{v} \tag{5.9}$$

Das lineare Reifenmodell:

$$F_{SV} = c_v * \alpha_v \tag{5.10}$$

$$F_{SH} = c_h * \alpha_h \tag{5.11}$$

Die Beschleunigung ergibt sich zu:

$$a_y = \frac{v^2}{r} = \frac{v}{r} * v = \frac{v}{r} * r * \left(\dot{\psi} - \dot{\beta}\right) = v\left(\dot{\psi} - \dot{\beta}\right) \tag{5.12}$$

Durch einsetzen von (5.5), (5.6), (5.7), (5.8) in (5.3) und (5.4) ergibt sich das folgende lineare Differentialgleichungssystem für den Schwimmwinkel und die Gierwinkelgeschwindigkeit:

$$\dot{\beta} = -\frac{c_h + c_h}{m * v} * \beta + \left(1 - \frac{c_h * l_h - c_v * l_v}{m * v^2}\right) * \dot{\psi} - \frac{c_v}{m * v} * \delta \tag{5.13}$$

$$\ddot{\psi} = \frac{c_v * l_v - c_h * l_h}{J_z} * \beta - \frac{c_h * l_h^2 + c_v * l_v^2}{J_z * v} * \dot{\psi} + \frac{c_v * l_v}{J_z} * \delta \quad (5.14)$$

Da der Lenkwinkel δ bei den Autos der Carrerabahn immer null ist und der Drehpunkt nicht im Schwerpunkt des Autos, sondern an der Finne an der Vorderachse liegt, ergeben sich die Differentialgleichungen zu:

$$\dot{\beta} = -\frac{c_h}{m * v} * \beta + \left(1 - \frac{c_h * l}{m * v^2}\right) * \dot{\psi} \quad (5.15)$$

$$\ddot{\psi} = \frac{c_h * l}{J_z} * \beta - \frac{c_h * l^2}{J_z * v} * \dot{\psi} \quad (5.16)$$

Mit diesem Gleichungssystem lassen sich nun Schwimmwinkel und die Gierwinkelgeschwindigkeit eines Autos, mittels numerischer Integration nach Euler oder Range-Kutta, zu jedem Simulationszeitpunkt bestimmen. Als Eingangsgröße fungiert hier der Schwimmwinkelβ und die Giergeschwindigkeit $\ddot{\psi}$.

Da der Lenkwinkel δ bei den Autos der Carrerabahn immer null ist ergeben sich die Differentialgleichungen zu

Aufgrund der Tatsache, dass dieses Modell jedoch nur für Querbeschleunigungen konzipiert ist welche $\leq$ 0,4 g betragen, ist es für die Simulation des Driftsensors eher ungeeignet. Desweiteren hat ein Auto der Carrerabahn eine starre Vorderachse und somit keine Lenkwinkel. Auch basiert dieses Modell auf der stationären Kreisfahrt, die nie über den Grenzbereich einer Kurvenfahrt hinausgeht und ist somit meines Erachtens für den konkreten Fall der Carrerabahn ungeeignet. Bei der Bestimmung des Driftwinkels wird ja gerade das Verhalten eines Autos bei extremer Überschreitung des Grenzbereichs untersucht und in diesem Fall scheint mir das vorgestellte Modell mehr als ungeeignet und es wurde von einer Implementierung abgesehen.

5.5 Nichtlineares Einspurmodell[4]

Werden größere Schwimmwinkel und Schräglaufwinkel in der Bewegung erreicht, so wird das lineare Einspurmodell ungenau, da die auftretenden Reifenseitenkräfte nicht mehr als linear angesehen werden können. In diesem Fall wird das Modell des linearen Einspurmodells zum nichtlinearen Einspurmodell erweitert.

[4] Vgl. [HeEr07] S.90 - 96

Mit diesem Modell ist es möglich die Querdynamik eines Fahrzeugs auch über den linearen Bereich hinaus zu simulieren. Bei der bisherigen Betrachtung des linearen Einspurmodells wurden die Reifenseitenkräfte senkrecht zur Fahrzeugmittelachse angenommen. Diese greifen jedoch senkrecht zu den Vorder- und Hinterradachse an.

Bei Betrachtung dieses Sachverhaltes kommt man unter Verwendung von Gleichung (5.6) zu den Folgenden beiden Gleichungen für die Längs- und Querrichtung:

$$m * a_x = -F_{SV} * \sin \delta_v - F_{SH} * \sin \delta_h \tag{5.17}$$

$$m * a_y = F_{SV} * \cos \delta_v + F_{SH} * \cos \delta_h \tag{5.18}$$

Durch Aufspaltung der Kraft in eine laterale und eine longitunale Komponente, ergeben sich die Beschleunigungen mit Hilfe von:

$$\vec{a} = \dot{v} + \frac{v^2}{r} \tag{5.19}$$

zu:

$$a_x = \dot{v} * \cos \beta + v\left(\dot{\psi} - \dot{\beta}\right) * \sin \beta \tag{5.20}$$

$$a_y = -\dot{v} * \sin \beta + v\left(\dot{\psi} - \dot{\beta}\right) * \cos \beta \tag{5.21}$$

Eingesetzt in (5.17), (5.18) und jeweils nach $\dot{v}$ umgestellt ergibt sich draus:

$$\dot{v} = \frac{F_{SV} * \sin \delta_v + F_{SH} * \sin \delta_h}{m * \cos \beta} + \frac{v(\dot{\psi} - \dot{\beta}) * \sin \beta}{\cos \beta} \tag{5.22}$$

$$\dot{v} = \frac{F_{SV} * \cos \delta_v + F_{SH} * \cos \delta_h}{m * \sin \beta} - \frac{v(\dot{\psi} - \dot{\beta}) * \cos \beta}{\sin \beta} \tag{5.23}$$

Gleichstellen liefert:

$$\frac{F_{SV}*\cos\delta_v + F_{SH}*\cos\delta_h}{m*\sin\beta} - \frac{F_{SV}*\sin\delta_v + F_{SH}*\sin\delta_h}{m*\cos\beta} = v\left(\dot{\psi} - \dot{\beta}\right) * \left[\frac{\cos\beta}{\sin\beta} + \frac{\sin\beta}{\cos\beta}\right] \quad (5.24)$$

Unter zu Verwendung des Satz des Pythagoras mit $\sin^2\alpha + \cos^2\alpha = 1$ und erweitern der rechten Seite ergibt sich:

$$\frac{(F_{SV}*\cos\delta_v + F_{SH}*\cos\delta_h)*\cos\beta}{m*\cos\beta*\sin\beta} - \frac{(F_{SV}*\sin\delta_v + F_{SH}*\sin\delta_h)*\sin\beta}{m*\sin\beta*\cos\beta} = \frac{v(\dot{\psi}-\dot{\beta})}{\sin\beta*\cos\beta} \quad (5.25)$$

$$\frac{F_{SV}[\cos\delta_v*\cos\beta - \sin\delta_v*\sin\beta]}{m*\cos\beta*\sin\beta} + \frac{F_{SV}[\cos\delta_h*\cos\beta - \sin\delta_h*\sin\beta]}{m*\sin\beta*\cos\beta} = \frac{v(\dot{\psi}-\dot{\beta})}{\sin\beta*\cos\beta} \quad (5.26)$$

Das Additionstheorem $\cos(\alpha+\beta) = \cos\alpha * \cos\beta - \sin\alpha * \sin\beta$ auf die Gleichung (5.21) angewandt, die beiden Gleichungen (5.10) und (5.11) eingesetzt, führen zu der Differentialgleichung für den Schwimmwinkel:

$$\dot{\beta} = \dot{\psi} - \frac{1}{m*v}\left[c_v * \left(\delta_v + \beta - \frac{l_v*\dot{\psi}}{v}\right)\cos(\delta_v+\beta) + c_h * \left(\delta_h + \beta + \frac{l_h*\dot{\psi}}{v}\right)\cos(\delta_h+\beta)\right] \quad (5.27)$$

Da der vordere und hintere Lenkwinkel vom Betrag her null sind und der Drehpunkt beim Auto der Carrerabahn nicht im Schwerpunkt liegt ergibt sich die DGL zu:

$$\dot{\beta} = \dot{\psi} - \frac{1}{m*v}\left[c_h * \left(\beta + \frac{l*\dot{\psi}}{v}\right)\cos(\beta)\right] \quad (5.28)$$

Der Drallsatz aus (5.4) wird mit Betrachtung der Lenkwinkel zu:

$$J_z * \ddot{\psi} = F_{SV} * l_v * \cos\delta_v - F_{SH} * l_h * \cos\delta_h \quad (5.29)$$

Auch hier werden wieder die Beziehungen aus (5.8) und (5.9) eingesetzt und es ergibt sich:

$$\ddot{\psi} = \frac{1}{J_z}\left[c_v * \left(\delta_v + \beta - \frac{l_v * \dot{\psi}}{v}\right)\cos\delta_v - c_h * \left(\delta_h + \beta + \frac{l_h * \dot{\psi}}{v}\right)\cos\delta_h\right] \tag{5.30}$$

Auch hier werden die beiden Lenkwinkel δ_v und δ_h gleich null gesetzt, der Drehpunkt in die Vorderachse gelegt und es ergibt sich nun die DGL:

$$\ddot{\psi} = \frac{1}{J_z}\left[c_h * \left(\beta + \frac{l * \dot{\psi}}{v}\right)\right] \tag{5.31}$$

Auch beim nichtlinearen Einspurmodell kam ich zu dem Schluss dass es für die Simulation ungeeignet ist. Im Prinzip stellt es nur eine Erweiterung des linearen Einspurmodells dar, bei dem die hinteren Räder lenkbar sind. Ansonsten gelten dieselben Einschränkungen für dieses Modell wie für das lineare Einspurmodell.

5.6 Nichtlineares Zweispurmodell[5]

Ein genaueres Modell um das Fahrverhalten eines Autos zu simulieren ist das nichtlineare Zweispurmodell. Dieses Modell schließt die Kipp-und Wankdynamik eines Fahrzeugs mit in die Betrachtung der kinematischen und dynamischen Vorgänge ein. Dies ist für die Simulation zum jetzigen Zeitpunkt nicht von Bedeutung, könnte aber in späteren Szenarien, z.B. mit LKW – Modellen, durchaus eine Rolle spielen und soll deshalb trotzdem hier kurz vorgestellt werden. Dieses Modell besitzt gegenüber den Einspurmodellen zwei Freiheitsgrade. Es basiert auf den Newton - Euler'schen Gleichungen.

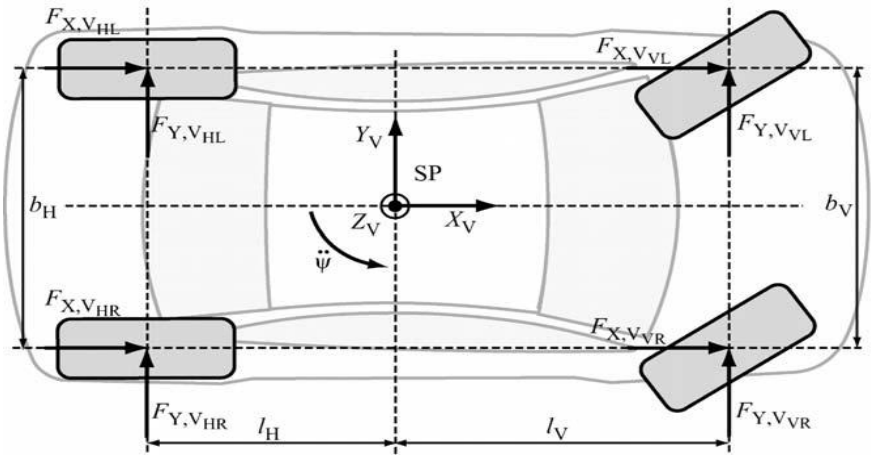

Abbildung 10: Nichtlineares Zweispurmodell (Horizontaldynamik) **Quelle [Ise06]**

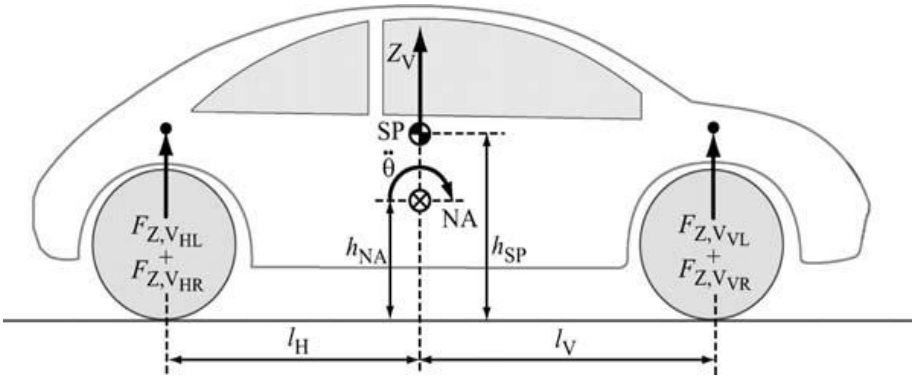

Abbildung 11: Nichtlineares Zweispurmodell (Nickdynamik) **Quelle [Ise06]**

[5] Vgl. [Ise06]

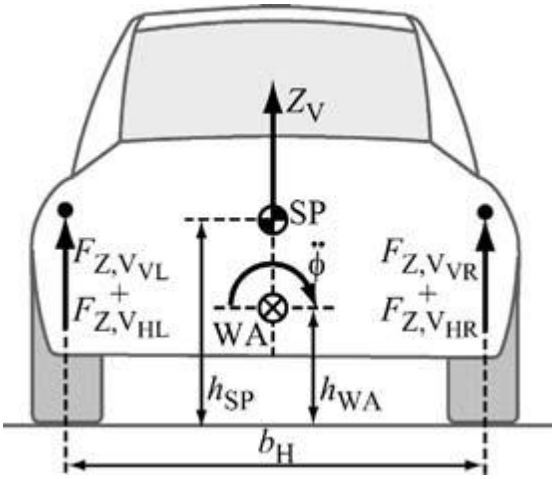

Abbildung 12: Nichtlineares Zweispurmodell (Wankdynamik) Quelle [Ise06]

Mit Hilfe des Impulssatzes nach Newton können die Kräftegleichungen entlang der drei Bewegungsrichtungen x, y und z aufgestellt werden. Die drei Gleichungen lauten wie folgt:

$$F_X = m * \dot{v}_X + \left(\dot{\Theta} * v_Z + \dot{\psi} * v_Y\right) \tag{5.32}$$

$$F_Y = m * \dot{v}_Y + \left(\dot{\psi} * v_X + \dot{\phi} * v_Z\right) \tag{5.33}$$

$$F_Z = m * \dot{v}_Z + \left(\dot{\phi} * v_Y + \dot{\Theta} * v_X\right) \tag{5.34}$$

Die Formeln zur Berechnung der resultierenden Kräfte in Längs- und Querrichtung und der Fahrzeughochachse lauten folgendermaßen:

$$F_X = F_{XVL} + F_{XVR} + F_{XHL} + F_{XHR} - m * g * \sin\alpha_X - F_{XLW} \tag{5.35}$$

$$F_Y = F_{YVL} + F_{YVR} + F_{YHL} + F_{YHR} - m * g * \sin\alpha_Y - F_{YLW} \tag{5.36}$$

$$F_Z = F_{ZVL} + F_{ZVR} + F_{ZHL} + F_{ZHR} - m * g + F_{ZAUF} \tag{5.37}$$

Hierbei sind die α_X und α_Y die Neigung der Fahrbahn in Quer- und Längsrichtung. Die Kräfte F_{XLW} und F_{YLW} stellen die Luftkräfte in Quer- und Längsrichtung dar, F_{ZAUF} die Kraft die durch den Auftrieb des Autos erzeugt wird. Zur Berechnung der Kräfte sei an dieser Stelle auf [Ise06] verwiesen.

Mit Hilfe der Euler'schen Kreisgleichung werden nun noch die drei möglichen Drehbewegungen um die Längs-, Quer- und Hochachse bestimmt.

Dies führt zu den Folgenden drei Gleichungen:

$$J_X * \ddot{\phi} = M_X + \dot{\Theta} * \dot{\psi} * (J_Y - J_Z) \tag{5.38}$$

$$J_Y * \ddot{\Theta} = M_Y + \dot{\psi} * \dot{\phi} * (J_Z - J_X) \tag{5.39}$$

$$J_Z * \ddot{\psi} = M_Z + \dot{\phi} * \dot{\Theta} * (J_X - J_Y) \tag{5.40}$$

Durch bremsen und beschleunigen wird eine Nickbewegung an der Karosserie eingeleitet. Diese wird durch die im Schwerpunkt des Fahrzeugs angreifende Trägheitskraft ausgelöst. Das Durchfahren einer Kurve bewirkt eine Wankbewegung des Autos durch die im Schwerpunkt angreifende Zentrifugalkraft. Die Wank- und Nickkräfte sind durch die Folgenden zwei Gleichungen gegeben:

$$F_{TX} = -F_X = -m * a_X \tag{5.41}$$

$$F_{TY} = -F_Y = -m * a_Y \tag{5.42}$$

Mit der Annahme das der Nick- und Wankpol senkrecht unter dem Schwerpunkt des Autos liegen, lassen sich die Gleichungen für die resultierenden Momente folgendermaßen angeben:

$$M_X = (F_{ZVL} - F_{ZVR}) * \frac{b_V}{2} + (F_{ZHL} - F_{ZHR}) * \frac{b_H}{2} - (h_{SP} - h_{WA}) * F_{TY} \tag{5.43}$$

$$M_Y = (F_{ZHL} + F_{ZHR}) * l_H - (F_{ZVL} + F_{ZVR}) * l_V - (h_{SP} - h_{NA}) * F_{TX} + \ldots$$
$$\ldots + (l_H * F_{AH} - l_V * F_{AV}) \quad (5.44)$$

$$M_Z = (F_{XVR} - F_{XVL}) * \frac{b_V}{2} + (F_{XHR} - F_{XHL}) * \frac{b_H}{2} + (F_{YVL} + F_{YVR}) * l_V - \ldots$$
$$\ldots - (F_{YHL} + F_{YHR}) * l_H \quad (5.45)$$

Durch einsetzen von (5.43) in (5.38), (5.44) in (5.39) und (5.45) in (5.40) können die Winkelbeschleunigungen um die X -, Y - und Z - Achse des Fahrzeugs berechnet werden. Durch Integration der Gleichungen (5.38), (5.39) und (5.40) können die jeweiligen Winkelgeschwindigkeiten bestimmt werden.

Setzt man (5.35) in (5.32), (5.36) in (5.33) und (5.37) in (5.34) ein so können unter Berücksichtigung der Gleichungen (5.38), (5.39) und (5.40) die Beschleunigungen in X -, Y - und Z - Richtung des Fahrzeugs ermittelt werden. Auch hier kann durch Integration der Gleichungen (5.32), (5.33) und (5.34) die Geschwindigkeit in der jeweiligen Richtung ermittelt werden.

Von einer Implementierung dieses Modells als Lösung für die Bestimmung des Driftwinkels hab ich abgesehen, da es meiner Meinung nach zu viel Overhead mit sich bringt. Für die Simulation insbesondere für die Realisierung des Driftwinkels denke ich reicht eine einfache schnelle Lösung aus, die auch auf Kosten der Genauigkeit gehen kann. Da bei diesem Modell alle möglichen Rotationsbewegungen des Kraftfahrzeugs berücksichtigt werden halte ich diese Modell für zu kompliziert um nur den Driftwinkel eines Autos auf der Carrerabahn zu bestimmen.

5.7 Einfaches Modell für den Schwimmwinkel

Das Modell wird auf der Basis von einfachen Bewegungs- und Kräftegleichungen entwickelt. Das Ziel bei der Simulation ist es im Moment des Kurveneintritts die Zeit zu bestimmen, die das Auto bis zur Auslösung des Sensors benötigt, unter der Berücksichtigung das das Auto zu schnell in die Kurve einfährt und auszubrechen beginnt. Für die Simulation ist nur die Dynamik in Querrichtung und Längsrichtung von Bedeutung. Kipp-, Nick- oder Lastwechselvorgänge wie sie in der Realität bei diesem Vorgang auftreten können vernachlässigt werden. Da die Autos der Carrerabahn durch eine Schiene geführt werden, wird beim überschreiten des Grenzbereichs nur das Übersteuern betrachtet, da sich ein Drehmoment um die Schienenführung aufbaut. Dieses Drehmoment veranlasst das Auto über das Heck zum Kurvenäußeren auszubrechen. Die Beschleunigung die für diesen Vorgang notwendig ist wird durch die Zentrifugalkraft erzeugt.

Die Seitenführungskräfte der Reifen werden in diesem vereinfachten Modell durch die Gleitreibungskraft zwischen Reifen und Fahrbahn ersetzt. Diese Vereinfachung der

Seitenführungskräfte wird damit begründet, dass die Gleichungen für die Reifenführungskräfte für echte Fahrzeugreifen gedacht sind, die bekanntlich einen komplizierten Aufbau aus unterschiedlichsten Materialien und Profilen haben. Ein echter Reifen ist mit Luft gefüllt wo durch es zu komplexen physikalischen Vorgängen zwischen Reifen und Fahrbahnoberfläche kommt. Die Reifen an einem Auto der Carrerabahn sind Vollgummireifen, weshalb für dieses Modell nur die geeigneten Reibungskoeffizienten für Haft-und Gleitreibung gefunden werden müssen. Die Koeffizienten können mittels Versuche an einer Steigung der Bahn empirisch Ermittelt werden. Dazu wird die Steigung der Bahn solange erhöht bis das Auto, das quer zur Fahrbahn steht, von alleine aus dem Stillstand zu rutschen beginnt. Man bestimmt den Winkel der Bahnsteigung und ist somit in der Lage den Haftreibungskoeffizienten für die Reifen des entsprechenden Fahrzeugs zu bestimmen. Der Haftreibungskoeffizient ist gerade $\tan \alpha$.

Um den Rollreibungskoeffizienten zu ermitteln wird der Vorgang der Bergabfahrt betrachtet. Man kann die Zeit und die Geschwindigkeit die ein Auto zum herunterfahren mit konstanter Geschwindigkeit und ohne Reibungseinflüsse benötigen würde berechnen. Diese Rechnung vergleicht man mit dem Vorgang in der Realität. Durch Anbringung von zwei Lichtschranken am Berg und im Tal kann man die Zeit und die Geschwindigkeit messen. Durch Anwendung des Geschwindigkeits-Zeit-Gesetzt für die beschleunigte Bewegung an der schiefen Ebene kann unter der Verwendung der gemessenen Werte für Endgeschwindigkeit und Zeit der Wert für den Gleitreibungskoeffizienten ungefähr berechnet werden

$$F = F_H - F_K \tag{5.46}$$

$$a = g(\sin \alpha - \mu * \cos \alpha) \tag{5.47}$$

$$v_{(t)} = v_0 + a * t \tag{5.48}$$

(5.47) eingesetzt in (5.48) und nach μ_R umgestellt ergibt dann:

$$\mu_R = \frac{v_0 - v + g * \sin \alpha * t}{g * \cos \alpha * t} \tag{5.49}$$

Um den Gleitreibungskoeffizienten zu bestimmen muss auch hier das Auto in der Bewegung betrachtet werden. Die beste Methode um den Koeffizienten zu bestimmen ist eine lange gerade, wie bei einer Steilkurve, schräg aufzubauen. Nun lässt man ein Auto langsam auf dieser geraden fahren. Man erhöht nun den Winkel der Fahrbahn solange bis das Heck des

Autos bei der langsamen Fahrt wegrutscht. Auch hier ergibt sich der Reibungskoeffizient wieder zu $\tan \alpha$.

Der gesamte Vorgang beim Driften besteht aus zwei Kreisbewegungen die sich gegenseitig beeinflussen. Die Zentrifugalkraft, die durch die erste Kreisbewegung erzeugt wird, hängt von der Geschwindigkeit des Autos bei der Kurvenfahrt ab. Durch das Ausbrechen des Hecks wird die zweite Kreisbewegung um den Drehpunkt in der Vorderachse eingeleitet und es entsteht nach Überwindung der Haftreibung des Reifens eine Gleitreibungskraft. Diese Gleitreibungskraft sorgt dafür, dass das Auto bei der Kurvenfahrt langsamer wird und die Zentrifugalkraft abnimmt. Diese Abnahme der Zentrifugalkraft beeinflusst die Geschwindigkeit mit dem der Schwimmwinkel beim Driftvorgang zu nimmt.

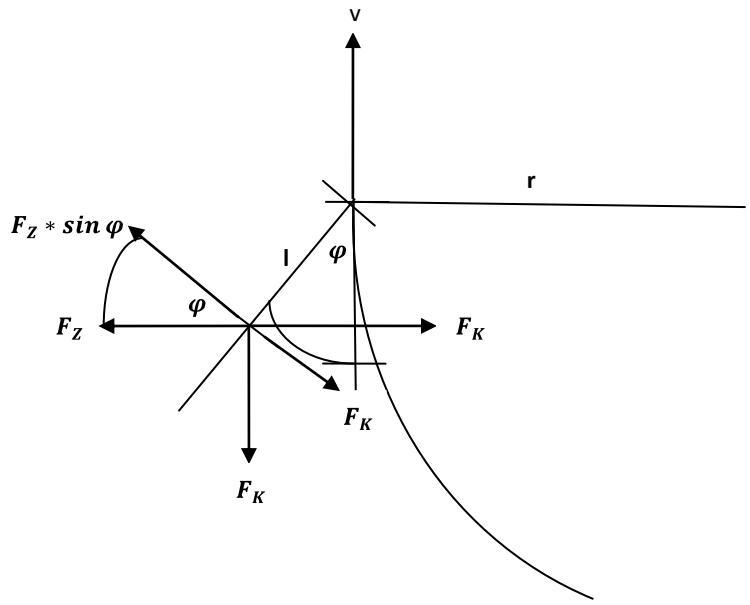

Abbildung 13: Kräfte bei der Kurvenfahrt mit Übersteuern

Bei diesem Modell wird angenommen, dass das Auto beim überschreiten des Grenzbereich in der Kurve eine gleichförmig verzögerte Bewegung beschreibt. Diese Bewegung ist durch die folgende Gleichung gegeben.

Ansatz mit Gleichgewichtsbedingung für stationäre Kreisfahrt:

$$F_Z \leq F_R \tag{5.50}$$

Resultierende Beschleunigung in lateraler Richtung:

$$F = F_Z - F_K \tag{5.51}$$

$$m * a_y = m * \frac{v^2}{r} - m * g * \mu \tag{5.52}$$

$$a_y(t) = \frac{v^2(t)}{r} - g * \mu \tag{5.53}$$

Mit der Verzögerung die durch die Reibung der zweiten Kreisbewegung auftritt, ergibt sich die Geschwindigkeit zu:

$$v^2(t) = (v_0 - g * \mu * t)^2 \tag{5.54}$$

(5.54) eingesetzt in (5.53) ergibt:

$$a_y(t) = \frac{(v_0 - g*\mu*t)^2}{r} - g * \mu \tag{5.55}$$

Die Zeit die das Auto bis zum Erreichen des Drift-Sensors benötigt lässt sich mit dem Weg-Zeit-Gesetzt der gleichförmig verzögerten Bewegung ausrechen:

$$s_{(t)} = v_0 * t - \frac{1}{2} a * t^2 \tag{5.56}$$

(5.56) nach t umgestellt ergibt sich folgende quadratische Gleichung:

$$t_{1/2} = \frac{v_0}{\mu*g} \pm \sqrt{\left(\left(\frac{v_0}{\mu*g}\right)^2 - \frac{2s}{\mu*g}\right)} \tag{5.57}$$

Die Geschwindigkeit am Sensor ergibt sich aus dem Geschwindigkeits-Zeit-Gesetzt der gleichförmig verzögerten Bewegung:

$$v_{(t)} = v_0 - \mu * g * t \tag{5.58}$$

Für die gleichförmig beschleunigte Kreisbewegung gelten Folgende Beziehungen:

$$\omega_{(t)} = \alpha * t = \frac{d\varphi}{dt} \tag{5.59}$$

$$\varphi_{(t)} = \int_0^t \omega dt = \int_0^t \alpha * t = \frac{1}{2}\alpha * t^2 \tag{5.60}$$

Dies führt nun zu Folgender Beziehung:

$$\varphi_{(t)} = \frac{1}{2}\Big(\big(a_y(t) * \sin\varphi\big) - \mu * g\Big) * t^2 \tag{5.61}$$

Gleichung (5.55) in (5.61) eingesetzt ergibt:

$$\varphi_{(t)} = \frac{1}{2}\left(\left(\left(\frac{(v_0 - g*\mu*t)^2}{r} - g * \mu\right) * \sin\varphi\right) - \mu * g\right) * t^2 \tag{5.62}$$

Dieser Ansatz führte leider nicht zum Ziel, da der zu berechnende Winkel auch in der rechten Seite der Gleichung auftaucht und nicht zu eliminieren war. Aus diesem Grund wurde dieser Ansatz nach längerer Überlegung nicht weiter verfolgt. Lediglich die Ansätze zur Bestimmung der Reibungskoeffizienten erweisen sich als nützlich. Mit ihnen kann man eine Vereinfachung für das lineare Einspurmodell treffen in dem man die Schräglaufsteifigkeit durch die Gleitreibungskraft ersetzt.

5.8 Winkelbestimmung durch neuronales Netz

Eine weitere Möglichkeit zur Implementierung des Driftsensors wäre das entwerfen und trainieren eines neuronalen Netzes. Da die Carrerabahn während der Erstellung der

Diplomarbeit noch nicht komplett fertig aufgebaut war konnte dieser Ansatz nicht durchgeführt werden. Die Driftsensoren sind noch nicht in der Bahn eingebaut und deshalb war es nicht möglich Trainingsdaten für ein neuronales Netz zu gewinnen. Ich möchte hier aber trotzdem ein paar Überlegungen zu diesem Ansatz und zu einer möglichen Architektur des neuronalen Netzes anstellen. Ich würde für dieses Problem ein vorwärtsgerichtetes Backpropagation-Netz wählen. Diese Art von neuronalen Netzen bestehen aus einer Eingangsschicht, einer Ausgangsschicht und ein oder mehreren verdeckten Zwischenschichten. Es würde das überwachte Lernen bei diesem neuronalen Netz eingesetzt.

Die Theorie besagt das jede kontinuierliche Abbildung der Form *„y = f(x)“* durch ein Netz mit nur einer verdeckten Schicht gelöst werden kann [Bit07], deshalb würde ich zur Bestimmung des Driftwinkels ein Netz mit genau einer verdeckten Schicht verwenden. Bei den Neuronen in den Zwischenschichten gibt es kein Rezept mit dem man die Anzahl ermitteln könnte. Man kann nur soviel sagen, dass zu wenig Neuronen ein langsames lernen zur Folge hat und zu viel Neuronen das Netz in einen Speicher verwandeln können da es schlechter generalisiert. Die Anzahl sollte durch ausgiebiges testen und experimentieren herausgefunden werden.

Die Anzahl der Neuronen der Eingangsschicht und der Ausgangsschicht sind klar durch das zu lösende Problem vorgegeben. Bei der Ausgangschicht würde ich vierzehn Neuronen verwenden, da der Driftsensor vierzehn Werte annehmen kann. Das Ergebnis wäre dann in der 1:n Darstellung der Ausgangschicht ablesbar. Die Anzahl der Neuronen in der Eingangsschicht wird durch die Anzahl der betrachteten Merkmale, also der Eingangsparameter bestimmt. Als mögliche Eingangsparameter könnte man die Geschwindigkeit, den Radius, das Bogenmaß von Kurveneintritt bis zur Sensorposition, das Fahrzeuggewicht, den Haft- und Gleitreibungskoeffizient, die Distanz von der Finne zur Hinterachse, die Bahnsteigung und die Reifenbreite verwenden. Dies sind dann neun Merkmale für das zu lösende Problem. Daraus ergeben sich neun Neuronen in der Eingangsschicht. Es ist beim Anwenden des fertigen neuronalen Netzes noch zu untersuchen ob eine Normierung der Werte z.B. in einem Intervall zwischen null und eins eine positive Auswirkung auf die Genauigkeit der Ergebnisse hat.

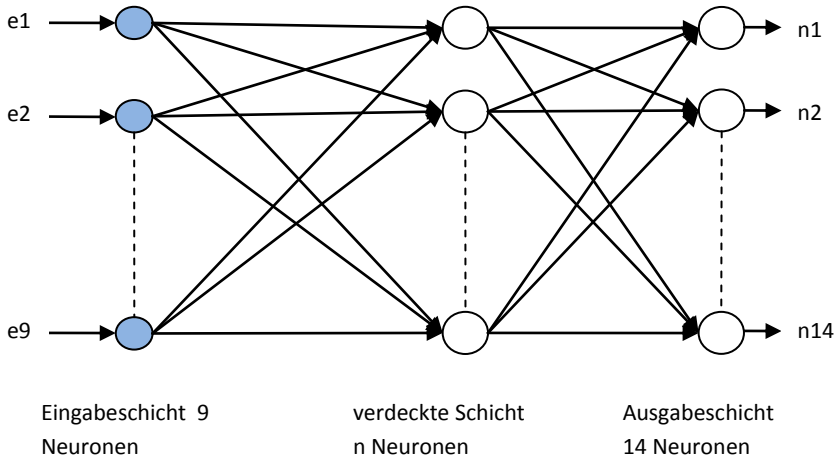

Abbildung 14: Neuronales Netz mit einer versteckten Zwischenschicht

Als Aktivierungsfunktion in der verdeckten Schicht wird eine binär, sigmoide Funktion benötigt. Eine solche Funktion hat eine s-förmige Darstellung. Sie stellt eine reelle und differenzierbare Funktion innerhalb des Intervalls [0, 1] dar. Diese Darstellung wird benötigt, da in der Ausgabeschicht eine 1:n Darstellung verwendet wird und die Differenzierbarkeit von einem Backpropagation Netz vorausgesetzt wird. In MATLAB wird einem hierfür nur die logistische Funktion zur Verfügung gestellt. Diese würde dann im neuronalen Netz zum Einsatz kommen. Diese Funktion wird durch folgende Gleichung beschrieben:

$$P_{(t)} = \frac{1}{1+e^{-t}} \tag{5.63}$$

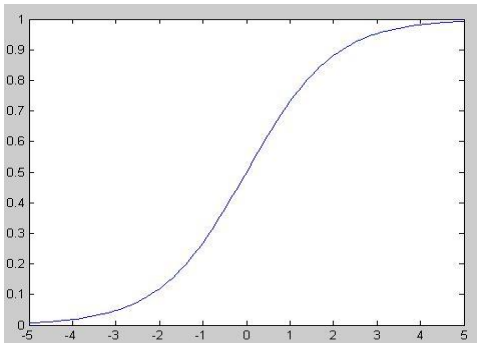

Abbildung 15: Funktion logsig in MATLAB

Als Trainingsalgorithmus könnte man den Levenberg-Marquard-Algorithmus oder den Resilient-Backpropagation Algorithmus verwenden. Der Levenberg-Marquard-Algorithmus

ist schneller als die meisten anderen Algorithmen die zum trainieren von neuronalen Netzen verwendet werden. Er kann aber zu Problemen führen, da er einen hohen Speicherbedarf hat wenn mit großen Lernmengen gearbeitet wird. Als Netzperformancefunktion würde ich den „mean-square-error“, also den mittleren quadratischen Fehler verwenden. Die Einstellungen für maximale Epochenzahl und für das Fehlerziel muss wieder durch testen und experimentieren herausgefunden werden. Der Wert für das Fehlerziel darf nicht zu groß und nicht zu klein gewählt werden, da das Netz sonst untrainiert bleibt oder nicht innerhalb der eingestellten Epochenzahl zum Ergebnis führt. Des Weiteren wäre noch zu Untersuchen in wie fern sich eine Vorinitialisierung des neuronalen Netzes auf die Performance und die Qualität des Ergebnisses auswirkt [Bit07].

Es ist auch zu Untersuchen wie sich die Aufbereitung der Daten in Trainings- und Testdaten auf die Lernfähigkeit und die Ergebnisse des neuronalen Netzes auswirken. Die Daten die man durch Versuche auf der Carrerabahn ermitteln muss sind in 14 Cluster aufteilbar. Jeder Driftsensorwert zwischen eins und vierzehn bildet ein Cluster. Man könnte hier zwei Strategien verfolgen. Entweder teilt man die Daten prozentual oder in immer gleichen Teilen auf.

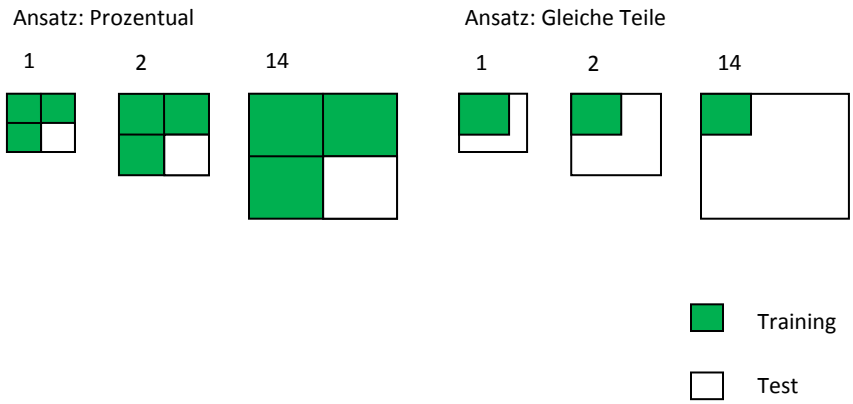

Abbildung 16: Aufteilung der Daten

Ich denke dass ein neuronales Netz eine gute Möglichkeit wäre um den Driftwinkel eines Autos zu bestimmen. Da es mir aber Aufgrund der Tatsache, dass die Driftsensoren noch nicht auf der Carrerabahn angebracht waren, nicht möglich war Test-und Trainingsdaten zu ermitteln, wurde dieser Ansatz verworfen.

5.9 Driftsensor als Wertetabelle

Für die Simulation habe ich mich für ein einfaches Modell entschieden um das Fahrzeug und damit auch den Driftsensor zu realisieren. Jedes Fahrzeug, das auf der Carrerabahn benutzt wird ist als XML-Datei hinterlegt. In dieser Datei sind die möglichen Geschwindigkeitswerte die das Fahrzeug annehmen kann hinterlegt. In der Datei gibt es eine „Speedmap“, welche alle von dem Fahrzeug realisierbaren Geschwindigkeiten enthält. Die Werte sind durch Messungen mit den Fahrzeugen auf der realen Carrerabahn zu ermitteln. Ein

Geschwindigkeitseintrag beinhaltet zwei Werte und eine weitere Struktur für den Schleudersensor. Der erste Wert ist ein hexadezimaler Zahlenwert der zweite Wert entspricht der Geschwindigkeit in mm/s. Die Schleudersensorstruktur enthält wiederum je eine Struktur pro Fahrzeugspur.

In den Strukturen für die Fahrzeugspuren sind die Werte des Schleudersensors, für die aktuelle Geschwindigkeit, für die sieben möglichen Kurven der Carrerabahn hinterlegt. Der Sensor kann Werte zwischen 1 und 14 annehmen. Die jeweiligen Werte sind wie die Geschwindigkeitswerte auf der realen Bahn für das Fahrzeug durch Messungen und Versuche zu ermitteln. Wenn der Schleudersensor einen Wert von -1 eingetragen hat, so bedeutet dies, dass das Auto bei der betreffenden Geschwindigkeit, auf seiner Spur in der jeweiligen Kurve heraus geflogen ist, da seine Geschwindigkeit für diese Kurve zu schnell war.

Ich denke dass dieser Ansatz effizient ist, da auf viele Rechnungen verzichtet, was sich wiederum positiv auf das Laufzeitverhaltender Simulation auswirkt. Das nachschauen in einer Tabelle ist um ein vielfaches schneller als die Integration von zwei oder mehreren Differentialgleichungs- Systemen. Ich denke auch die damit erreichbare Genauigkeit der Simulation ist ausreichend, da ja nicht die exakte physikalische Simulation der Carrerabahn als Ziel im Vordergrund steht, sondern eine möglichst genaue. Ich denke das dieser Ansatz ausreichend ist um die Simulation in ihrem Einsatzgebiet, dem Testen von Programmen die Autos auf der Carrerabahn steuern, erfolgreich einzusetzen.

5.10 Vereinfachtes physikalisches Modell

Dieses Modell basiert im Wesentlichen auf dem in Kapitel 5.7 vorgestellten Modell. Als Vereinfachung wird bei diesem Modell aber eine konstante Reibungskraft angenommen. Des Weiteren wird angenommen, dass die Zentrifugalkraft auch konstant ist. Es wird wieder davon ausgegangen, dass das Auto zu schleudern beginnt, wenn die Zentrifugalkraft größer als die Haftreibungskraft der Reifen ist. Die daraus resultierende Kraft versetzt das Auto in eine Drehbewegung. Die resultierende Kraft ergibt sich aus Zentrifugalkraft minus Haftreibungskraft. Daraus lässt sich die Beschleunigung bestimmen mit der der Schleudervorgang eingeleitet wird. Diese Beschleunigung tritt auch in der Gleichung für die Winkelgeschwindigkeit auf und durch gleichsetzen mit anschließenden Einsetzen hat man eine Formel, die nach dem Winkel umgestellt werden kann. Nun muss noch die Zeit bestimmt werden, welche zwischen Kurveneintritt bis zum passieren des Schleudersensors vergeht. Diese wird dann in die Formel eingesetzt.

$$m * a = F_Z - F_R \tag{5.64}$$

$$a = \frac{v^2}{r} - g * \mu_H = \omega^2 * r \tag{5.65}$$

$$\omega = \sqrt{\frac{a}{r}} = \frac{\Delta\varphi}{\Delta t} \tag{5.66}$$

$$\Delta\varphi = \sqrt{\frac{\frac{v^2}{r} - g*\mu_H}{r}} * \Delta t \tag{5.67}$$

$$s = v_0 * t - \frac{1}{2} * a * t \tag{5.68}$$

$$t_1/t_2 = \frac{v_0}{a} \mp \sqrt{\left(\frac{v_0}{a}\right)^2 - \frac{2s}{a}} \tag{5.69}$$

Kapitel 6

6. Simulation

6.1 Betriebssystem

Die Simulation wurde unter dem Betriebssystem Linux entwickelt. Die Wahl dieses Betriebssystems begründet sich durch den Vorschlag von Prof. Dr. Michael Mächtel, die Simulation auf einer schon vorhandenen, ähnlichen Simulation aufzubauen. Gerade die Tatsache, dass der Treiber den mir Prof. Mächtel zur Verfügung gestellt hat, für das Betriebssystem Linux geschrieben wurde, schränkte die Auswahl eines anderen Betriebssystems enorm ein. Da Linux eine offene API zur Treiberentwicklung bietet und der Treiber von Prof. Mächtel für das Betriebssystem Linux geschrieben wurde, wurde Linux verwendet.
Zum Einsatz kam die Distribution Ubuntu 7.10, später 8.04 als 32 Bit Installation. Als Betriebssystem- Kernel wurde Version 2.6.22-14 verwendet. Ubuntu ist eine freie Linux-Distribution und basiert auf der Distribution Debian. Das Projekt Ubuntu wird von der Firma Canonical Ltd. unterstützt und finanziert. Ziel dieser Distribution ist es ein einfach zu installierendes und zu wartendes Betriebssystem auf Linux-Basis bereitzustellen, zu garantieren das die benutzten Pakete und Bibliotheken auf einander abgestimmt sind und Benutzern die das arbeiten mit Windows gewöhnt sind einen schnellen Einstieg zu gewähren. Es gibt verschiedenen Versionen der Distribution mit die auf unterschiedliche Desktops aufsetzen. Die von mir gewählte Version benutzt als Desktop KDE und wird deshalb Kubuntu genannt. Diese Wahl hat aber keinen Einfluss auf die Qualität der Arbeit sondern ist eine persönliche Vorliebe meinerseits. Genauso gut hätte man Ubuntu, welches auf Gnome aufsetzt, oder Xubuntu welches auf Xfce aufsetzt zur Entwicklung benutzen können.

6.2 Programmiersprache und Entwicklungsumgebung

Als Programmiersprache wurde Ansi C eingesetzt. Die Wahl hierauf viel auch auf Grund der Tatsache, dass die Simulation und der Treiber, welcher mir von Prof. Mächtel zur Verfügung gestellt wurde in dieser Sprache geschrieben wurde. Eine andere Möglichkeit wäre eine Neuentwicklung bzw. Teilportierung in der Sprache C++ gewesen. Diese Alternative wurde aber wegen unnötigem Mehraufwand von mir verworfen. Bei der Versionsverwaltung wurde auf die Open-Source Software Subversion SVN gesetzt, da es

über einige Vorteile gegenüber der Alternative CVS verfügt. So ist SVN wesentlich schneller als CVS und Subversion versioniert auch Verzeichnisse, Umbenennungen und Kopien von Dateien. Mit SVN ist es möglich mit binären Dateien umzugehen und die einzelnen Commits sind atomar, d.h. ein Schreibzugriff wird erst dann wirksam wenn der gesamte Vorgang abgeschlossen ist.
Bei der Wahl einer geeigneten Entwicklungsumgebung standen mehrere Alternativen zur Diskussion. Allen sollte gemein sein, dass es frei verfügbare, sogenannte Open-Source-Software sein sollte. Die erste Möglichkeit wäre die Verwendung von einem normalen Texteditor mit Hilfe der Kommandozeile, in Verbindung mit den Werkzeugen gcc und gdb, unter Kubuntu. Diese Art der Entwicklung wurde aber schon im Vorfeld von mir abgelehnt, da ich wenn möglich eine Entwicklungsumgebung zum erstellen des Codes benutzen wollte. Der Vorteil einer Entwicklungsumgebung gegenüber der ersten Möglichkeit liegt klar in der besseren Bedienbarkeit und Effizienz beim erstellen von Code. Dies fängt mit der Hilfe bei der Codeformatierung an und geht bis hin zur Codevervollständigung.

Eine Entwicklungsumgebung die es für Kubuntu gibt und die auch einfach über den Packetmanager Adept unter Kubuntu installiert werden kann heißt KDevelop. Mit dieser IDE ist es möglich Projekte mit C/C++ und vielen anderen Programmiersprachen anzulegen. Die Entwicklungsumgebung verfügt über eine grafische Debugger-Schnittstelle die den gdb verwendet. Über Plugin-Werkzeuge ist es jederzeit möglich den Funktionsumfang der IDE zu erweitern. Um grafische Benutzeroberflächen zu erstellen verfügt KDevelop über einen Designer der das Trolltech Framework Qt benutzt. Desweiteren verfügt die IDE über einen Automake Manager, der die Verwendung der GNU Entwicklungstools zum erstellen von Software, vereinfacht. Mit ihm wird die Erstellung von Makefiles und configure – Skripts einfacher.

Meine Wahl fiel dann aber auf die Plattform Eclipse. Da Eclipse heute schon in vielen kommerziellen Softwareentwicklungen eingesetzt wir und von einer Reihe großer Namhafter Firmen unterstützt wird, viel die Wahl auf diese IDE recht schnell. Eclipse ist eine in Java geschriebene freie Entwicklungsumgebung. Daraus ergibt sich ein weiterer nicht zu unterschätzender Vorteil von Eclipse gegenüber anderen Entwicklungsumgebungen, die Plattformunabhängigkeit. Alles was zum arbeiten mit dieser IDE benötigt wird ist eine Betriebssystem auf dem eine Java Runtime läuft. Der Funktionsumfang kann auch bei dieser Entwicklungsumgebung mit Hilfe von Plugin-Werkzeugen erweitert werden. Unter Kubuntu Eclipse wie die meiste Software auch über den Adept Paketmanager heruntergeladen und installiert werden. In dieser Diplomarbeit kam die Version 3.3.1.1, später 3.4 zum Einsatz. In der Grundinstallation verfügt Eclipse nur über die notwendigen Plugins um ein Java Projekt zu erstellen. Für die Verwendung als Entwicklungsumgebung um die Simulation zu erstellen, mussten erst noch die notwendigen Plugins installiert werden. Hier kam schon ein großer Vorteil von Eclipse gegenüber anderen IDE zum tragen. Eclipse kann aus der DIE heraus auf dem Neusten Stand gehalten und um neue Funktionalität erweitert werden. Als erste Erweiterungen wurden das CDT Plugin in der Version 4.0.3.2 installiert. Mit diesem Plugin ist es möglich C/C++ Projekte unter Eclipse zu erstellen. Es benutzt die Standard GNU Entwicklungstools gcc, gdb und make, zum linken, kompilieren und debuggen von C/C++ Programmen und Bibliotheken. Auch in diese IDE bietet eine grafische Schnittstelle zum komfortablen

debuggen mit dem gdb. Um aus der Entwicklungsumgebung direkt mit der Versionsverwaltung arbeiten zu können, wurde das Plugin Subclipse in der Version 1.2.4 installiert. Subclipse ist ein Plugin das auf der Versionsverwaltungssoftware Subversion SVN aufsetzt und dadurch die gesamte Funktionalität von SVN zur Verfügung stellt. In einem extra Ansichtsfenster, unter Eclipse heißen diese Fenster Perspektive, können bekannte Repositories durch die Eingabe von URL und Benutzerdaten ein gepflegt werden. In der SVN-Perspektive kann das Repository dann in einer Baumansicht betrachtet werden. In der Perspektive für die C/C++ Entwicklung können die Änderungen im Projektbaum an das Repository übertragen werden. Mit Eclipse hat man die Wahl ob man sich die Makefiles zum erstellen der ausführbaren Datei selbst erstellen will oder ob das Eclipse automatisch übernimmt. Es bedarf am Anfang einer gewissen Einarbeitungszeit bis man sich mit allen Funktionen und Einstellungen der IDE vertraut gemacht hat aber wenn diese erste Hürde geschafft ist, so ist das Arbeiten und der Funktionsumfang mit bzw. von Eclipse eine Erleichterung. Lediglich die Geschwindigkeit beim Starten der IDE könnte man als einen Kritikpunkt anbringen, da alle Plugins beim Start geladen werden. Man sollte also darauf achten den Funktionsumfang der IDE dem entsprechenden Projekt anzupassen und lieber für unterschiedliche Projekte mehrere angepasste Installationen zu benutzen, als eine Eclipse Installation mit allen verfügbaren Erweiterungen.

6.3 Verwendete Bibliotheken

Um mit dem Betriebssystem Kubuntu Software zu entwickeln, sollte sichergestellt sein, dass die Standard GNU Entwicklungswerkzeuge installiert sind. Die Standard C-Bibliothek wird automatisch bei der Installation des Betriebssystems mit installiert, die Entwicklungswerkzeuge jedoch nicht. Man kann die Werkzeuge jedoch wie bei den Entwicklungsumgebungen einfach über den Adept-Packetmanager nachinstallieren. Dazu muss nur das Paket „build-essentials“ installiert werden. Mit diesem Paket werden alle wichtigen Werkzeuge zum erstellen von C/C++ Programmen installiert. Zum debuggen sollte man dann noch das Paket gdb über den Adept Paketmanager installieren.

Da alle Daten für die Autos und die Strecke in XML-Dateien gespeichert sind, benötigt man noch eine Bibliothek, die den Umgang mit diesem Standard erleichtert. Hierfür benötigt man einen XML-Parser. Ein frei erhältlicher XML-Parser ist der Expat-Parser. Er ist in C geschrieben und ist unter der GPL Lizenz veröffentlicht .Um den XML-Parser zu benützen müssen die Pakete „libexpat1“ und „libexpat1-dev“ auf dem Entwicklungsrechner installiert werden. Dies kann auch wieder einfach über den Adept Paketmanager vorgenommen werden. Das Paket „libexpat1“ muss auch auf dem Rechner installiert sein auf dem die Simulation letztendlich läuft. Handelt es sich dabei um eine 64 Bit Installation des Betriebssystems Linux, so muss das Paket „lib64expat1“ installiert werden. Um den Umgang mit den XML-Dateien noch weiter zu erleichtern, wurde eine weitere Bibliothek verwendet. Die Bibliothek heißt „scew“ und stellt nichts anderes als eine einfache in C geschriebene Wrapper-Bibliothek dar. Über simple Schnittstellen kann hiermit ein Zugriff auf die Funktionalitäten des Expat-Parsers erfolgen. Auch diese Software ist frei erhältlich und fällt unter die LGPL Lizenz. Diese Bibliothek ist jedoch nicht mit dem Adept

Paketmanager installierbar. Man kann die Quellen auf der Seite http://www.nongnu.org/scew/ herunterladen. Die Quellen wurden dann einfach kompiliert und auf dem Rechner installiert.

6.4 Ursprüngliche Simulation

Als Grundlage der erstellten Simulation dienten eine Simulation und ein dazu gehörender Treiber, der mir von Prof. Mächtel für diese Diplomarbeit zur Verfügung gestellt wurde. Mit diesem Programm wurde eine zweispurige Carrerabahn simuliert, die in Form einer Acht aufgebaut ist. Im ersten Schritt musst diese Simulation an die Gegebenheiten der Carrerabahn an der HTWG-Konstanz angepasst werden. Die Anpassungen wurden am Treiber sowie auch am Simulationsprogramm mit den dazugehörenden XML-Dateien vorgenommen. Zu den Anpassungen gehörte die Erweiterung auf Vier Spuren im Simulationsprogramm und im Kernel und das Ein pflegen der neuen Streckendaten für die HTWG-Rennbahn. Nach der Anpassung wurde noch eine Alternative zur Realisierung der Driftsensoren, sowie eine GUI implementiert.

6.4.1 Grundzustand und Modell der Simulation von Professor Mächtel

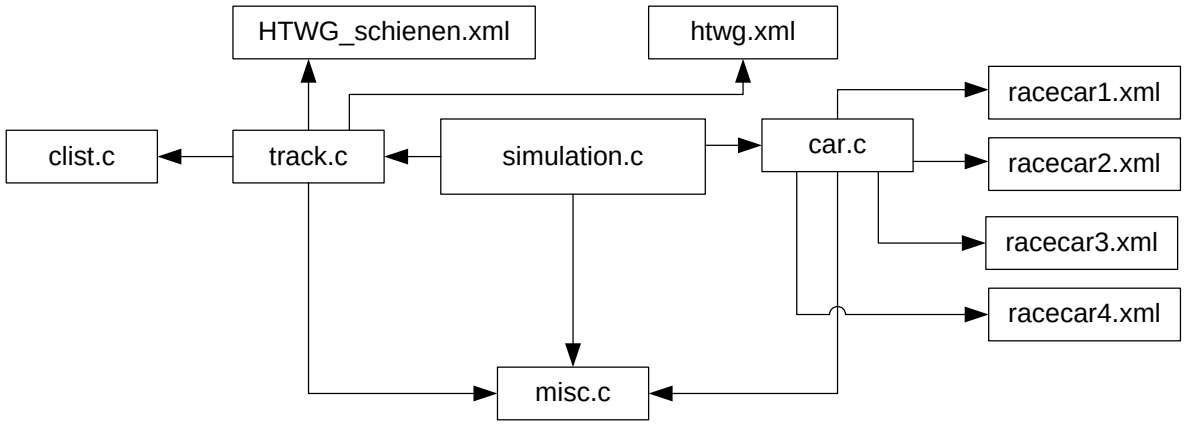

Abbildung 17: Simulationsaufbau

Die Rennwagen der Simulation werden jeweils durch eine eigene Datenstruktur repräsentiert. Die Kenngrößen der Autos werden in einer XML-Datei gespeichert. Die Kenngrößen umfassen die von einem Auto realisierbaren Geschwindigkeiten in mm/s mit einer hexadezimalen Entsprechung, den Schleudersensorwert jeweils für die Innen-und Außenspur der Bahn bei der Geschwindigkeit, die Beschleunigungswerte die von einem Auto bei einem bestimmten Geschwindigkeitsunterschied erreicht werden können in

mm/s² und der hexadezimalen Entsprechung des Geschwindigkeitsunterschiedes so wie die maximale Bremsverzögerung eines Autos in mm/s².
Dateiaufbau Fahrzeug-Datei:

```
<fahrzeug name="xyz">
    <geschwindigkeit>
        <hex>0x20</hex>
        <real>509</real>
        <schleudersensor>
            <innen>-2</innen>
            <aussen>-2/aussen>
        </schleudersensor>
    </geschwindigkeit>
    .
    .
    <beschleunigung>
      <positiv>
          <delta_v>
              <hex>0x10</hex>
                <a>1099</a>
            </delta_v>
            .
            .
      </positiv>
      <negativ>3273</negativ>
    </ beschleunigung >
</fahrzeug>
```

Mit Hilfe dieser Dateien wird nun für jedes Auto, das an der Simulation beteiligt ist, eine Datenstruktur aufgebaut die das Auto in der Simulation darstellt. In der Anfangsversion also maximal zwei Autos die an einem Rennen teilnehmen können.
Bei der Darstellung eines Autos sind vier Strukturen beteiligt. Diese sind alle in einer eigenen Headerdatei definiert.

Die erste Struktur wird benötigt um einen Geschwindigkeitswert aufzunehmen und besteht aus zwei Variablen für Geschwindigkeitswerte für die Innen-und Außenbahn und einer Variablen für die hexadezimale Repräsentation dieser beiden Werte.

Die zweite Struktur wird für die Beschleunigungsvorgänge benutzt und besteht aus vier Variablen. Die erste wird benötigt um eine Geschwindigkeitswert für positive Beschleunigungen aufzunehmen und die zweite für die hexadezimale Entsprechung dieses Wertes aus der Liste der möglichen Geschwindigkeiten aus der Datei. Die zwei verbleibenden Variablen werden genau so benutzt wie die beiden zuvor, ihre Werte werden aber bei Verzögerungen berechnet. Ansonsten nimmt die erste der beiden wieder einen Geschwindigkeitswert auf und die letzte der zwei noch verbleibende einen hexadezimalen Wert der der Geschwindigkeit entspricht.

Die dritte Struktur besteht aus zwei Variablen welche wieder für einen Geschwindigkeitswert und für dessen hexadezimale Entsprechung benutzt werden. Die dritte Struktur wird auch für Werte benötigt die bei Verzögerungen berechnet werden. Auf die genauen Berechnungen um die drei Strukturen mit Werten zu füllen wird weiter unten in diesem Abschnitt noch detaillierter eingegangen. Die vierte Struktur ist die eigentliche Datenstruktur die das Auto in der Simulation darstellt. Sie besteht aus vier Feldern und einer Variablen. Die Variable dient zum Speichern der maximalen negativen Verzögerung und wird aus der XML-Datei gespeichert.

Das erste Feld ist eindimensional und vom Typ der ersten Datenstruktur, welche Geschwindigkeitswerte und deren hexadezimale Entsprechung aufnehmen kann. Durch dieses Feld wird eine „Speedmap“ angelegt, welche alle Geschwindigkeiten die vom Auto angenommen werden können beinhaltet. Die Werte hierfür werden aus der XML-Datei eingelesen. Das Feld hat eine gewisse Ähnlichkeit mit einer Hashtable, da auch hier immer mit Schlüssel-Wertepaaren gearbeitet wird. Der Hexadezimalwert bildet den Schlüssel und Feldindex, die Geschwindigkeit den zugehörigen Wert.

Das zweite Feld in der Datenstruktur ist auch ein eindimensionales in dem die möglichen Beschleunigungen eines Autos aus der entsprechenden XML-Datei gespeichert werden.

Das dritte Feld ist dreidimensional und vom Typ der zweiten Datenstruktur, welche zur Aufnahme von einem Hexadezimalwert und dem zugehörigem Geschwindigkeitswert für jeweils positive und negative Beschleunigungen dient. Die erste Dimension gibt die Beschleunigung an, die zweite Dimension den Weg in 50 mm Schritten und die dritte Dimension die Anfangsgeschwindigkeit.

Das Feldelement **„sVAccelerate[1][2][2]“** gibt demnach also die Geschwindigkeit und den zugehörigen hexadezimalen Wert zurück, für eine Beschleunigung oder eine Verzögerung die dem zweiten Element des zweiten Feldes in dieser Datenstruktur entspricht (**aPositive[1]**) einem dabei zurückgelegten Weg von 2 * 50 mm und einer Anfangsgeschwindigkeit der dem 16 Geschwindigkeitswert in der Speedmap entspricht. Die 16 kommen dadurch zustande, da immer jeder achte Geschwindigkeitswert als mögliche Anfangsgeschwindigkeit betrachtet wird. Mit Hilfe dieses Feldes muss in der Simulation keine Geschwindigkeitsberechnung durchgeführt werden, da einfach auf das entsprechende Element des Feldes zugegriffen wird welches der aktuellen Beschleunigungs-oder Verzögerungssituation entspricht. Die nötigen physikalischen Berechnungen zur Simulationszeit entfallen da sie schon beim Starten der Simulation im Voraus berechnet worden sind.

Das vierte Feld der Datenstruktur hat zwei Dimensionen und ist vom Typ der dritten Struktur welche einen Geschwindigkeitswert und seinen zugehörigen Hexadezimalwert speichert. Die erste Dimension beinhaltet wieder den Weg in 50 mm Schritten, die zweite Dimension die Anfangsgeschwindigkeit. Das Feld wird beim Bremsen mit der vollen Verzögerung benötigt. Die Berechnung der Feldelemente geschieht analog zum oben vorgestellten dreidimensionalen Feld. Hier kann aber auf die Dimension der Beschleunigung verzichtet werden, da sie nicht Variabel ist sondern immer die maximale Bremsverzögerung des Autos aufweist.

Das Feldelement „**sVDecelerate[2][2]**“ gibt z.B. den Geschwindigkeitswert bei einer Vollbremsung, mit einer Anfangsgeschwindigkeit die dem 16 Element der Speedmap entspricht, zurück. Auch diese Werte müssen nicht zur Simulationszeit berechnet werden sondern werden vor dem Start der Simulation ermittelt.

Die beiden Felder stellen aber nur Daten bereit die auf der Annahme beruhen, dass die Beschleunigungs- oder Verzögerungsvorgänge in der Ebene auftreten und keine Steigungen und Gefälle vorliegen. Dies war bei dieser Simulation nicht vorgesehen.

Alle Methoden die bei der Verarbeitung und Berechnung von Daten beteiligt sind die die Darstellung des Autos betreffen sind in einer extra Datei implementiert. Es gibt eine Methode, welche nur dafür Verantwortlich ist, die Daten des Autos aus der XML-Datei in die Variablen und Felder der vierten Datenstruktur zu speichern. Dies geschieht unter der Zuhilfenahme von Funktionen der Bibliothek **libexpat, scew** und der Datei „**misc.c“** welche Funktionalität bereitstellen um XML-Dateien zu parsen. In einer anderen Methode werden die Werte der beiden Strukturen für die Geschwindigkeitsänderungen und das scharfe Abbremsen berechnet. Dies wird mittels der Weg-Zeit-Gesetze der Kinematik für gleichförmig beschleunigte oder verzögerte Bewegungen mit Anfangsgeschwindigkeit durchgeführt. Die Größe der jeweiligen Dimensionen der Felder hängen von den Kennwerten der Autos und des Bahnlayouts ab. Im Fall der Simulation die mir von Professor Mächtel zur Verfügung gestellt wurde, hatte die erste Dimension des dritten Feldes (**sVAccelerate[6][20][12]**) eine maximale Größe von sechs. Dies kommt daher da das Auto auch nur sechs mögliche Beschleunigungen realisieren konnte. Die hat zweite Dimension kann maximal 20 Elemente aufnehmen, da der maximale Beschleunigungsweg beim Layout der dortigen Carrerabahn 1000 mm beträgt und 20 mal 50 gleich 1000 sind. Die dritte Dimension hat eine maximale Größe von zwölf, da nur jeder achte Geschwindigkeitswert der Speedmap benutzt wird. Auf die zwölf kommt man, weil die Speedmap genau 0x80 – 0x20 = 96 Einträge hat und dies durch acht teilt. Die 0x20 kommen daher, dass dies der kleinste Geschwindigkeitswert ist der von einem Auto realisiert werden kann. Auf Grund der gleichen Tatsachen kommen die maximalen Dimensionsgrenzen im vierten Feld (**sVDecelerate[20][12]**) zustande.

6.4.2 Darstellung der Bahn in der Simulation

Die Bahn wird in der Simulation unter Zuhilfenahme von XML-Dateien beschrieben. Das Layout, also die Information darüber welches Carrerabahnsegment nach welchem kommt, wird in einer Datei gespeichert und die Kennwerte der unterschiedlichen Segmente in einer zweiten XML-Datei.

In der Datei mit den Layoutinformationen wird die für jedes Segment der Strecke ein Tag angelegt mit dem Namen des Segments und gegebenenfalls wenn es sich bei dem Segment um eine Kurve handelt noch die Richtung. Wenn das Element einen Schleudersensor enthält wir auch dafür ein Tag angelegt.

Dateiaufbau Layout-Datei:

```
<strecke name="xyz" modellnr="0">
    <selement>
        <name>startziel</name>
    </selement>
    .
    .
    <selement>
        <name>kurve45weit</name>
        <richtung>rechts</richtung>
    </selement>
    .
    .
    <selement>
        <name>kurve90eng</name>
        <richtung>rechts</richtung>
        <ssensor>1</ssensor>
    </selement>
</strecke>
```

In der XML-Datei mit den Segmentinformationen wird für jeden Segmenttyp (Gerade, Kurve, Kreuzung, usw.) ein eigenes Tag definiert. Jedes Segment bekommt eine Nummer und einen Namen als Identifizierung. Die Informationen über die Längen auf der Innen- und Außenbahn werden in zwei zusätzlichen Tags gespeichert. Die Längenangaben sind in mm.

Dateiaufbau Segment-Datei:

```
<schienen>
    <schiene>
        <id>1</id>
        <name>startziel</name>
        <aussen>215</aussen>
        <innen>215</innen>
    </schiene>
    .
    .
    <schiene>
        <id>9</id>
        <name>kurve45weit</name>
        <aussen>339</aussen>
        <innen>227</innen>
    </schiene>
</schienen>
```

Auch bei der Darstellung der Bahn im Rechner wird analog zu den Autos verfahren. Mit Hilfe der beiden Dateien werden Datenstrukturen gefüllt, welche zur Darstellung der Bahn benötigt werden.

Mit der ersten Struktur wird eine Carrerabahnschiene abgebildet. Die Datenstruktur besteht aus sechs Variablen. Die erste Variable ist für die Speicherung des Segmentnamen, die zweite für die Segment-ID, die dritte beinhaltet die Länge der Innenspur, die vierte die Länge der Außenspur, die fünfte die Kurvenrichtung und die sechste Variable ist ein Flag das bei einem Schleudersensor im Segment gesetzt wird.

Die zweite Datenstruktur beinhaltet vier Variable und sie dient zur Aufnahme von Daten um einen Schleudersensor abzubilden. In der ersten Variablen wird die Anzahl der Geraden bis zu dieser Schiene gespeichert. Mit der zweiten Variablen wird die Position des Schleudersensors in der Schiene gespeichert. Die dritte speichert ein Flag ob der Sensor aktiv ist und die vierte Variable beinhaltet einen Offsetwert.

Die dritte Struktur wird für die Darstellung der Spuren und ihrer Streckenabschnitte benötigt. Jeder Streckenabschnitt zwischen zwei Lichtschranken wird durch solch eine Datenstruktur repräsentiert und dies für jede Spur. Die Struktur besteht aus einem Feld von Zeigern und sieben Variablen. Die erste Variable speichert das Hardwarestatuswort des Streckenabschnitts, in der zweiten wird die Streckenlänge in mm gespeichert, in der dritten Variablen wird die Information über die Spurnummer gehalten, die vierte Variable dient zum Speichern der Anzahl von Schleudersensoren in dem betreffenden Streckenabschnitt. Variable Nummer fünf enthält die Länge bis zu einem eventuellen Gefahrenbereich und Variable sechs die Länge bis zum Ende dieses Bereichs auch in mm. Variable sieben enthält den Namen des Streckenabschnitts. Das Feld in der Datenstruktur ist vom Typ der zweiten Datenstruktur und dient zur Speicherung von Informationen für die Driftsensoren in diesem Abschnitt.

Auch hier werden die ganzen Datenstrukturen in einer eigenen Headerdatei „**track.h**“ definiert.

Zweckmäßigerweise wird die Carrerabahn intern durch eine doppelt verkettete Ringliste aufgebaut. Für jede Spur wird eine solche doppelt verkettete Ringliste erstellt. Bei der ursprünglichen Simulation also maximal zwei Ringlisten. Die Elemente der Listen beinhalten Instanzen der dritten Datenstruktur. Die Funktionalität der Ringliste ist in einer eigenen Datei „**clist.c**“, das Interface dazu in der Datei „**clist.h**“, untergebracht. Die Funktionen die benötigt werden um die Listen aufzubauen und die Daten aus den zwei XML-Dateien zu speichern sind auch hier wieder in einer extra Datei „**track.c**“ implementiert. Im ersten Schritt wird eine doppelt verkettete Ringliste angelegt welche aus den einzelnen Schienenelementen der Carrerabahn besteht. Die Informationen dafür kommen aus den beiden XML-Dateien mit den Layoutinformationen und den Segmentinformationen. In dieser ist die Carrerabahn Schiene für Schiene aufgebaut. Die einzelnen Listenelemente beinhalten jeweils eine Instanz der ersten Datenstruktur mit den Kennwerten der jeweiligen Schiene.

Im nächsten Schritt wird nun auf Basis dieser Liste für jede Spur eine eigene doppelt verkettet Ringliste erstellt welche aus den Segmenten der Bahn besteht und Informationen enthält die speziell für die jeweilige Spurnummer sind. Dabei wird die erste Liste durchlaufen und Anhand der darin enthaltenen Elemente die Kenngrößen wie Spurlänge, Schleudersensorposition, Hardwarestatuswort, Gefahrenbereich Anfang und Gefahrenbereich Ende, Kurvenrichtung. Schleudersensoranzahl und Schleudersensorparameter ermittelt. Danach gibt es für jede an einem Rennen beteiligte Spur eine solche Liste mit Streckenabschnitten.

6.4.2 Kommunikation Simulation – Treiber

Der Treiber legt im Geräteverzeichnis von Linux für jede Spur ein eigenes Gerät an. Für die Simulation wird ebenfalls zusätzlich für jede Spur ein eigenes Gerät angelegt. Im laufenden Betrieb werden von den Anwendungsprogrammen, welche die Autos steuern, neue Geschwindigkeitswerte an den Treiber übermittelt. Im Simulationsmodus wird dann der neue Geschwindigkeitswert, der auf das Gerät einer Spur in den kernel-space geschrieben wurde, auf das entsprechende Gerät der Simulation in den user-space zurückgeschrieben. Dadurch wird der Simulation mitgeteilt, dass das Auto auf der entsprechenden Spur, auf eine neue Geschwindigkeit beschleunigt. Durch diesen Ansatz merkt ein Steuerprogramm keinen Unterschied ob es gerade mit der Simulation oder mit der realen Carrerabahn zusammenarbeitet.

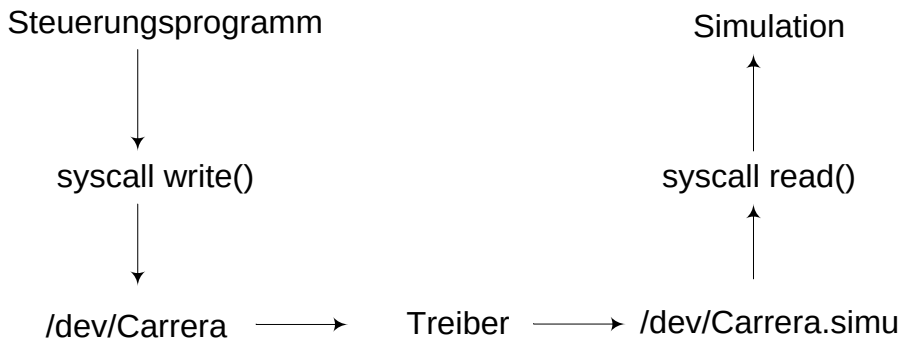

Abbildung 18: Kommunikation Steuerprogramm – Simulation

Fährt ein Auto in der Simulation durch eine virtuelle Lichtschranke oder an einem virtuellen Drift-Sensor vorbei, so dreht sich dieser Vorgang um. Ein Statuswort welches Informationen über den Sensor und die Spur enthält wird nun auf umgekehrtem Weg von der Simulationssoftware auf die entsprechende Gerätedatei der Simulation geschrieben und liegt somit im Treiber bereit.

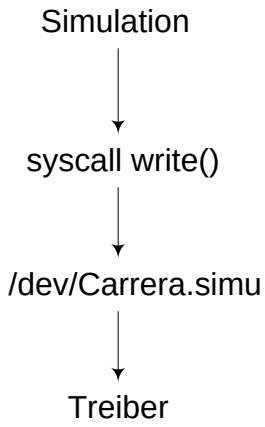

Abbildung 19: Kommunikation Simulation – Treiber

6.4.3 Ablauf der Simulation

Die Simulation die mir für diese Arbeit zur Verfügung gestellt wurde basiert auf einem ereignisorientierten und threadorientierten Ansatz. Für jedes Auto das auf der Bahn simuliert werden soll, werden zwei Threads angelegt. Der erste Thread der sogenannte „**readThread**“ pollt auf dem entsprechenden Gerät welches die betreffende Spur im Rechner darstellt und kann so auf eventuelle Geschwindigkeitsänderungen wie Beschleunigen, Verzögern oder Bremsen reagieren. Geschwindigkeitsänderungen werden durch das Schreiben eines neuen hexadezimalen Werts in die entsprechende Gerätedatei realisiert.

Die Simulation berechnet bei einem auftretenden Ereignis den momentanen Simulationszustand und auf dieser Grundlage die Zeit bis zum Eintreten des nächsten Ereignisses. Für diese Berechnung ist der zweite Thread, der sogenannte **„writeThread“** verantwortlich.

Für die so ermittelte Zeit wird dieser dann schlafen gelegt. Er kann jedoch vorzeitig beim Eintreten einer Geschwindigkeitsänderung durch den zugehörigen „readThread“ geweckt werden und die Berechnung startet wieder von neuem. Ein Teil der Ereignisse ist fest durch das Layout der Carrerabahn vorgegeben. Zu diesen Ereignissen gehören das Passieren einer Lichtschranke und das Auslösen eines Driftsensors. Die Zeiten die dadurch berechnet werden hängen nur von der momentanen Geschwindigkeit oder Beschleunigung ab. Der „writeThread“ besteht aus einer Endlosschleife, welche auf die möglichen festen Ereignisse reagiert. Zu diesen gehört das auslösen und verlassen einer Lichtschranke, das Auslösen eines Driftsensors, das Betreten eines Gefahrenbereichs und das Verlassen eines Gefahrenbereichs. Auf jedes dieser Ereignisse wird entsprechend reagiert.

Die Behandlung der Geschwindigkeitsänderung ist ein variables Ereignis, da dieses durch das Steuerprogramm des jeweiligen Autos vorgeben wird. Wenn also eine solche Geschwindigkeitsänderung eintritt wird der zugehörige „writeThread“ geweckt, die Zeit bis zum eintreten des nächsten festen Ereignisses neu berechnet und der Thread wird wieder schlafen gelegt. Dieses Vorgehen wird für jede an der Simulation beteiligte Spur angewandt. Im Fall der mir zur Verfügung gestellten Simulation für zwei Spuren. Da das Layout der Rennbahn auch Spurwechsel durch Kreuzungen vorsieht, muss auf diese Situation besonders reagiert werden, da eine mögliche Kollision von zwei Fahrzeugen zu vermeiden ist.

Die Threadpaare verfügen über eine gemeinsame Datenstruktur. In der Datenstruktur sind sechs Variablen, zwei Threads , ein Mutex, eine Conditionvariable, eine doppelt verkettete Ringliste mit den Streckenabschnitten für die eigene Spur und eine Datenstruktur mit den Informationen des Autos auf der eigenen Spur. In der ersten Variablen wird der Filedeskriptor der Gerätedatei gespeichert, der vom Treiber für die jeweilige Spur bereitgestellt wird. Die zweite Variable speichert die Spurnummer und die dritte Variable ist ein Flag anhand dessen man erkennen kann ob der Thread das eigene Auto ist oder ein gegnerisches. Die vierte Variable enthält den aktuellen Geschwindigkeitswert des Autos. Die fünfte Variable ist wiederum ein Flag mit dessen Hilfe ein Bremsen des Autos signalisiert werden kann. Mit Hilfe der sechsten Variablen, die auch ein Flag ist kann ermittelt werden ob der Startvorgang auf einer Brücke innerhalb der Bahn beginnt. Die zwei Threads sind der **„writeThread“** und der **„readThread“** einer jeden Spur welche diese Struktur als gemeinsame Datenstruktur haben. Auf Grund der gemeinsamen Datenstruktur ist bei diesem Simulationsansatz die Synchronisation der jeweiligen Threadpaare untereinander ausschlaggebend. Dies wird durch die gemeinsame Verwendung von Mutexen und Conditionvariablen gewährleistet. Mit Hilfe des Mutex kann erreicht werden, dass die beiden Threads synchronisiert auf den Geschwindigkeitswert aus der Gerätedatei zugreifen können. Der „readThread“ schreibend und der „writeThread“ lesend. Mit Hilfe der Conditionvariablen kann der „readThread“ den eventuell schlafenden „writeThread“ über einen neuen Geschwindigkeitswert benachrichtigen und gegebenenfalls aufwecken.

Beim Start der Simulation können eine Reihe von Kommandozeilenparameter übergeben werden. Mit den Parametern kann die eigene Spur und das eigene Auto, sowie das gegnerische Auto festgelegt werden. Da bei dieser Simulation nur zwei Spuren simuliert werden, ergibt sich die gegnerische Spur zwangsläufig aus der Wahl der eigenen. Auch eine zweite Bahnstrecke lässt sich mittels Übergabeparameter festlegen.

Nach dem Auswerten der Parameter werden die „Speedmaps“ der beiden Fahrzeuge mit Defaultwerte gefüllt. Im nächsten Schritt wird Versucht die Gerätedateien jeder Spur zu öffnen und die Dateideskriptoren in die jeweilige Datenstruktur für die Threadpaare geschrieben. Sollte die aus irgendeinem Grund bei einer Spur scheitern, so wird ein Flag gesetzt damit das betreffende Threadpaar nicht initialisiert werden kann. Bei Erfolg wird die Spurnummer, das Flag welches zum Erkennen des eigenen Autos ist, gesetzt. Die Geschwindigkeit wird mit Null vorinitialisiert. Nach diesem Schritt werden die Datenstrukturen für die Autos und die Datenstrukturen für die Ringlisten mit den

jeweiligen Spurabschnitten angelegt. Im letzten Schritt werden dann auch die Threadpaare für die Spuren angelegt.

Die **„readThreads“** starten sofort mit der Überwachung der eigenen Gerätedatei an und die **„writeThreads“** legen sich sofort schlafen, da am Anfang noch keine Geschwindigkeit in der Gerätedatei steht. Wenn ein Steuerprogramm für ein Auto einen Geschwindigkeitswert in hexadezimaler Form, der Größer als Null ist, in seine zugehörige Gerätedatei schreibt, so bekommt dies der zugehörige „readThread“ mit und weckt seinen **„writeThread“**. Der Simulationsvorgang hat somit begonnen. In der Funktion die dem **„writeThread“** zugeteilt ist, wird jetzt die Zeit bis zum ersten Auftreten eines festen Ereignisses bei gleich bleibender aktueller Geschwindigkeit berechnet. Beim Starten der Simulation ist dies für jedes Auto das an einem Simulationsvorgang beteiligt ist das Auslösen einer Lichtschranke. Beim Auslösen einer Lichtschranke wird in der Ringliste mit den Streckenabschnitten ein Element weiter gerückt. Die Simulation definiert für das passieren einer Lichtschranke zwei feste Ereignisse. Das erste ist das Auslösen der Lichtschranke und das darauf folgende ist das Verlassen dieser. Der Sensor bleibt also für eine gewisse Zeit lang aktiv. Diese Zeit richtet sich nach der momentanen Geschwindigkeit des Autos und nach der Länge der Führung des Autos in der Schiene. Die doppelt verkettet Ringliste mit den Informationen über die Streckenabschnitte die jeder Thread einer jeden Spur als Parameter besitzt, ist so aufgebaut, dass das erste Element der Streckenabschnitt der Bahn ist, auf dem die Autos beim Start stehen. In der Simulation stehen die Autos direkt vor der ersten Lichtschranke des darauf folgenden Elements.

Es werden nun die ersten Berechnungen angestellt. Dazu gehören das Ermitteln der Geschwindigkeitsdifferenz zwischen der Anfangsgeschwindigkeit und der in der Gerätedatei stehenden Endgeschwindigkeit, da über diesen Wert in der Datenstruktur des Threads die passende Beschleunigung ermittelt werden kann, die in der Struktur „**sCarData.aPositive[valueDiff]**“ zu finden ist. Der aktuelle Geschwindigkeitswert wird einfach aus der „Speedmap“ ermittelt. Es werden nun die Zeit bis zum Erreichen der Endgeschwindigkeit und der bis dahin benötigte Weg berechnet. Mit Hilfe des Weges wird festgestellt ob der Beschleunigungsvorgang zwischen zwei Ereignissen abgeschlossen werden kann oder nicht. Dann wird das Hardwarestatuswort des Streckenabschnittes aktualisiert und an den Treiber übermittelt, der somit die Information erhält auf welcher Spur und auf welchem Abschnitt sich das Auto gerade befindet. Der Thread wird nun bis zum Auslösen des Ereignisses beim Verlassen der Lichtschranke schlafen gelegt. Sollte für innerhalb dieser Zeit ein neuer Geschwindigkeitswert in die Datei geschrieben werden, so würde der Thread geweckt und die Berechnungen müssten von neuem gestartet werden. Nach Ablauf des Zeitfensters wird der Thread geweckt und behandelt das neue Ereignis Lichtschranke verlassen. Auch hier wird nun das Hardwarestatuswort des Streckenabschnitts aktualisiert und an den Treiber übermittelt. Im folgenden Schritt muss nun bestimmt werden welches Ereignis als nächstes Auftritt. Als Möglichkeiten kommen wieder eine Lichtschranke, ein Schleudersensor oder das Betreten eines Gefahrenbereichs in Betracht. Die Information darüber kann sich die Simulation aus der Datenstruktur des Threads und der darin enthaltenen Ringliste besorgen, wo über Flags abgerufen werden kann über welche Gegebenheiten ein Streckenabschnitt verfügt. Über den Vergleich der Positionen kann nun ermittelt werden welches Ereignis als nächstes Eintritt.

Mit den Informationen über Position und aktueller Geschwindigkeit kann nun wieder das Zeitfenster berechnet werden und der Thread wird wieder Schlafen gelegt. Tritt als nächstes Ereignis das passieren eines Schleudersensors auf, so wird wieder das Hardwarestatuswort des Abschnittes aktualisiert und übermittelt. Dazu wird die Geschwindigkeit des Autos beim eventuellen Schleudern ermittelt, die mit Hilfe des Offsets aus der Datenstruktur des Schleudersensors, der vorherigen Kurven- bzw. Schleudergeschwindigkeit und der Geraden Anzahl des Streckenabschnitts ermittelt wird. Mit Hilfe dieser Geschwindigkeit wir aus der „Speedmap“ der zugehörige Schleudersensorwert ermittelt. Mit Hilfe von Zufallszahlen wird nun die Zeit ermittelt die der Thread zusätzlich benötigt, da durch das rutschen keine konstante gleichförmige Bewegung mehr vorhanden ist. Dies alles geschieht in einer extra Funktion und die so berechnete Zeit wird zurückgegeben. Sollte festgestellt werden, dass das Auto raus geflogen ist, so wird ein fest definierter Wert zurückgegeben. Es wird nun bestimmt welches feste Ereignis als nächstes Auftritt und der Thread wird wieder anhand dieser Bestimmung und des Wertes der von der Funktion zur Berechnung des Schleuderverhaltens ermittelt wurde schlafen gelegt.

Wird als nächstes Ereignis der Eintritt in einen Gefahrenbereich festgestellt so wird wieder ermittelt welches Ereignis darauf folgt. Eventuell nochmal das passieren eines Schleudersensors oder das Ereignis beim Verlassen des Gefahrenbereichs. Danach muss überprüft werden ob das gegnerische Auto sich nicht schon in dem gleichen Gefahrenbereich befindet. Das ist aber nur von Relevanz wenn es sich nicht schon auf der gleichen Spur befindet. Zur Behandlung dieser Situation wurden in der Simulation zwei Mutexe und eine Conditionvariable vorgesehen. Ein Mutex schützt das **danger-Flag** , der andere das **colision-Flag** , über welche die zwei konkurrierenden Autos feststellen können ob sie Gefahr laufen zu kollidieren und gegebenenfalls die Durchfahrt eines Gefahrenbereichs synchronisieren können. Die Conditionvariable überwacht zusätzlich das **colision-Flag**. Wenn jetzt mittels gesetztem **danger-Flag** festgestellt wird, dass das das gegnerische Auto auch im Gefahrenbereich ist, so wird das **colision-Flag** gesetzt und der Thread wird für maximal sechs Sekunden schlafen gelegt. Er kann aber durch Eintreten der Bedingung durch das **colision-Flag** vorzeitig geweckt werden. Nach dem Wecken des Threads wird das **colision-Flag** wieder gelöscht. Wenn festgestellt wird, dass kein gegnerisches Auto im Gefahrenbereich vorhanden ist, wird das **danger-Flag** gesetzt.

Wenn am Anfang der Ereignisbehandlung des Gefahrenbereichs Eintritt festgestellt wurde, dass das folgende Ereignis das Verlassen des Gefahrenbereichs ist, so wird diese Situation als nächstes behandelt. Es wird nun geprüft das eigene Auto warten musste und die Berechnungen für Beschleunigungszeit und Beschleunigungsweg wird durchgeführt. Das **danger-Flag** wird gelöscht und es wird bestimmt welches feste Ereignis als nächstes Auftritt.

Bei den ganzen hier beschriebenen Vorgängen wird immer auch auf den Fall reagiert das während einer solchen Ereignisbehandlung das variable Ereignis der Geschwindigkeitsänderung eintreten kann, was in einer Beschleunigung, einer Verringerung der Geschwindigkeit oder einer Vollbremsung resultieren kann. Die ganzen Parameter wie Beschleunigungszeiten , Startzeiten, Restzeiten, Beschleunigungsweg,

restlicher Streckenabschnittsweg usw. die für den korrekten Ablauf der Simulation wichtig sind, werden dann entsprechend neu berechnet und die Simulation reagiert entsprechend auf diese Veränderung.

6.5 Erweiterungen der Simulation

Da der Wunsch von Professor Mächtel darin bestand die alte Simulation mit dem zugehörigen Treiber möglichst zu verwenden, musste die zur Verfügung gestellte Simulation in vielen Punkten angepasst und abgeändert werde. Die Simulation musste auf den Betrieb mit bis zu vier Autos und auf das erheblich größere und komplexere Streckenlayout der HTWG umgestellt werden. Auch die Realisierung der Schleudersensoren bedurfte einer Anpassung und Erweiterung.

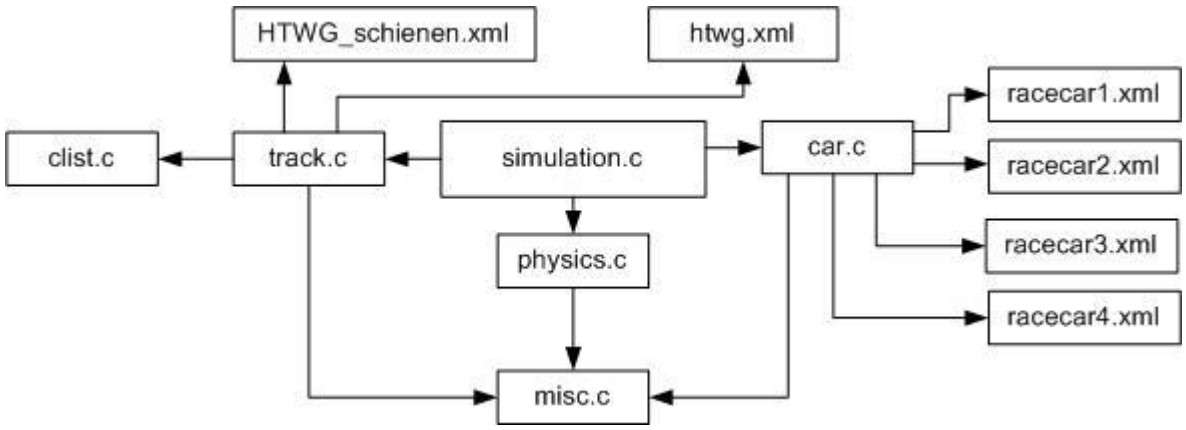

Abbildung 20: Aufbau der Simulation

6.5.1 Anpassungen an den Datenstrukturen der Autos

Die XML-Dateien für die Autos wurden um ein neues Tag erweitert das die physikalische Kenngrößen Gewicht, Länge, Roll- und Gleitreibungskoeffizient, Spurabstand, Länge der Finne zur Führung des Autos, Radius vom Drehpunkt bis zur Hinterachse enthalten und zwei Tags die als Offsets für die alternative Schleudersensor Implementierung fungieren. Auch die Tags für die möglichen Geschwindigkeiten wurden verändert. Das Schleudersensor-Tag wurde mit Tags für die Spuren versehen. Jedes dieser neuen Tags beinhaltet sieben Tags mit den Kurven die Schleudersensoren enthalten. In diesen Tags sind dann die entsprechenden Werte des Schleudersensors der Spur mit der entsprechenden Geschwindigkeit hinterlegt. Ein zweites neuer Tag wurde eingeführt um die Winkelbereiche zu spezifizieren welche von einem Sensorwert des Schleudersensors abgedeckt werden.

Dateiaufbau geänderte Fahrzeug-Datei:

```
<fahrzeug name="xyz">
    <data>
        <width>70.0</width>
        <distance>100.0</distance>
        <rail>27.0</rail>
        <length>250.0</length>
        <friction_G>0.85</friction_G>
        <friction_R>0.014</friction_R>
        <distance2>100.0</distance2>
        <mass>500.0</mass>
        <t_offset>1.1</t_offset>
        <a_offset>0.8</a_offset>
    </data>
    <speedmap>
        <geschwindigkeit>
            <hex>0x20</hex>
            <real>509</real>
            <schleudersensor>
                <track1>
                    <K_210_E>0</K_210_E>
                    <K_120>0</K_120>
                    <K_360>0</K_360>
                    <K_240>0</K_240>
                    <K_180>0</K_180>
                    <K_120_E>0</K_120_E>
                    <K_210>0</K_210>
                </track1>
                .
                .
            </schleudersensor>
        </geschwindigkeit>
        .
        .
    </speedmap>
        <driftangles>
            <value>
                <start>1</start>
                <end>4</end>
            </value>
            .
            .
        </driftangles>
</fahrzeug>
```

Das Tag für die mögliche Beschleunigungen wurde nicht geändert. Es wurde eine neue Datenstruktur eingeführt, die für die alternative Driftwinkelberechnung benötigt wird. Sie enthält zwei Variablen mit deren Hilfe ein Winkelbereich festgelegt werden kann. Die zweite neue Datenstruktur enthält sieben Variablen, die stellvertretend für die Schleudersensoren in den sieben Kurven stehen. Die Struktur die für die Geschwindigkeitswerte benötigt wird, wurde um ein eindimensionales Feld mit vier Elementen erweitert und die zwei Variablen welche zuvor die innere und äußere Geschwindigkeit aufnehmen konnten wurden entfernt. Dies war nötig um von zwei auf vier Spuren zu erweitern. Der Datenstruktur welche das Auto im Rechner darstellt wurden sieben neue Variablen und ein eindimensionales Feld hinzugefügt. Die sieben neuen Variablen werden für die physikalischen Kenngrößen des Autos benötigt die neu in der XML-Datei hinzugekommen ist. Das neue Feld ist vom Typ der ersten neuen Struktur, welche einen Winkelbereich darstellt, es kann 14 Elemente aufnehmen. Es sind deshalb 14 Elemente weil der Schleudersensor zwischen einen Wertebereich von 1 bis 14 hat. Die maximalen Größen bei den beiden Felder für die Beschleunigung (**sVAccelerate[6][50][12]**) und die Verzögerung(**sVDecelerate[50][12]**) wurde auch von 20 auf 50 Elemente geändert, da mit dem neuen Bahnlayout eine maximale Beschleunigungsstrecke von 2500 mm angenommen wurde.

Ein Beispiel:

Ein Aufruf von „**vAccelerate[2][4][1]“** würde die Geschwindigkeit liefern wenn das Auto eine Anfangsgeschwindigkeit hätte, welchem dem achten Eintrag in der „Speedmap“ entspricht, und auf einer Distanz von 4 * 50 mm mit einer Beschleunigung, die dem 3 Eintrag im Feld der möglichen Beschleunigungen entspricht, beschleunigt oder verzögert würde.

Für das Bremsen mit der maximalen Bremsverzögerung existiert ein zweidimensionales Feld in der Struktur des Autos. Dieses Feld liefert die Geschwindigkeit, wenn mit maximaler Bremsverzögerung eine Distanz die ein Vielfaches von 50 ist, gebremst wird. Die erste Dimension gibt hier den Weg an. Sie hat genau die gleiche Größe wie im Feld für die Geschwindigkeitsänderung. Es werden 50 Elemente gespeichert. Die zweite Dimension gibt die Geschwindigkeit an bei der gebremst wird. Die maximale Anzahl an Elementen ergibt sich analog zu der Struktur für die Geschwindigkeitsänderungen. Auch hier wird wieder jede achte Geschwindigkeit gespeichert.

Ein Beispiel:

Ein Aufruf von **vDecelerate[2][1]** würde die Geschwindigkeit liefern wenn das Auto auf einer Strecke von 2 * 50 mm mit einer Anfangsgeschwindigkeit, die dem achten Eintrag in der Struktur „Speedmap“ entspricht, mit maximaler Verzögerung abbremst.

Das Auto hat also insgesamt fünf Strukturen. Die Struktur **„carData“** enthält ein Feld vom Typ der Struktur **„speed“** welches ein Feld vom Typ der Struktur sensorData enthält. Des Weiteren enthält die Struktur **„carData“** noch ein dreidimensionales Feld vom Typ der Struktur vAccelerate und ein zweidimensionales Feld vom Typ der Struktur vDecelerate.

Im ersten Schritt werden die Werte des Autos wie z.B. Gewicht, Reibungskoeffizienten usw. aus der XML-Datei ausgelesen und den entsprechenden Variablen in der Struktur carData zugewiesen. Als nächstes wird das Feld für die Geschwindigkeits- und Sensorwerte mit den Werten aus der Datei gefüllt. Zum Schluss wird noch das Feld mit den Beschleunigungs-werten gefüllt und die Variable mit der maximalen Bremsverzögerung initialisiert. Damit ist das Einlesen der Daten aus der XML-Datei beendet. Im nun Folgenden Schritt werden die Werte für das dreidimensionale und das zweidimensionale Feld berechnet. Die berechneten Geschwindigkeitswerte werden mit den von den Autos realisierbaren Geschwindigkeits- werten verglichen und es wird die Geschwindigkeit eingetragen welche zu dem berechneten Wert den kleinsten Unterschied hat.

Die Funktion für das ermitteln der Daten aus der XML-Datei wurde so angepasst, dass die erweiterten alten und die neuen Datenstrukturen mit den neuen Werten aus der XML-Datei gefüllt werden können. Auch die Funktion welche die Werte für die Beschleunigungs- und Verzögerungsdatenstruktur berechnet wurde an die neuen Strukturen angepasst und an den entsprechenden Stellen den neuen Änderungen angepasst.

6.5.2 Anpassungen der Datenstrukturen für die Rennbahn

Für das Layout der Rennbahn an der HTWG wurden auch zwei XML-Dateien erstellt. Wie auch in der Ursprünglichen Version der Simulation wird eine Datei für das Layout der Bahn verwendet und die andere zur Beschreibung der Schienenelemente die von der neuen Bahn benutzt werden. Die Datei zur Beschreibung der Schienenelemente wurde aber im Bezug auf die darin enthaltene Information geändert. Jedes verwendetet Schienenelement wurde mit zusätzlichen Informationen versehen die für die Simulation der HTWG-Rennbahn wichtig sind. Dadurch wurden den Unterschieden der Simulation Rechnung getragen. Die zusätzlichen Informationen werden für die Simulation von vier Spuren, Steigungen und Gefällen und für die alternative Schleudersensor-Simulation benötigt. Jedes Schienen-Tag hat in der neuen Version vier Tags für die Spurlänge, vier Tags für die eventuelle Schleudersensorposition, vier Tags für die Grenzgeschwindigkeit für Kurvenfahrten, vier Tags mit den eventuellen Kurvenradien und ein Tag für einen eventuellen Steigungswinkel des Segments bzw. der Schiene.

Dateiaufbau geänderte Segment-Datei:

```
<schiene>
    <id>9</id>
    <name>kurve_360_mittel</name>
    <Spur_4>4680</Spur_4>
    <Spur_3>4056</Spur_3>
    <Spur_2>3432</Spur_2>
    <Spur_1>2808</Spur_1>
    <SENSOR_POS_4>642</SENSOR_POS_4>
    <SENSOR_POS_3>562</SENSOR_POS_3>
    <SENSOR_POS_2>478</SENSOR_POS_2>
    <SENSOR_POS_1>407</SENSOR_POS_1>
    <V4>2400</V4>
    <V3>2300</V3>
    <V2>2100</V2>
    <V1>1900</V1>
    <R4>744</R4>
    <R3>642</R3>
    <R2>546</R2>
    <R1>444</R1>
    <alpha>4.0</alpha>
    <sAlpha>40</sAlpha>
</schiene>
```

Im Zuge der Änderungen an der XML-Datei mussten die Datenstrukturen für die Rennbahn auch abgeändert werden. In der ersten Datenstruktur, welche für den Aufbau der Ringliste mit den Schienen der Rennbahn benötigt wird, wurden die zwei Variablen für die innere und äußere Länge der Schiene entfernt. Die Struktur wurde um fünf Felder, die alle vier Elemente aufnehmen können und einer Variablen für den Steigungswinkel erweitert. Die

fünf Felder können alle vier Elemente aufnehmen und jedes Element ist für eine Spur gedacht. Das Element mit dem Index 0 repräsentiert die 1 Spur und das letzte Element mit dem Index 3 die vierte Spur. Die fünf Felder werden für die Spurlängen des Segments, die Schleudersensorposition, die Grenzgeschwindigkeiten, die Radien und für die Bogenlänge von Kurvenanfang bis zur Schleudersensorposition benötigt. Das Feld mit den Bogenlängen von Kurveneintritt bis zu den Schleudersensorpositionen wird nicht aus einer Datei mit Werten gefüllt. Sie werden mit Hilfe der Gradzahl die in der Datei im Tag **<sAlpha>** angegeben ist und den jeweiligen Radien berechnet. Die Struktur für den Schleudersensor hat nur noch zwei Variablen, eine Für die Position innerhalb des Bahnabschnitts und eine die als Flag fungiert um anzuzeigen das der Sensor aktiviert ist oder nicht benutzt wird. Auch die Datenstruktur die als Elemente dient um die Ringlisten für jede Spur aufzubauen wurde geändert bzw. angepasst. Da in der HTWG-Bahn immer nur maximal ein Schleudersensor für jeden Streckenabschnitt benötigt wird, wurde das Feld von Zeigern in einen einfachen Zeiger vom Typ der Schleudersensordatenstruktur geändert. Ein neuer Zeiger vom Typ der Schienendatenstruktur wurde eingeführt um Zugriff auf die Radien, Grenzgeschwindigkeiten und den Steigungswinkel zu bekommen. Die Variable für die Anzahl der Schleudersensoren im Segment wurde entfernt.

Auch an dieser Stelle mussten die Funktionen die am Aufbau der Ringlisten beteiligt sind an die neue Simulation angepasst werden. Im einzelnen Bedeutete dies die Funktion welche die Daten aus den XML-Dateien liest und in den Datenstrukturen speichert an die neue XML-Datei für die Segmentbeschreibung anzupassen. Sie wurde dahingehend erweitert, dass die neuen Elemente der Datenstrukturen mit den neuen Daten aus der neuen XML-Datei gefüllt werden konnten. Alle beteiligten Funktionen mussten an den Sachverhalt der vier Spuren angepasst werden.

Die Funktion welche die Ringliste für jede Spur erstellt musste natürlich auch an die neue Situation angepasst werden, dass sich im Vergleich zur alten Simulation einige Dinge gravierend geändert haben. Es musste eine Logik implementiert werden die unter Berücksichtigung der vier möglichen Spuren und den möglichen Spurwechseln zwischen zwei benachbarten Autos die Strecke für jedes Auto korrekt zusammenstellt und als Ringliste aufbereitet. Dabei wird jedes Element der ersten Ringliste betrachtet und dann entschieden, ob es noch zum aktuellen Streckenabschnitt gehört oder ob mit dieser Schiene schon ein neuer Abschnitt beginnt und eine neue Datenstruktur dafür angelegt werden muss und die alte in die Ringliste eingefügt wird. Im Vergleich zur ursprünglichen Simulation sind die Lichtschranken bei der HTWG-Rennbahn nicht immer an derselben Position einer Schiene. Aus diesem Grund musste eine Logik implementiert werden, die dafür sorgt, dass ein Streckenabschnitt zwischen zwei Lichtschranken korrekt berechnet werden kann. Da ein Segment in der Bahn zweimal vorkommt nur mit dem unterschied das es sich einmal um eine Linkskurve und das andere mal um eine Rechtskurve handelt wurde auch hierfür eine Logik implementiert, die in dem jeweiligen Fall die richtige Spur berücksichtigt beim erstellen der Elemente für die Ringliste einer jeden Spur. Da jeder Streckenabschnitt durch ein Hardwarestatuswort für den Treiber definiert wird, musste die Generierung dieses Statusworts auch an das neue Layout der Statuswörter für die HTWG-Bahn und Treiber angepasst werden.

6.5.3 Anpassungen des Simulationsablaufs

Die Simulation für die HTWG-Bahn basiert auch auf dem thread- und ereignisorientierten Ansatz der Version die mir von Professor Mächtel zur Verfügung gestellt wurde. Das Threadmodell musste jedoch um zwei Threadpaare für die zwei zusätzlichen Spuren erweitert werden. Auch die Datenstruktur für die Threads wurde geändert. Die Anzahl der möglichen Übergabeparameter wurde angepasst, da es nun möglich sein muss sein Auto auf jeder Spur zu platzieren und für insgesamt bis zu vier Autos Konfigurationsdateien anzugeben. Ein neuer Parameter wurde eingeführt um zwischen den Simulationsansätzen des Schleudersensors zu wechseln und ein weiterer Parameter um die Simulation mit einer GUI zu starten. Der generelle Simulationsablauf wurde auch so erweitert, dass die in der Rennbahn enthaltene Steigung und das Gefälle bei den Berechnungen berücksichtigt werden und in die Ergebnisse der Berechnungen in der Simulation mit einfließen. Die Ursprüngliche Simulation des Schleudersensors wurde auch modifiziert.

Die Datenstruktur für die Threads wurde um zwei neue Parameter erweitert. Der erste Parameter wird zur Identifizierung des eigenen Autos benutzt. Mit dem zweiten Parameter wird festgelegt welches Auto der direkte Gegner im Simulationsbetrieb ist. Dies ist wichtig, da der Thread beim durchfahren einer Gefahrenstellen wissen muss auf welchen Thread er achten muss. Bei der ursprünglichen Simulation war dies immer klar gegeben da es nur noch einen anderen Thread gab. Jetzt stehen aber drei zur Auswahl. Die Variable die als Flag für einen Brückenstart fungierte wurde aus der Datenstruktur entfernt. Um auf die Steigung und das Gefälle in der HTWG-Bahn im Simulationsbetrieb zu reagieren wurde ein neue Datei angelegt und das zugehörige Interface in eine neue Headerdatei. Hier sind alle Funktionen implementiert die bei der Berechnung von Geschwindigkeiten, Zeiten und Strecken beteiligt sind, wenn eine Steigung oder ein Gefälle im aktuellen Segment vorliegt. In diesem Fall kann nicht mehr auf die Datenstruktur des Autos alleine zugegriffen werden, da die Felder mit den Beschleunigungen, den Verzögerungsgeschwindigkeiten und den Beschleunigungsgeschwindigkeiten nicht mehr stimmen. In solch einem Fall muss immer noch die Hangabtriebskomponente als zusätzlicher verzögernder oder beschleunigender Faktor in die Berechnungen mit einbezogen werden. Die Datenstrukturen können in diesem Fall nur sehr begrenzt benutzt werden.

Die Simulation startet wie in der ursprünglichen Version nach dem erkannt wurde, dass ein Geschwindigkeitswert in die Gerätedatei geschrieben wurde. Die Logik die für das Abarbeiten der fest eintretenden Ereignisse zuständig ist musst an den Stellen angepasst werden wo Hardwarestatuswörter modifiziert oder geändert wurden, da der alte Aufbau dieser Statuswörter nicht zu dem der HTWG kompatibel ist. Es wurde dahingehend geändert, dass immer ein Feld aus zwei Hardwarestatuswörtern an den Treiber übermittelt wird. Das erste Element enthält das Statuswort des aktuellen Abschnitts und das zweite Element gegebenenfalls eine neu eingeführtes Statuswort mit dem Schleudersensorwert. Da auf der HTWG-Bahn immer nur maximal ein Schleudersensor pro Abschnitt installiert ist wurde dieser Umstand auch in der Logik abgeändert. Zur Realisierung eines flexiblen Austauschs der Schleudersensorimplementierungen wurde ein Funktionszeiger eingeführt.

6.5.4 Implementierung von Alternativen zur Driftwinkelberechnung

Die Implementierung zur Generierung des Schleudersensorwertes auf Basis des ursprünglichen Ansatzes wurde auch an einigen Stellen modifiziert. Die jetzige Implementation, geht davon aus, dass die Schleudersensorwerte für ein Auto für jede mögliche Geschwindigkeit und Kurve durch Experimente ermittelt werden und in der XML-Datei bereitsteht. Dieser Ansatz hat den Vorteil, dass während des Simulationsbetriebs nur minimale Berechnungen getätigt werden müssen und die Simulation vom Ergebnis was den Schleudersensorwert angeht zu fast hundert Prozent mit der Realität übereinstimmt. Lediglich die Zeiten müssen berechnet werden. Bei der alten Simulation wurde dies auf der Basis von Zufallszahlen bewerkstelligt. Da die Simulation immer die Zeit bis zum nächsten Eintritt eines festen Ereignisses berechnet, dabei aber immer mit den Feldern der Beschleunigungen und Verzögerungen arbeitet, werden die berechneten Zeiten zu kurz da Reibungseffekte beim Schleudern vernachlässigt werden. Diese zusätzliche Verlängerung des Zeitfensters die durch den Schleudervorgang des Autos auftreten muss berechnet werden. Hier wurde ein anderer Ansatz als der der alten Simulation gewählt und es wurde auf Zufallszahlen verzichtet. Der neue Ansatz berechnet die Zeiten unter Einbeziehung der auftretenden Reibung. Dies ist zwar auch eine sehr starke Vereinfachung der physikalischen Vorgänge, da für die Simulation eine konstante Reibkraft angenommen wurde welche in der Realität dynamisch ist.

Des Weiteren wird im dem modifizierten Ansatz erst eine Überprüfung vorgenommen die ermittelt ob die aktuelle Kurvengeschwindigkeit überhaupt größer als die Grenzgeschwindigkeit der Kurve für die aktuelle Spur ist. Ist dies nicht der Fall, so wird keine Berechnung angestellt und die Simulation fährt fort ohne den Schleudersensor zu betrachten. Dies ist in der ursprünglichen Simulation nicht so gelöst. Dort wird immer eine Berechnung durchgeführt, egal ob die Geschwindigkeit des Autos zu einem Schleudervorgang führen kann oder nicht.

In einer weiteren Alternative wird der Ansatz bei dem die Zeit per Zufallszahlen berechnet wird mit dem modifizierten eigenen Ansatz kombiniert. Hier werden die Schleudersensorwerte wieder aus der geänderten Datenstruktur gelesen und die Zeit die bis zum passieren des Sensors benötigt wird mit Hilfe des Ansatzes der Zufallszahlen berechnet.

Die dritte alternative Implementierung versucht auf den Sensorwert mit Hilfe eines vorher berechneten Driftwinkel zu schließen. Zu diesem Zweck wurde die XML-Datei, die die Kennwerte eines Autos enthält, um das Tag **<driftangles>** erweitert. In diesem Tag können den 14 Möglichen Schleudersensorwerten Winkelbereiche zugeteilt werden, damit anhand eines berechneten Winkels ein Sensorwert bestimmt werden kann. Die Datenstruktur des Autos kann diese Werte in einem Feld mit 14 Elementen aufnehmen. Bei diesem Ansatz wird zu erst die Zeit berechnet, welche unter Berücksichtigung der entstehenden Reibung beim Schleudervorgang benötigt wird, um bis zu dem Schleudersensor zu gelangen. Im folgenden Schritt wird nun der Winkel berechnet, denn das Auto zu seiner Bahntangente angenommen hat. Der Winkel wird unter der Annahme berechnet das die Kraft, welche zum Ausbrechen des Autos führt, die Resultierende aus der Zentrifugalkraft minus der

auftretenden Reibkraft zwischen Fahrbahn und Reifen ist. Mit Hilfe dieser Annahme lässt sich die Beschleunigung bestimmen, die im Moment des Schleudervorgangs an der Hinterachse des Autos angreift. Mit dieser Beschleunigung lässt sich nun die Winkelgeschwindigkeit der Kreisbewegung um den Drehpunkt des Autos bestimmen und mit Hilfe der Winkelgeschwindigkeit und der zuvor berechneten Zeit den dabei überschrittenen Winkel. Da bei diesem Ansatz die Vereinfachung einer konstanten Beschleunigung getroffen wurde, welche aber in der Realität nicht konstant ist, ist dieser Ansatz ungenau. Er berechnet immer größere Winkel als sie tatsächlich vorkommen. Um diesen Fehler zu korrigieren wurde ein Offset als Korrekturwert eingeführt. Unter Verwendung des so berechneten Driftwinkel und den Winkelbereichen aus der XML-Datei kann jetzt eine Zuordnung des Winkels zu einem Bereich vorgenommen werden und so der Sensorwert bestimmt werden. Da der Thread immer nach der Bestimmung des Winkels, für die Zeit die er benötigt um an den Sensor zu gelangen, schlafen gelegt wird, ist nun noch die Zeit für das zusätzliche Warten zu bestimmen. Der normale Simulationsablauf berechnet seine Zeiten ja immer nur auf Grundlage der Datenstrukturen die vor dem Start berechnet wurden. Es wird nun die Differenz zwischen der Zeit die mit Reibung berechnet wurde und der Zeit, die ohne Reibung berechnet wurde, bestimmt und der Thread für dieses kleine Zeitfenster zusätzlich schlafen gelegt. Bei dem ursprünglichen Simulationsmodell wurde diese Zeit per Zufallszahlen und Offsets bestimmt.

6.5.5 Integrierung einer GUI

Ein weiteres Ziel dieser Diplomarbeit bestand darin eine grafische Benutzeroberfläche für die Simulation zu implementieren. Um dies zu realisieren musste eine API gefunden werden, mit deren Hilfe eine GUI möglichst einfach in die bestehende Simulation zu integrieren wäre. Mögliche Kandidaten hierfür waren OpenGL und SDL.

Die Wahl viel dann recht schnell auf SDL. Da OpenGL seine Stärke als API im erstellen von 3D-Grafiken hat, die GUI für die Simulation aber als 2D-Grafik realisiert werden sollte, wäre der Aufwand mit OpenGL einfach zu groß gewesen. Mit SDL ist das gewünschte Ergebnis in kürzerer Zeit und mit weniger Einarbeitungsaufwand zu bewerkstelligen.

Bei SDL handelt es sich um eine freie Multimedia Bibliothek. Sie stellt dem Programmierer eine API zur Verfügung um relativ einfach Anwendungen zu schreiben, die einen low-level Zugang zu Video-, Audio-, Maus- und Tastaturgeräten bieten. Die Bibliothek wurde von Sam Lantinga, einem Spielentwickler bei der damaligen Firma Loki Software, entwickelt. SDL ist in C geschrieben, kann aber über Adapter in einer Vielzahl von Programmiersprachen eingesetzt werden. Die API zeichnet sich durch Plattformunabhängigkeit aus und ist für eine Reihe von Betriebssystemen erhältlich. Der Vorteil von SDL gegenüber anderen APIs, wie OpenGL oder DirectX liegt, wie der Name wahrscheinlich schon erahnen lässt, in der relativen Einfachheit und Kompaktheit des Codes. Man kann mit relative geringem Aufwand und einer kurzen Einarbeitung in die API eigenen Code schreiben. Der Nachteil liegt in der Beschränkung auf 2D-Grafik. Bei Bedarf kann dieser Nachteil aber durch die Einbindung von OpenGL wieder ausgeglichen werden.

Durch die gemeinsame Nutzung ist es auch somit auch Möglich bei Bedarf 3D-Grafiken mit SDL zu programmieren.

Bei der Entwicklung der GUI gibt es zwei mögliche Ansätze. Der erste Ansatz basiert auf einer kontinuierlichen Simulation der Carrerabahn. Zu jedem Zeitpunkt wenn ein Bild der GUI gezeichnet wird, muss der Gesamtzustand der Carrerabahn bekannt sein. Zu diesem Zustand gehört die Momentangeschwindigkeit jedes aktiven Autos, die Momentanbeschleunigung jedes aktiven Autos, die Position jedes Autos auf seiner Spur und die Werte der Sensoren die zu diesem Zeitpunkt eventuell aktiv sind. Dieser Ansatz für die GUI setzt voraus, dass der Simulationsvorgang kontinuierlich in einer festgelegten Zeitscheibe neu bestimmt wird. Der Gesamtzustand der Simulation wird dann nach Ablauf dieser Zeitscheibe neu bestimmt und gezeichnet. Durch diese Vorgehensweise könnte man die Autos auf der GUI in Echtzeit verfolgen.

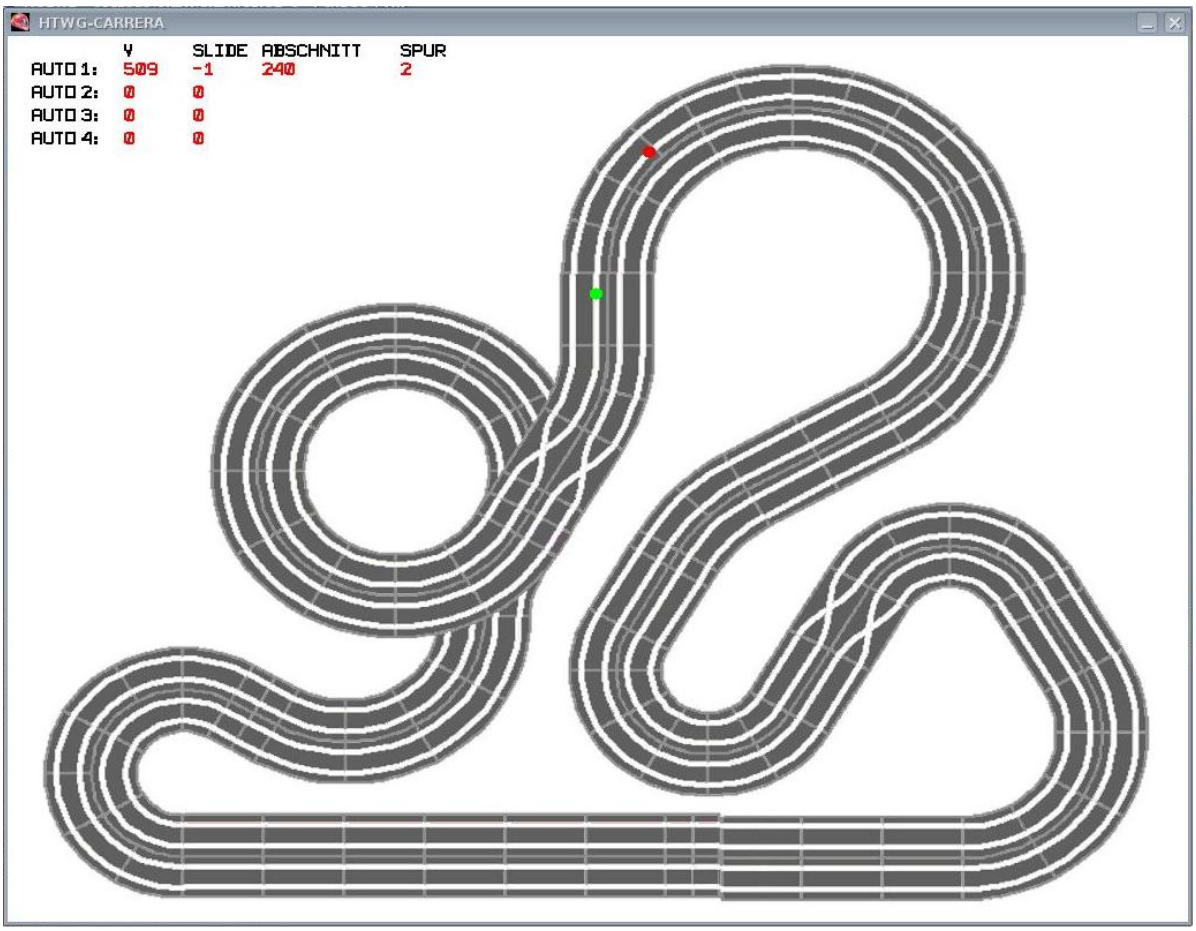

Abbildung 21: GUI

Dieser Ansatz wurde aber bei der Simulation nicht angewendet. Aus diesem Grund ist auch der Ansatz für die GUI ein anderer. Um die GUI zu realisieren habe ich mich für den zweiten Ansatz entschieden. Er hat den Nachteil, dass man die Autos nicht als solche auf der Bahn fahren sieht, aber man sieht wenn sie die Sensoren auslösen. Der Vorteil dieser Implementierung basiert auf der Tatsache, dass dadurch das alte Konzept der Simulation, welche ich von Professor Mächtel zur Erweiterung und Anpassung bekommen hatte, verwendet werden konnte. Dieses Konzept basiert nicht auf dem Ansatz der kontinuierlichen Simulation. Der Ansatz beruht wie schon weiter oben ausgeführt darauf, dass für jedes Fahrzeug zwei Threads zuständig sind. Ein Thread reagiert nur auf neue Geschwindigkeiten. Der zweite aber realisiert die eigentliche Simulation des Autos auf seiner Spur. Dies geschieht aber nicht kontinuierlich. Der Simulationsvorgang beschränkt sich auf das Abarbeiten von Ereignissen die durch die Sensoren auf der Bahn vorgegeben sind und plötzlich eintretende Ereignisse durch Geschwindigkeitsänderungen. Es werden die Zeiten zwischen den Ereignissen berechnet und die Simulationsthreads werden für diese Zeiten schlafen gelegt und können nur durch neue Geschwindigkeitsänderungen unterbrochen werden. Dieses Schlafen der Simulationsthreads verhindert eine kontinuierliche Simulation, da der Gesamtzustand der Simulation nie bekannt ist. Es ist immer nur der Zustand eines Autos bekannt, da es keine gemeinsame Datenstruktur der Simulation gibt, sondern jedes Auto seine eigene Datenstruktur besitzt. Der Zustand eines Autos ist deshalb entweder wenn ein Ereignis durch die Sensoren ausgelöst wird oder wenn eine neue Geschwindigkeit an die Autos übertragen wird bekannt.

6.5.6 Funktionsweise

Um mit dem gewählten Ansatz eine GUI zu realisieren mussten einige Abstriche bei der Visualisierung gemacht werden. Die kann keine fahrenden Autos darstellen. Dies ist Aufgrund des implementierten Simulationsmodells nicht möglich. Die GUI visualisiert die auf der Rennbahn verbauten Lichtschranken und Schleudersensoren. Zusätzlich wird eine Textausgabe für jedes Auto dargestellt, in der die Momentangeschwindigkeit, ein eventueller Schleuderwert, der aktuelle Streckenabschnitt und die momentane Spur angezeigt. Ein Auto das eine Lichtschranke passiert und auslöst, veranlasst die GUI an der Position der Lichtschranke einen grünen Punkt zu zeichnen. Da die Zeit der aktivierten Lichtschranke extrem kurz ist würde man, wenn man den Punkt nur für diese Zeit zeichnen würde nur ein kurzes grünes blinken wahrnehmen. Der Informationsgewinn durch die GUI wäre gleich Null. Deshalb wird der Punkt solange gezeichnet, bis die nächste Lichtschranke ausgelöst wird. Das gleiche Prinzip wird auch bei den Schleudersensoren angewendet. Wenn keine Auslenkung eines Autos messbar ist, so wird nichts gezeichnet. Wenn eine Auslenkung ermittelt wird, so zeichnet die GUI einen roten Punkt, bis das Auto die nächste Lichtschranke auslöst.

6.5.7 Datentrukturen, Dateien & Implementierung

Um die GUI in die Simulation zu integrieren wurde eine neue globale Datenstruktur definiert, auf die alle an der Simulation beteiligten Threads zugreifen können. Das bedeutet, dass die Datenstruktur der Threads um einen Zeiger auf die neue globale Datenstruktur erweitert wurde. Außerdem wurde ein neuer Thread in die Simulation implementiert, dessen einzige Aufgabe im Zeichen der GUI besteht. Aus Gründen der Performance wird der Thread nur alle 10 ms ausgeführt, um nicht zu viel CPU – Zeit zu verbrauchen und das restliche System reaktionsfreudig zu lassen. Der Thread bezieht alle zum Zeichnen benötigten Daten aus der globalen Datenstruktur und die Simulationsthreads füllen einen Teil dieser, wenn ein festes oder ein variables Ereignis eintritt. Dieses Vorgehen verlangt nach Synchronisation der beteiligten Threads und aus diesem Grund wurde ein neuer Mutex eingeführt der diesen Zweck erfüllt.

Die globale Datenstruktur beinhaltet fünf eindimensionale Felder und zwei zweidimensionale Felder. Die zweidimensionalen Felder beinhalten die Pixelkoordinaten der Sensoren. Diese Koordinaten werden zum Zeichnen der Sensorpunkte auf der GUI benötigt. Die Werte für diese Koordinaten liegen in einer XML-Datei. Mittels einer eigenen Funktion werden diese beiden Felder mit den Werten aus der Datei gefüllt. Da es immer eine feste Anzahl an Sensoren gibt und immer ein Sensor auf pro Spur verbaut ist benötigt man zweidimensionale Felder. Für die Schleudersensoren ist die maximale Anzahl an Elementen in der ersten Dimension sieben, da es sieben Abschnitte mit Schleudersensoren gibt und die zweite Dimension vier, da es maximal vier Spuren gibt. Analog verhält es sich bei den Lichtschranken. Die Gesamtanzahl an Lichtschranken beträgt hier zehn. Die Werte wurden durch umrechnen der realen Werte in mm unter Berücksichtigung des Maßstabes des Bitmaps der GUI ermittelt. Für die Koordinaten wurde eine eigene Datenstruktur eingeführt. Die zweidimensionalen Felder sind von dessen Typ.

Aufbau der XML-Datei:

```
<sensorData>
    <light>
        <sensor>
            <Spur1>
                <x>606</x>
                <y>737</y>
            </Spur1>
            <Spur2>
                <x>606</x>
                <y>721</y>
            </Spur2>
            <Spur3>
                <x>606</x>
                <y>702</y>
            </Spur3>
```

```
                <Spur4>
                    <x>606</x>
                    <y>683</y>
                </Spur4>
            </sensor>
            .
            .
        </light>
        <drift>
            <sensor>
                <Spur1>
                    <x>84</x>
                    <y>717</y>
                </Spur1>
                <Spur2>
                    <x>97</x>
                    <y>702</y>
                </Spur2>
                <Spur3>
                    <x>109</x>
                    <y>688</y>
                </Spur3>
                <Spur4>
                    <x>121</x>
                    <y>674</y>
                </Spur4>
            </sensor>
            .
            .
        </drift>
    </sensorData>
```

In einem weiteren Feld der Größe vier werden die aktuellen Geschwindigkeiten jedes Autos gespeichert. Die aktuellen Hardwarestatuswörter werden auch in einem Feld der Größe vier gespeichert. In einem weiteren Feld der Größe vier können die Werte des Schleudersensors für alle vier Spuren gespeichert werden. Die zwei restlichen Felder werden benutzt um Flags zu speichern die dem GUI-Thread signalisieren wann er einen Punkt für den Schleudersensor oder die Lichtschranken zu zeichnen hat. Findet er eine Eins in irgendeinem Feldelement so weiß der Thread, dass ein Punkt zu zeichnen ist. Anhand der Indexe jener Feldelemente in denen die Einsen gespeichert sind, kann der Thread bestimmen welche Sensoren zu Zeichnen sind und sich die entsprechenden Koordinaten aus den zweidimensionalen Feldern beschaffen. In dem GUI-Thread wird nun mittels der Flags gesteuert wann welche Sensoren und welche Werte in der Textausgabe gezeichnet werden. Der jeweilige aktuelle Abschnitt kann über das Feld mit den

Hardwarestatuswörtern ermittelt werden. Um die Effektivität und Geschwindigkeit der GUI zu steigern, werden nur die Bereiche der GUI neu gezeichnet, in denen sich etwas geändert hat.

7. Fazit

Da bei dieser Simulation eine Wiederverwendung des Ansatzes aus der originalen Simulation angestrebt wurde, waren die Möglichkeiten der Schleudersensorsimulation eher beschränkt. Der Ansatz der Simulation hat hier und im Besonderen bei der Erweiterbarkeit um eine GUI große Schwächen. Bei einer erneuten Implementierung einer solchen Simulation würde ich von dem gewählten Ansatz Abstand nehmen und einen kontinuierlichen Simulationsansatz wählen. Mit diesem Ansatz sollte es dann Möglich sein den Gesamtzustand der Simulation und ihrer darin enthaltenen Teilnehmer quasi zu jedem Zeitpunkt zu erfassen. Die Schleudersensoren könnten dann eher wie das ESP an einem Auto realisiert werden. Die jetzt auftretenden Probleme bei der GUI wären auch nicht gegeben, da für das Zeichnen nur die Datenstruktur des Gesamtsystems betrachtet werden müsste und alle benötigten Werte vorhanden wären. Das was mit der jetzigen Simulation im Bezug auf eine GUI machbar war, ist eher unzureichend. Aber aus dem gewählten Ansatz ist dies das Maximum was herauszuholen ist. Da im implementierten Ansatz kein Weltkoordinatensystem definiert ist, welches die Simulation zur Berechnung ihrer Position benötigen würde, war es auch nicht Möglich das Fahren der Autos grafisch darzustellen. Hierfür wäre der kontinuierliche Ansatz besser geeignet. Mit ihm könnte man für kleine Zeitfenster kontinuierlich den Zustand aller an der Simulation beteiligten Sensoren und Autos unter Berücksichtigung alles physikalischen Gegebenheiten berechnen und die Positionen im Weltkoordinatensystem wären zu jedem Zeitpunkt bekannt. Gerade die Ansätze mit Hilfe des dynamischen Einspur- und Zweispurmodell wären in diesem Ansatz leicht zu integrieren gewesen. Da sie nur als Teil der kontinuierlichen physikalischen Berechnung ausgeführt worden wären. Sie hätten nahtlos in das Modell eingefügt werden können. Ich denke aber, dass der beste Weg für die Implementierung einer Schleudersensorsimulation auf einem Ansatz mit neuronalen Netzen beruht. Wobei hier natürlich große Veränderungen am Bahnlayout nicht so einfach zu Bewerkstelligen sind. Man müsste dann das neuronale Netz wieder von neuem trainieren und so auf die neuen Gegebenheiten einstellen. Dies ist bei dem von mir implementierten Ansatz durch eine neue XML-Datei und durch wenige Anpassungen in der Routine zur Bestimmung der Sensorpositionen durchaus in sehr kurzer Zeit möglich.

In diesem Abschnitt soll ein kurzer Überblick über den generellen Ablauf der Simulation gegeben werden, insbesondere welche Schritte beim Start des Programms ausgeführt werden. Nach dem Start der Simulation wird geprüft ob das Programm mit Parametern

gestartet wurde. Ist dies der Fall so werden diese ausgewertet und das Programm mit der entsprechenden Konfiguration gestartet. Werden keine Parameter gestartet, so startet das Programm in der Default-Konfiguration. Im nächsten Schritt werden die Geräte, welche die Spuren eins bis vier darstellen, geöffnet und der entsprechende Filedeskriptor wird der jeweiligen Threadstruktur zugewiesen. Als nächstes werden die Fahrzeugdaten aus den entsprechenden XML-Dateien gelesen, die zugehörigen Strukturen damit initialisiert und benötigte Daten berechnet.

Als nächstes werden die Streckeninformationen aus XML-Dateien eingelesen und die Strukturen, welche die Strecke im Rechner darstellt, werden für jede Spur aufgebaut. Danach werden zwei Threads generiert. Einer zum Lesen der Gerätedatei und einer der auf die Gerätedatei schreibt. Die Vorgänge des Streckenaufbaus und der Initialisierung der beiden Threads werden für jede Spur wiederholt, wenn darauf ein Auto fahren soll.

Anhang

8. Datenstrukturen

8.1 struct carData

```
typedef struct _carData
{
  speed speedMap[0x80];
  int aPositive[6];
  sVAccelerate[6][50][12
  int aNegative;
  vDecelerate sVDecelerate[50][12];
  double width;
  double distance;
  double rail;
  double length;
  double friction_G;
  double friction_R;
  double distance2;
  double mass;
  double angle_offset;
  double t_offset;
  angleRange myAngleRange[14];
}carData;
```

8.2 struct angleRange

```
typedef struct _angleRange
{
  int startAngle;
  int endAngle;
}angleRange;
```

8.3 struct vAccelerate

```
typedef struct _vAccelerate
{
	int realPositive;
	int hexPositive;
	int realNegative;
	int hexNegative;
}vAccelerate;
```

8.4 struct vDecelerate

```
typedef struct _vDecelerate
{
	int real;
	int hex;
}vDecelerate;
```

8.5 struct speed

```
typedef struct _speed
{
	int real;
	sSensorData tracks[4];
}speed;
```

8.6 struct sSensorData

```
typedef struct _sensorData
{
	int seg_210_E;
	int seg_120;
	int seg_360;
	int seg_240;
	int seg_180;
	int seg_120_E;
	int seg_210;
}sSensorData;
```

8.7 struct sTrackData

```
typedef struct _Trackdata
{
	char *name;
	int id;
	int track[4];
	int s[4];
	int v[4];
	int r[4];
	int sK[4];
	double alpha;
	double sAlpha;
	int direction;
	int slide;
} sTrackData;
```

8.8 struct sSegData

```
typedef struct _SegData
{
	int   length;
	int   line;
	int dangerAreaStart;
	int dangerAreaEnd;
	char name[20];
	sTrackData *trackData;
	sSlideData *slideData;
	__u32 hw_status;
}sSegData;
```

8.9 struct sSlideData

```
typedef struct _slideData
{
	int slidePos;
	int slideSensor;
}sSlideData;
```

8.10 struct threadParam

```
typedef struct _ThreadParam
{
	int fp;
	int line;
	int own;
	int carNr;
	int* enemyLine;
	__u32 value;
	__u32 breaking;
	pthread_t readthread;
	pthread_t writethread;
	pthread_mutex_t speedMut;
	pthread_cond_t speedCond;
	guiData* guiData_ptr;
	C_List segments;
	carData sCarData;
} threadParam;
```

8.11 struct coordinate

```
typedef struct _coordiate
{
	int x;
	int y;
}coordinate;
```

8.12 struct guiData

```
typedef struct _guiData{

	coordinate lBCoordinates[10][4];
	coordinate dSCoordinates[7][4];
	int speeds[4];
	int slides[4];
	__u32 HwStatuswords[4];
	int draw[4];
	int red[4];
}guiData;
```

Literaturverzeichnis

[Rob04]. **Robert Bosch GmbH.** *BOSCH Automotive Handbook.* s.l. : John Wiley and Sons, 2004.

[Ass04]. **Assmann, Bruno.** *Technische Mechanik. 3. Kinematik und Kinetik.* s.l. : Wissenschaftsverlag, 2004.

[Bec01]. **Beckmann, Brian.** *Physics of Racecars.* 2001.

[Bit07]. **Bittel, Oliver.** *NeuroNetze_4.pdf.* 2007.

[DoKrVo07]. **Dobrinski, Paul, Krakau, Gunter und Vogel, Anselm.** *Physik für Ingenieure.* s.l. : Vieweg+Teubner Verlag, 2007.

[Gen97]. **Genta, Giancarlo.** *Motor Vehicle Dynamics: Modeling and Simulation.* s.l. : World Scientific, 1997.

[HaHo03]. **Halfmann, Christoph und Holzmann, Henning.** *Adaptive Modelle für die Kraftfahrzeugdynamik.* s.l. : Springer, 2003.

[HeEr07]. **Heißing, Bernd und Ersoy, Metin.** *Fahrwerkhandbuch: Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik.* s.l. : Vieweg Verlag, 2007.

[Her04]. **Hermann und Martin.** *Numerik gewöhnlicher Differentialgleichungen: Anfangs- und Randwertprobleme.* s.l. : Oldenbourg Wissenschaftsverlag, 2004.

[www_ika]. **ika-achen.** *http://www.ika.rwth-aachen.de/lehre/kfz-labor/6_fahrdynamik_de.pdf.*

[Ise06]. **Isermann, Rolf.** *Fahrdynamikregelung.* s.l. : Vieweg Verlag, 2006.

[JudGer05]. **Judy Hsu, Yung-Hsiang und Gerdes, J. Christian.** *Stability at the Limits PowerPoint Präsentation.* s.l. : Stanford University, 2005.

[MiWa04]. **Mitschke, Manfred und Wallentowitz, Henning.** *Dynamik der Kraftfahrzeuge.* s.l. : Springer, 2004.

[Mar]. **Monster, Marco.** *Physics Tutorial.*

[RieSchu39]. **Riekert, P und E., Schunk T.** *Zur Fahrmechanik des gummibereiften Kraftfahrzeugs .* s.l. : Springer Berlin / Heidelberg, 1939.

[RiFiSchFu05]. **Rittenschober, Thomas, et al.** *Fahrdynamikregelung mit differentialgeometrischen Methoden der Regelungstechnik.* s.l. : Institut für Regelungstechnik und Prozessautomatisierung, Johannes Kepler Universität Linz, 2005.

[www_erl]. **uni-erlangen.** *http://www.tp2.uni-erlangen.de/Lehrveranstaltungen/medi/Medi_Materialien/W-G-B_Zeit_Diagramme.pdf.*

[www_wü]. **uni-würzburg.** *http://www.physik.uni-wuerzburg.de/physikonline/repetmech/zusammenfassungen/zudynamik1p1.html.*

[SSV98]. **Vary, Peter, Heute, Ulrich und Hess, Wolfgang.** *Digitale Sprachsignalverarbeitung.* Stuttgart : Teubner, 1998.

[Wal96]. **Wallentowitz, H.** *Vertikal-/Querdynamik von Kraftfahrzeugen.* s.l. : Umdruck zur Vorlesung, Aachen, 1996.

Ehrenwörtliche Erklärung

Hiermit erkläre Ich, Marco Fahr, geboren am 07.03.1976 in Stockach (DE),

(1) dass Ich meine Diplomarbeit mit dem Titel:

Carrerabahn - Implementierung einer Simulation zur Verifizierung von Echtzeitprogrammen für die Steuerung von Hardware

An der HTWG Konstanz unter Anleitung von Professor Dr. M. Mächtel selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angeführten Hilfen benutzt habe;

(2) dass Ich die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Diese Arbeit hat in dieser oder ähnlicher Form noch nicht im Rahmen einer anderen Prüfung oder an einem anderen Ort vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 09.07.2008

Danksagung

Schließlich möchte Ich mich herzlich bedanken, bei allen, die mir für die Erstellung dieser Arbeit den Rücken freigehalten haben.

Besonders Danken möchte ich hierbei Herrn Prof. Dr. M. Mächtel und der gesamten Fakultät IN, welche durch die technische Ausrüstung die Erstellung dieser Arbeit ermöglicht haben.

Meinen Eltern und Jasmina danke Ich für die Unterstützung während des gesamten Studiums.

Außerdem danke Ich allen, die mich während der Entstehungszeit dieser Arbeit Motiviert, Unterstützt oder Versorgt haben.

Für Jerome