

Bachelorarbeit

Kordian Kubat

Eine hierarchische Steuerungsarchitektur mit Fuzzy-Regelung
zur positionsmarkenbasierten Navigation eines autonomen
Modellfahrzeugs

Kordian Kubat

**Eine hierarchische Steuerungsarchitektur mit Fuzzy-Regelung
zur positionsmarkenbasierten Navigation eines autonomen
Modellfahrzeugs**

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Abgegeben am 28. September 2007

Kordian Kubat

Thema der Bachelorarbeit

Eine hierarchische Steuerungsarchitektur mit Fuzzy-Regelung zur positionsmarkenbasierten Navigation eines autonomen Modellfahrzeugs

Stichworte

Steuerungsarchitektur, autonome Steuerung, Fuzzy-Regelung, autonome Fahrzeuge

Kurzzusammenfassung

Diese Bachelorarbeit befaßt sich mit der Entwicklung und Implementierung einer Steuerungsarchitektur und einer Fuzzy-Regelung zur positionsmarkenbasierten Navigation eines autonomen Modellfahrzeugs. Inhaltlich wird auf Themen der Steuerungsarchitekturen, sowie auf Aspekte der Fuzzy-Regelung eingegangen. Aufbauend darauf werden das Konzept und die Anwendung vorgestellt, welche im Rahmen dieser Arbeit für die autonome Steuerung des Fahrzeuges entwickelt wurden.

Kordian Kubat

Title of the paper

A Hierarchical Control Architecture with Fuzzy Control System for Landmark-Based Navigation of an Autonomous Model Vehicle

Keywords

Control architecture, autonomous control, Fuzzy-Control, autonomous vehicle

Abstract

This thesis (Bachelor) deals with the development and implementation of a control architecture and a fuzzy control system for landmark-based navigation of an autonomous model vehicle. One purpose of this work was to develop an architecture of an autonomous control system. A special consideration was to the development of a fuzzy logic control system which presents a method to control a vehicle towards a landmark.

Danksagung

Einen ganz besonderen Dank möchte ich an dieser Stelle an meinen betreuenden Professor Dr. Stephan Pareigis richten, der mir dieses interessante Thema zur Verfügung stellte und während unserer Gespräche immer wieder inspirierende Ideen und Vorschläge äußerte.

Ganz herzlichen Dank auch an Herrn Professor Dr. Andreas Meisel der mir als weiterer Betreuer und Zweitprüfer zur Seite stand. Ebenso einen Dank an Herrn Professor Dr. Franz Korf und den Assistenten des 7. Stocks der HAW Hamburg. Ihre Unterstützung half in so mancher schwierigen Situation neue Gedanken und Inspirationen zu erhalten.

Weiterhin möchte ich mich bei dem Team intelliTruck für die Unterstützung und die gute Zusammenarbeit bedanken. Nicht nur im alltäglichen Laborbetrieb, auch in der Nacht-des-Wissens hat mir die Zusammenarbeit sehr viel Freude bereitet.

An dieser Stelle möchte mich ganz besonders bei zwei Studienkollegen und Teammitgliedern bedanken, die mich in dieser Zeit begleitet haben. Einen Dank an Enrico Hensel und Sven Geibert.

Eine Gruppe von Personen die mir besonders am Herzen liegt, möchte ich an dieser Stelle ebenfalls erwähnen und meinen außerordentlichen Dank aussprechen, meiner Familie. Danke für die Unterstützung während meiner Studienzzeit.

Zu guter Letzt, jedoch mit größtem persönlichem Anliegen, möchte ich meiner Frau danken. Ich danke Dir, dass Du mich während meiner Abschlussarbeit so aufopferungsvoll unterstützt und begleitet hast. Danke für die Motivation und Stärke die ich durch Dich erhalten habe, danke für dein Verständnis, danke für dein Dasein!

Inhaltsverzeichnis

Inhaltsverzeichnis	1
Abbildungsverzeichnis.....	3
Tabellenverzeichnis	4
1 Einleitung	5
1.1 Ausgangssituation	5
1.2 Das Modellfahrzeug - „intelliTruck“	5
1.3 Aufgabenstellung der Bachelor-Arbeit	6
1.4 Vorgehen und Inhalte	6
2 Architekturen für autonome Systeme	7
2.1 Funktionsorientierte, hierarchische Architekturen.....	7
2.2 Verhaltesbasierte, reaktive Architekturen	9
3 Realisierte Steuerungsarchitektur	11
3.1 Begriffsbildung	11
3.2 Hierarchische Struktur	12
3.3 Reaktive Steuerung	13
3.3.1 Zustandsreflektor	14
3.3.2 Interpreter.....	14
3.3.3 Navigator	14
3.3.4 Fähigkeiten.....	14
3.4 Schnittstellenkomponenten	15
3.4.1 Abstrakte Treiber	15
3.4.2 CAN-Transceiver.....	16
3.4.3 Abstrakte Aktorenschnittstelle.....	16
3.5 Implementierung	16
3.5.1 Sensordatenerfassung.....	16
3.5.2 Ereignisverarbeitung.....	17
3.5.3 Koordination	18
3.5.4 CAN-Bus	20
3.6 Einordnung.....	20
4 Positionsmarkenverfolgung mit Fuzzy-Regeln.....	22
4.1 Einführung in die Fuzzy-Methodiken	22

4.1.1	Fuzzy-Mengen und linguistische Terme.....	22
4.1.2	Operatoren auf Fuzzy-Mengen	23
4.1.3	Fuzzy-Implikation und Fuzzy-Inferenz	24
4.2	Konzeption des Fuzzy-Positionsmarkenreglers	26
4.2.1	Aufbau	26
4.2.2	Grundlage der Regelbasis	26
4.3	Konkrete Umsetzung.....	28
4.3.1	Längsregelung.....	28
4.3.2	Querregelung	30
4.3.3	Implementation	31
4.4	Simulator	31
4.5	Simulationsergebnisse.....	33
4.6	Zusammenfassung der Simulationsergebnisse.....	38
5	Praktische Anwendung der Architektur	39
5.1	Beschreibung.....	39
5.2	Simulation	40
5.3	Testszenarien.....	42
6	Zusammenfassung	43
	Literaturverzeichnis	44
	Anhang A: Regelsätze der Fuzzy-Reglung	46
A.1	Regelsatz der Längsregelung.....	46
A.2	Regelsatz der Querregelung	46

Abbildungsverzeichnis

Abbildung 1: „intelliTruck“ – Die autonome Testplattform	5
Abbildung 2: Horizontale Dekomposition [Bro86]	7
Abbildung 3: Shakey – Roboter mit künstlicher Intelligenz	8
Abbildung 4: Vertikale Dekomposition [Bro86]	9
Abbildung 5: Genghis - Sechsbeiniger Roboter	10
Abbildung 6: Positionsmarkenbasierte Fahrt.....	11
Abbildung 7: Schematische Darstellung der hierarchischen Struktur der Steuerung.....	12
Abbildung 8: Schematische Darstellung des reaktiven Steuerungszyklus	13
Abbildung 9: Schematische Darstellung der unterschiedlichen Schichten des Gesamtsystems.....	15
Abbildung 10: Ereignisse.....	18
Abbildung 11: Von <i>ISensorListener</i> abgeleitete Klassen	18
Abbildung 12: Klassendiagramm - Darstellung der implementierten Fähigkeiten und des Navigators im Koordinator	19
Abbildung 13: Fuzzy-Mengen am Beispiel der linguistischen Variable X: „Geschwindigkeit“	23
Abbildung 14: Links: Schnitt zweier Fuzzy-Mengen (ODER-Operator). Rechts: Vereinigung zweier Fuzzy-Mengen (UND-Operator).....	24
Abbildung 15:Fuzzy-Inferenz mit zwei Regeln über einem Eingangsbereich X mit einem Ergebnisbereich Y.....	25
Abbildung 16: Fuzzy-Regelung zur Positionsmarkenverfolgung	26
Abbildung 17: Eingänge der Fuzzy-Regelung.....	27
Abbildung 18: Schematische Darstellung eines Fuzzy-Reglers mit zwei Eingangsgrößen, einer Ausgangsgröße.....	28
Abbildung 19: GUI –Simulator: Oben links: Hauptbedienfeld. Unten links: Kameraeinstellungen. Rechts: virtuelle Karte	32
Abbildung 20: Das Vierrad-Modell.....	33
Abbildung 21: Fahrverhalten in Abhängigkeit von der Abtastzeiten der Kamera	34
Abbildung 22: Längsreglung bei unterschiedlichen Abtastzeiten der Kamera	35
Abbildung 23: Querreglung bei unterschiedlichen Abtastzeiten der Kamera	35
Abbildung 24: Simulation mit schiefer Kamera	36
Abbildung 25: Längs- und Querreglung bei versetzter Kamera um 20 Grad.....	37
Abbildung 26: Flussdiagramm der Fähigkeitsselektion	40
Abbildung 27: Simulation der Anwendung.....	41

Tabellenverzeichnis

Tabelle 1: Basisklassen-Übersicht	16
Tabelle 2: Implementierte Sensoren	17
Tabelle 3: Die schematischen Fähigkeiten	39
Tabelle 4: Schematische Darstellung des Interpreters	39

1 Einleitung

1.1 Ausgangssituation

Diese Bachelor-Arbeit wurde im Rahmen des „intelliTruck“-Projektes eingerichtet, eines Entwicklungsprojektes aus dem Bereich Fahrerassistenz- und Autonome Systeme (FAUST) der HAW-Hamburg. Dabei soll ein Modellfahrzeug, anhand von Positionsmarken, einen Parcours im freien Gelände autonom¹ abfahren. Zu diesem Zweck sind eine ausgeklügelte Bildverarbeitung, zuverlässige Sensorik und ein Sicherheitskonzept erforderlich. Die dafür notwendigen Systemkomponenten werden durch andere Projektmitglieder, weitestgehend Parallel zu dieser Arbeit entwickelt. Diese und bereits abgeschlossene Arbeiten sind grundlegend für die vorliegende Arbeit.

Das Projekt steht noch am Anfang und es wurden noch keine Versuche zuvor unternommen das Modellfahrzeug eigenständig fahren zu lassen. Viele Aufbauarbeiten am Fahrzeug sind noch durchzuführen. Eine enge Zusammenarbeit des Teams ist notwendig um die Herausforderung einer autonomen Fahrt zu bewältigen.

1.2 Das Modellfahrzeug - „intelliTruck“

Das Modellfahrzeug soll kurz vorgestellt werden. Es handelt sich um ein Modellfahrzeug eines Geländewagens im Maßstab 1:6 mit einem 2 kW starkem Verbrennungsmotor. Damit ist das Fahrzeug in der Lage eine Geschwindigkeit von bis zu 65km/h zu erreichen. Dabei steuert die Funkanlage mit den sogenannten PWM-Signalen zwei Stellmotoren (kurz Servos). Der eine Servo wird für die Lenkung, der andere für Gas und Bremse eingesetzt.



Abbildung 1: „intelliTruck“ – Die autonome Testplattform

¹ autonom: selbständig, eigengesetzlich

Das Modellfahrzeug wurde mit Hall- und Beschleunigungssensoren ausgerüstet. Ein Gyroskop soll zusätzliche Informationen über die Bewegung des Fahrzeugs liefern. Dabei werden Microcontroller (AVR-Einheiten) für die Vorverarbeitung der Sensorensignale eingesetzt. Eine am Fahrzeug angebrachte Kamera soll die Positionsmarkenerkennung möglich machen. Desweiteren befindet sich ein Hauptrechner an Board, der über den CAN-Bus mit den AVR-Einheiten verbunden ist und die Gesamtsteuerung übernehmen soll. Eine Sicherheitsschaltung ermöglicht das Unterbrechen einer autonomen Fahrt im Fehlerfall. Diese wurde in [EH07] entwickelt.

1.3 Aufgabenstellung der Bachelor-Arbeit

Die Zielstellung dieser Arbeit besteht darin eine Steuerungsarchitektur zu realisieren, die ein autonomes Agieren des Modelfahrzeuges ermöglicht. Die Steuerungsarchitektur soll dabei einerseits die Kommunikation mit den Hardware-Komponenten übernehmen, andererseits in ihrer Funktionsweise positionsmarkenbasiertes Navigieren ermöglichen und soll als Grundlage für Weiterentwicklungen genutzt werden können. Ein modularer Aufbau, sowie einfache Erweiterungs- und Anwendungsmöglichkeiten der Architektur werden gefordert. Ein weiteres Ziel der Arbeit besteht darin eine konkrete Anwendung zu entwickeln, um eine positionsmarkenbasierte, autonome Fahrt durchzuführen und damit die Praxiseignung des Gesamtsystems zu beweisen.

1.4 Vorgehen und Inhalte

In **Kapitel 2** werden zunächst zwei grundlegende Architekturen zur Steuerung autonomer mobiler Roboter vorgestellt, die in der Fachliteratur bekannt sind.

Kapitel 3 stellt die Konzepte und die Komponenten der realisierten Steuerungsarchitektur vor.

Kapitel 4 beschäftigt sich mit der Aufgabe, wie eine autonome Positionsmarkenverfolgung des Fahrzeugs erreicht werden kann und stellt eine Fuzzy-Regelung als Lösung vor.

In **Kapitel 5** wird eine Anwendung, basierend auf den Komponenten der realisierten Steuerungsarchitektur, vorgestellt. Die Fuzzy-Regelung aus dem vorangehenden Kapitel wird dabei benutzt um das Fahrzeug in die Nähe einer Positionsmarke zu steuern.

In **Kapitel 6** wird in der Zusammenfassung noch mal im Einzelnen auf die realisierten Steuerungsarchitektur und Fuzzy-Regelung eingegangen und deren Testergebnisse zusammenfassend dargestellt.

2 Architekturen für autonome Systeme

In der Literatur [Ark98], [Kni], [JonFly] werden zwei grundlegende Architekturen für autonome System beschrieben. Beide Architekturen versuchen das relative komplexe Gesamtproblem einer autonomen Steuerung (z.B. viele Sensordaten, die sich evtl. widersprechen, Priorisierung der Aufgaben, Echtzeitbedingungen) durch Zerteilung in kleinere Problemstellungen zu bewältigen und unterteilen dazu das Gesamtsystem in Funktionseinheiten bzw. Komponenten mit unterschiedlicher Zuständigkeit. Die Abgrenzung der beiden Steuerungsarchitekturen erfolgt durch Unterschiede im Zusammenspiel dieser Komponenten.

2.1 Funktionsorientierte, hierarchische Architekturen

Die funktionsbasierte Architektur ist die klassische Art von Steuerungssystemen, bei der das Steuerungsproblem in eine Reihe von eigenständigen Funktionseinheiten zerlegt wird. Jede dieser Einheiten bearbeitet eine gewisse Aufgabe wie Wahrnehmung, Planung oder Aktion. Das Gesamtverhalten des Systems entsteht dadurch, dass jedes dieser Module zeitlich nacheinander durchlaufen wird. Die Wahrnehmung liefert eine abstrakte, symbolische Beschreibung der Welt (oder eines bestimmten Aspektes davon) und die Motorkontrolle setzt symbolische Anweisungen in reale Aktionen um. Dazwischen verarbeitet eine Zentrale Einheit diese abstrakten Informationen, welche in der Tradition der klassischen KI, Pläne generiert. Dieser Ansatz wird nach [Bro86] auch als horizontale Dekomposition bezeichnet.



Abbildung 2: Horizontale Dekomposition [Bro86]

Im Falle des Versagens eines einzelnen Moduls versagt auch das gesamte System, wenn nicht spezielle Vorkehrungen getroffen wurden. Die Reaktionszeit eines solchen System ist aufgrund der streng sequentiellen Ausführung der einzelnen Verarbeitungsschritte relativ lang, sobald die Abbildung der Daten auf das Weltmodell nicht trivial ist und die Planungseinheit einen großen Hypothesenraum durchsuchen muss.

Shakey, der erste Roboter mit künstlicher Intelligenz arbeitete auf diese Weise. Er wurde ab 1967 am Stanford Research Institute (SRI) entwickelt. Das Weltmodell wurde dabei unter der Annahme der Weltabgeschlossenheit gebildet. Daher man ging davon aus, mit einer Menge von vorgegebenen Fakten und Regeln sowie den jeweiligen Sensordaten ein vollständiges Modell der Umwelt erfassen zu können.

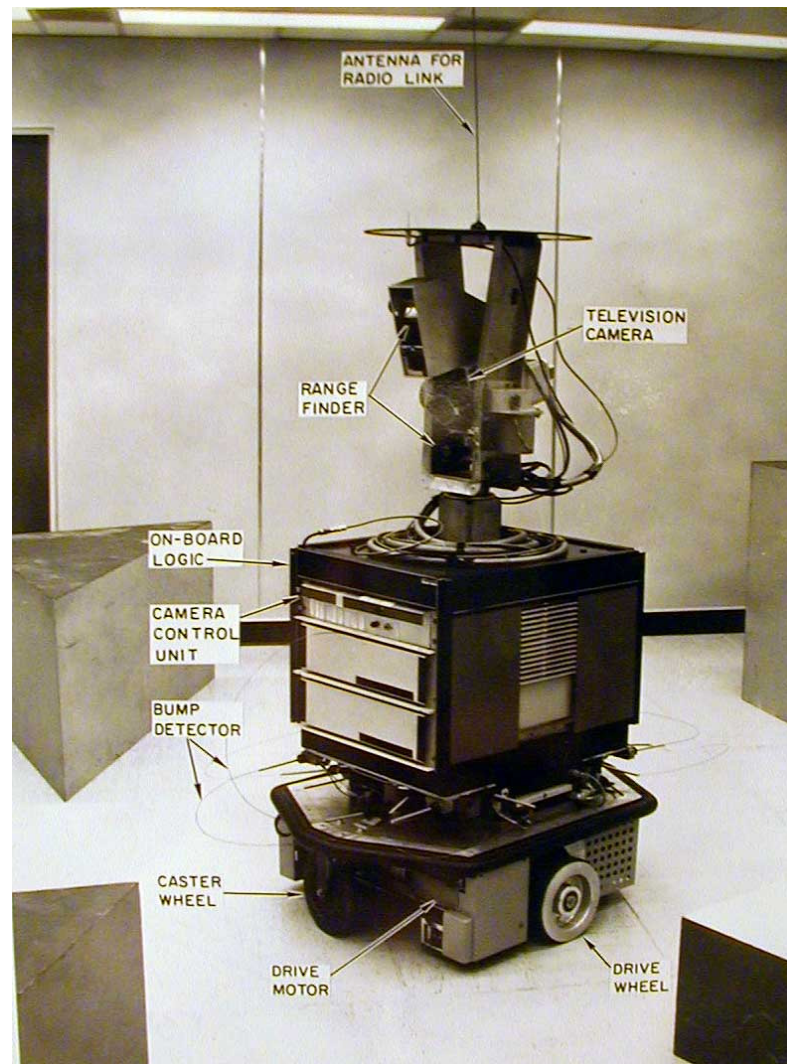


Abbildung 3: Shakey – Roboter mit künstlicher Intelligenz

Neben den Problemen, die sich beim erstellen des Weltmodells ergaben, war ein wesentlicher Nachteil dieses Vorgehens, dass die ständige Neuberechnung des Plans keine flüssige Bewegung zu Stande kommen ließ.

Der wichtigste Vorteil dieser Architekturart ist das Vorhandensein einer globalen Sicht, aufgrund derer eine komplexe Planung durchgeführt werden kann. Diese Planung ermittelt eine Sequenz von Aktionen, die sicher zum Ziel führt oder man kann ggf. schon von Beginn an zeigen, dass die Aufgabe unlösbar ist. So können z.B. lokale Sackgassen, unter Ausnutzung von KI-Methoden erkannt und ein Umwegplan ermittelt werden. Zudem führt die Zerteilung der Architektur in Funktionskomponenten, die eigenständige Teilaufgaben übernehmen, zu einem modularen und gut überschaubaren Systemaufbau.

Die wesentlichen Eigenschaften funktionsorientierter Architekturen nach [Ark98]:

- Sie besitzen hierarchische Strukturen mit klar identifizierbaren, funktionalen Einheiten.
- Sie sind stark abhängig von einer symbolischen Repräsentation der Welt.

- Der Kontrollfluss zwischen den Hierarchieebenen und zwischen den Einheiten ist klar definiert. Höhere Hierarchieebenen geben den tieferen Ebenen Teilziele vor.
- Die Hierarchieebenen besitzen unterschiedliche zeitliche und räumliche Planungshorizonte.

2.2 Verhaltesbasierte, reaktive Architekturen

Mitte der 80er Jahre wurde damit begonnen reaktive Systeme zu bauen. Bei diesem Vorgehen wurde die Planungskomponente komplett aus der Architektur entfernt und die Komponenten der Wahrnehmung und Motorkontrolle direkt verbunden. Jede Aufnahme von Sensordaten führt damit zu einer Aktion bzw. Reaktion. Die Wahrnehmung eines Hindernisses kann beispielsweise direkt eine Ausweichbewegung einleiten.

Im Gegensatz zur funktionalen Zerlegung eines Problems wird bei den Verhaltensbasierten Architekturen gemäß [Bro86] eine Zerlegung nach Aktionen angestrebt. Es werden selbständig agierende Verhaltensmodule erzeugt, die jeweils Zugriff aus sämtliche Sensordaten haben und alle Motoren ansteuern können.

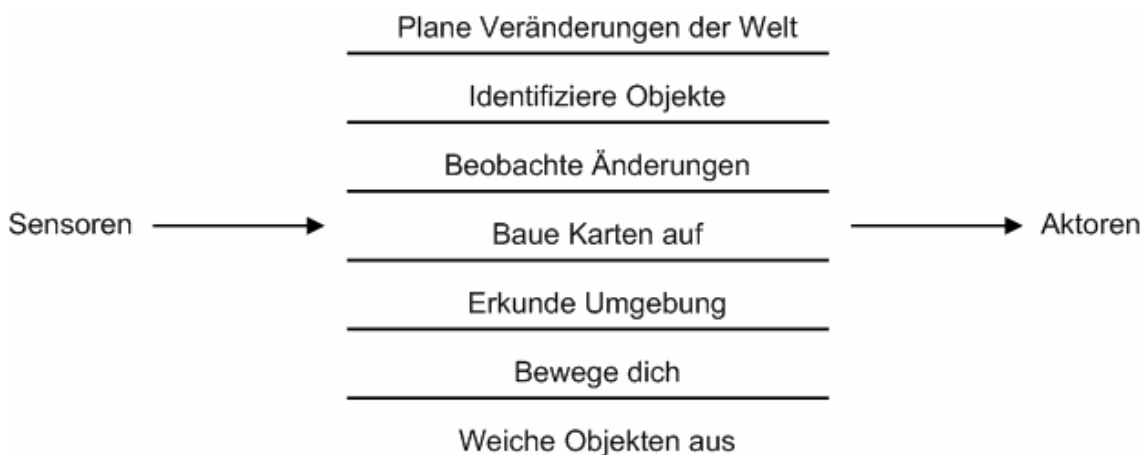


Abbildung 4: Vertikale Dekomposition [Bro86]

Wahrnehmung und Aktion sind in sogenannten Verhalten miteinander verbunden, die über eine enge Kopplung von Wahrnehmung und Aktion verfügen, wobei die Kopplungen von speziellen Teilzielen bestimmt werden. Der Abstraktionsgrad der zu erfüllenden Aufgabe steigt von unten nach oben an. Die Idee ist, durch Überlagerung einfacher Verhalten eine Steuerung für komplexe Aufgaben zu entwickeln. Dabei sollen höher gestellten Verhalten die Aktionen unterstellter Verhalten unterdrücken können. Es sollte möglich sein, die Architektur an einer beliebigen Stelle zwischen zwei Ebenen zu trennen und den unteren Teil als eigenständige Steuerungsarchitektur zu verwenden.

Verhaltensbasierte Ansätze erfuhren einen großen Aufschwung als Brooks 1986 seinen Artikel "A robust layered control system for a mobile robot" [Bro86] veröffentlichte und gleichzeitig demonstrierte, dass dieser Ansatz Überzeugende Ergebnisse auf dem Gebiet der robusten Steuerung von sechsbeinigen Robotern zeigte [Bro89].

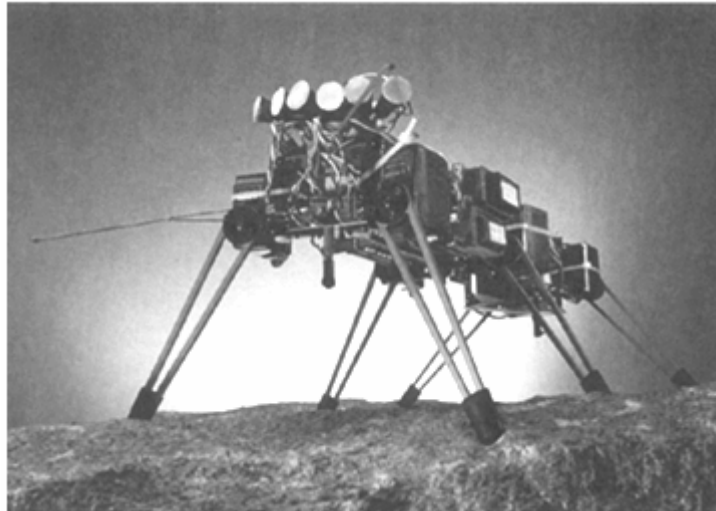


Abbildung 5: Genghis - Sechsbeiniger Roboter

Dies war deshalb erstaunlich, weil vorhergehende Ansätze zur Steuerung sechsbeiniger Roboter, die versuchten die Bewegung aller sechs Beine zentral zu steuern, auf erhebliche Probleme gestoßen waren.

Die Vorgehensweise in der unter den Namen „Subsumption Architecture“ bekannten reaktiven Steuerungsarchitektur von Rodney Brooks bringt spezifische Probleme mit sich. Es gibt keine klare Methodik, wie eine verhaltensbasierte Zerlegung eines Problems tatsächlich anzugehen ist. Ein Problem kann auf sehr unterschiedliche Weise erfolgreich modelliert werden und häufig entsprechen erfolgreiche Zerlegungen nicht der intuitiven Vorstellung. Zudem garantieren pure Reaktionen nicht, dass das autonome System jemals das Ziel erreicht.

Ein großer Vorteil verhaltensbasierter Systeme ist ihre Reaktivität und Robustheit. Da alle Verhalten die Sensoren und Aktoren des Systems bedienen, führt ein Ausfall eines Verhaltens lediglich zu einer Leistungsminderung. Im Vergleich zu den funktionalen liegen die Vorteile verhaltensbasierter Architekturen in dynamischen und unbekanntem Umgebungen, wo schnell auf unerwartete Ereignisse reagiert werden muss.

Die wesentlichen Eigenschaften verhaltensbasierter Architekturen nach [Ark98]:

- Wahrnehmung und Aktion sind in sogenannten Verhalten miteinander verbunden.
- Man verzichtet bewusst auf eine explizite Repräsentation von räumlichen und zeitlichen Aspekten.
- Das Verhaltensbasierte System verfügt über einen Satz von Verhalten mit geringer Komplexität, die auch gleichzeitig aktiv sein können.

3 Realisierte Steuerungsarchitektur

In diesem Kapitel soll die im Rahmen dieser Arbeit entwickelte Steuerungsarchitektur für den „intelliTruck“ vorgestellt werden. Die Architektur und die entwickelten Systemkomponenten bieten ein Rahmenwerk um Anwendungen zur Steuerung des mobilen Fahrzeuges zu entwickeln. Basierend auf der zugrundeliegenden System-Hardware sind Software-Komponenten für die Sensordatenverarbeitung, Navigation, Steuerung und Regelung realisiert worden, die in einer funktionalen Beziehung zueinander stehen. Ein Schnittstellenkonzept verbindet dabei die Hardware- mit der Steuerungsebene.

3.1 Begriffsbildung

Das autonome Fahrzeug soll in einem freien Gelände positionsmarkenbasiert Navigieren. In Abbildung 4 ist ein mögliches Beispielszenario dargestellt.

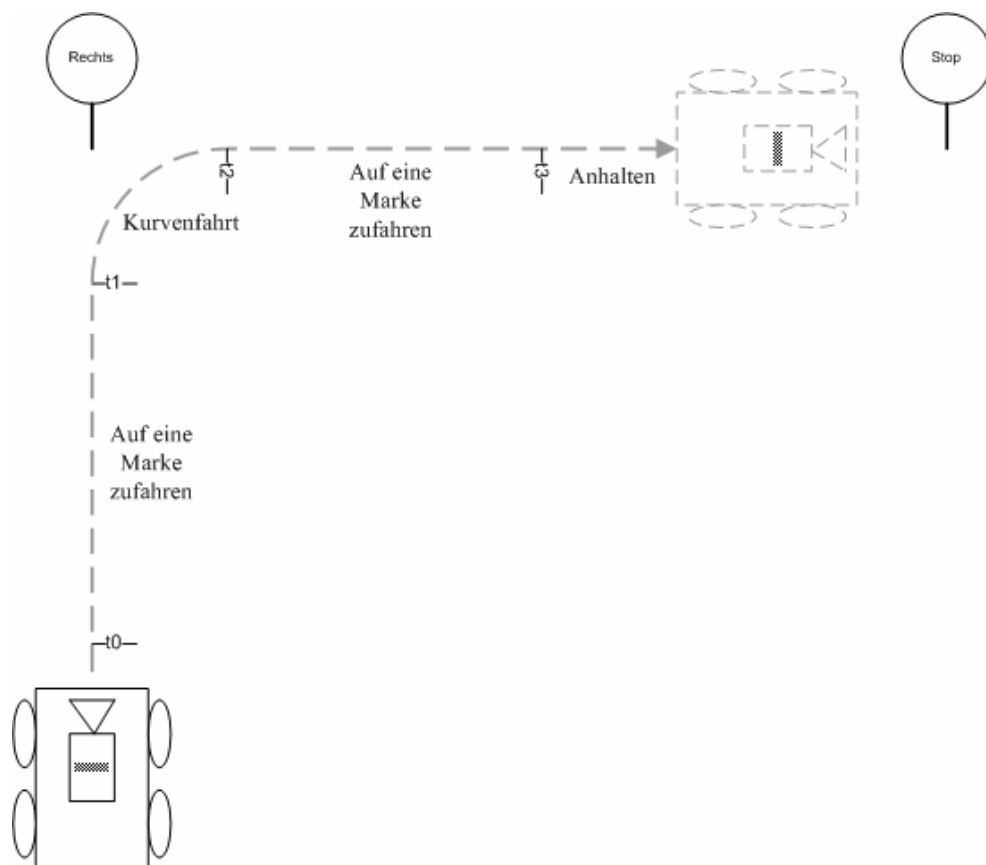


Abbildung 6: Positionsmarkenbasierte Fahrt

Die Gesamtaufgabe einer solchen Fahrt kann in Teilaufgaben unterteilt werden (z.B. die Durchführung einer Kurvenfahrt oder das Anhalten). Diese Teilaufgaben stellen unterschiedliche Bedingungen an die Steuerung und Regelung des Fahrzeuges und werden in

der realisierten Architektur getrennt modelliert. Die dafür realisierten Systemkomponenten heißen **Fähigkeiten**.

Die richtige Sequenz der Fähigkeiten ist entscheidend um den Erfolg einer vergleichbaren Fahrt zu sichern. Dabei spielen Situationsaspekte wie z.B. der aktuelle Abstand zur erkannten Marke oder die momentane Geschwindigkeit eine entscheidende Rolle um eine adäquate Koordination der Fähigkeiten zu gewährleisten. Die dafür realisierte Systemkomponente heißt **Navigator**.

3.2 Hierarchische Struktur

Die realisierte Architektur entspricht in ihrer Struktur dem Funktionsorientiertem bzw. Hierarchischem Ansatz aus Kapitel 2, und kann auf diese Abgebildet werden. Die Aufgabe der Erfassung und Interpretation der Sensordaten übernimmt der Zustandsreflektor [ARIA]. Die Planungskomponente dieser Architektur ist der Navigator. Die Fähigkeiten übernehmen die Funktion der Steuerung und Regelung der Aktoren. Ein schematische Darstellung der hierarchischen Struktur der Steuerung wird in Abbildung 7 dargestellt.

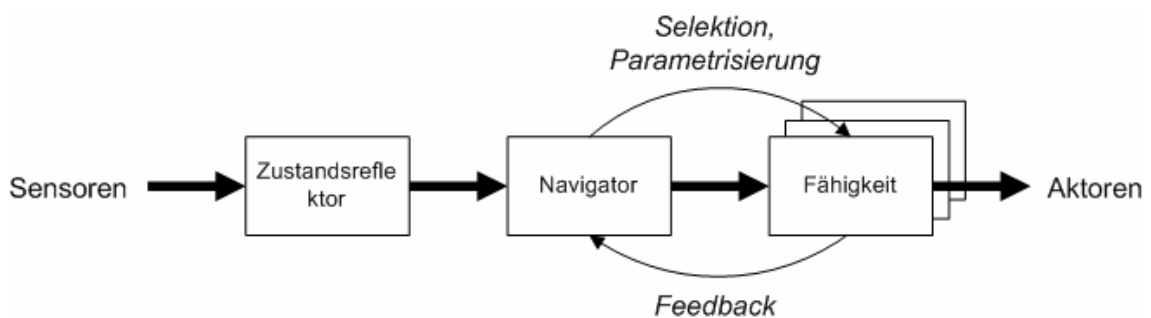


Abbildung 7: Schematische Darstellung der hierarchischen Struktur der Steuerung

Im Gegensatz zu einem rein hierarchischen Ansatz bekommen Fähigkeiten keine Steuerbefehle von dem Navigator, sondern werden mit den Sensoren verbunden. Dabei entscheidet der übergeordnete Navigator welche Fähigkeit mit den Sensoren verbunden wird und damit die Kontrolle über die Aktoren hat. Durch diese Kopplung können die Fähigkeiten in dieser Architektur auch wie Verhalten verstanden werden.

Die Mechanismen für die Parametrisierung und Feedback erlauben zudem das modellieren der Fähigkeiten in Form von abgeschlossenen Aktionen. Diese Möglichkeit kann z.B. für eine Fähigkeit der Kurvenfahrt genutzt werden. Anstatt für jede erdenkliche Kurvenfahrt (z.B. 90-Grad-Rechtskurve oder 45-Grad-Linkskurve) eine eigene Fähigkeit anzulegen, bedarf es nur einer parametrisierbaren Fähigkeit. Die konkreten Kurvenparameter würde in dem Fall der Navigator bestimmen und vor der Aktivierung an die Fähigkeit übergeben. Der Feedback-Mechanismus kann dann dazu verwendet werden, dem Navigator die Fertigstellung der Kurvenfahrt mitzuteilen.

3.3 Reaktive Steuerung

Die reaktive Steuerung erfolgt durch den in Abbildung 8 beschriebenen Zyklus. Die Sensordaten gelangen durch eine Schnittstelle in das Steuerungssystem und lösen den Steuerungsprozess aus.

Bevor die Daten weiterverarbeitet werden, werden diese im Zustandsreflektor gespeichert. Der Zustandsreflektor stellt die unbearbeiteten Sensordaten den Komponenten zur Sensordateninterpretation zur Verfügung. Diese Komponenten werden in dieser Arbeit Interpretierer genannt und können durch Sensorfusion bzw. Interpretation aus den Rohdaten abstrakte Sensordaten erstellen. Nachdem die Interpretation der Sensordaten abgeschlossen ist, löst der Zustandsreflektor ein Ereignis im Navigator aus.

Der Navigator kann das Ereignis einem bestimmten, im System verfügbaren Sensor zuordnen und bekommt gleichzeitig direkten Zugriff auf die dazugehörigen Daten. Situationsbedingt wird eine geeignete Fähigkeit selektiert, anschließend das Ereignis an die Fähigkeit weitergereicht. Dabei werden Ereignisse, die für die Navigation unnötig sind, an die zum aktuellen Zeitpunkt selektierte Fähigkeit weitergereicht ohne eine Verarbeitung im Navigator auszulösen.

Das Ereignis erreicht nun die selektierte Fähigkeit. Fähigkeitsrelevante Ereignisse führen dabei im Regelfall zu neuen Stellgrößen an die Aktoren.

Die in der Abbildung 8 dargestellten Schnittstellenkomponenten werden in Abschnitt 3.4 beschrieben.

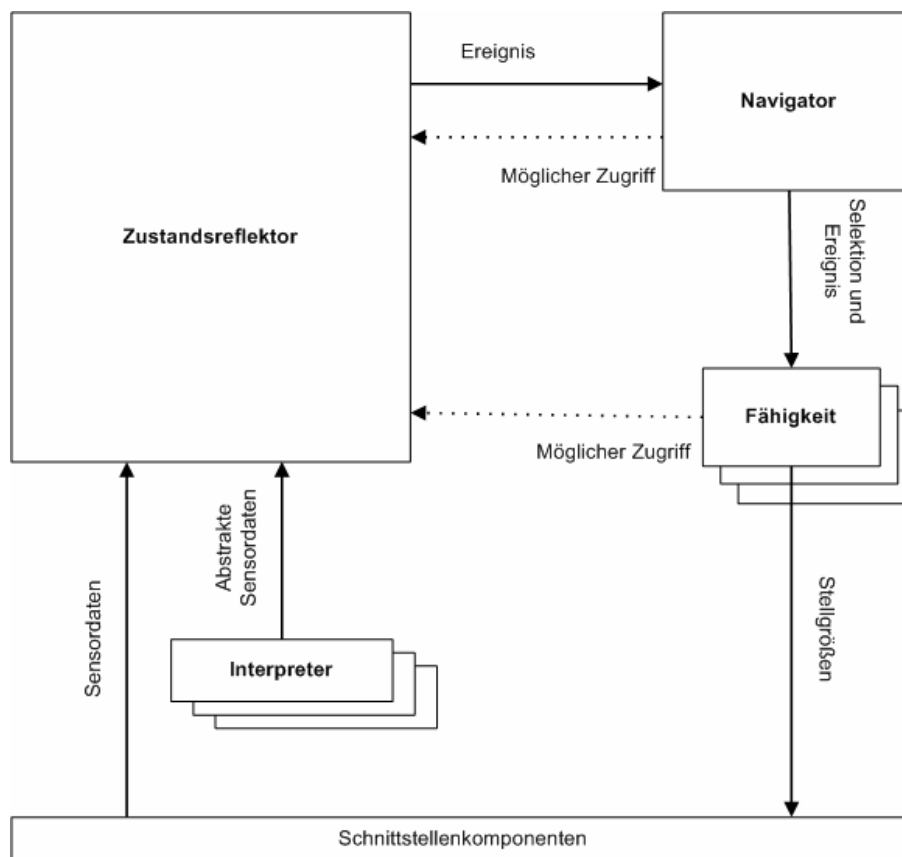


Abbildung 8: Schematische Darstellung des reaktiven Steuerungszyklus

3.3.1 Zustandsreflektor

Der Zustandsreflektor dient in erster Linie zum Sammeln aller unbearbeiteten und abstrakten Sensordaten. Dafür wurde für jeden Sensorentyp (z.B. Kamera, Hall-Sensor) ein eigener Container angelegt, der den Stand der letzten zehn Sensordaten, des jeweiligen Sensorentyps, sammelt. Die Daten des Zustandsreflektors sollen alle verfügbaren Sensordaten zentral erfassen und systemweit zur Verfügung stellen.

3.3.2 Interpretierer

Die Interpretation, beziehungsweise die Fusion von Sensordaten ist ein wichtiger Aspekt einer stabilen Steuerung. Erst durch das Zusammenführen der Informationen mehrerer Sensoren kann ein zuverlässiges Bild der Umgebung erstellt werden, da es so möglich ist, die Ungenauigkeit einzelner Sensoren auszugleichen.

Die Komponenten zur Sensorinterpretation, werden im Weiteren kurz als Interpretierer bezeichnet. Neben dem Verarbeiten von Sensordaten, die von am Fahrzeug vorhandenen Sensoren gemessen werden, ermöglichen die Interpretierer die Integration abstrakter Sensordaten in die Architektur. Abstrakte Sensordaten sind keinem Sensor direkt zuzuordnen, sondern können beispielsweise durch Fusion mehrerer Sensoren ermittelt werden. Die Position kann als abstraktes Sensordatum betrachtet werden. Sie ist an keinen Sensor direkt gebunden und kann auf unterschiedliche Art und Weise, mit Hilfe eines oder mehrerer Sensoren, bestimmt werden.

Die Interpretierer sind eigenständige Komponenten und registrieren sich für den Empfang bestimmter Sensordaten bei dem Zustandsreflektor. Die durch die Interpretation gewonnenen Daten werden anschließend wieder im Zustandsreflektor gespeichert.

3.3.3 Navigator

Aufgrund der unbekanntenen Anordnung der Positionsmarken ist eine Vorausplanung einer Ausführungssequenz nicht möglich. Die Entscheidung über die Auswahl, einer geeigneten Fähigkeit muss zur Laufzeit getroffen werden. Situationsaspekte (z.B. der aktuelle Abstand zur Marke) sind dafür entscheidend und müssen, durch den Navigator richtig gedeutet werden damit ein erwünschtes Verhalten des autonomen Systems entsteht. Dem Navigator werden dafür alle Sensordaten zur Verfügung gestellt. Situationsbedingungen können im Navigator selbst definiert werden.

3.3.4 Fähigkeiten

Fähigkeiten werden zur Fahrzeugführung eingesetzt. Sie verknüpfen funktionell die Sensorsignale mit den Aktoren und werden damit auf der systemdynamischen Ebene ausgeführt. Dabei übernimmt nur eine Fähigkeit zurzeit die komplette Steuerung und Regelung des Fahrzeugs. Die Fähigkeiten sollen Teilziele einer autonomen Fahrt erreichen. Wie auch der Navigator werden sie dafür mit allen Sensorsignalen versorgt.

3.4 Schnittstellenkomponenten

In diesem Abschnitt werden die realisierten Schnittstellenkomponenten kurz beschrieben. Eine genauere Erläuterung soll durch Kapitel 3.5 gegeben werden.

3.4.1 Abstrakte Treiber

Die abstrakten Treiber stellen ein Subsystem dieser Architektur dar. Dieses Subsystem verbindet die im Fahrzeug eingesetzte System-Hardware mit dem in dieser Arbeit vorgestellten Zustandsreflektor und bietet ein einheitliches Konzept für Erweiterungen.

Die abstrakten Treiber übernehmen auf der einen Seite die Kommunikation mit den Hardware-Komponenten und übermitteln auf der anderen Seite die Sensordaten an den Zustandsreflektor. Ein Abstrakter Treiber verbindet dabei eine Hardware-Komponente mit der Steuerungsarchitektur (siehe Abbildung 9).

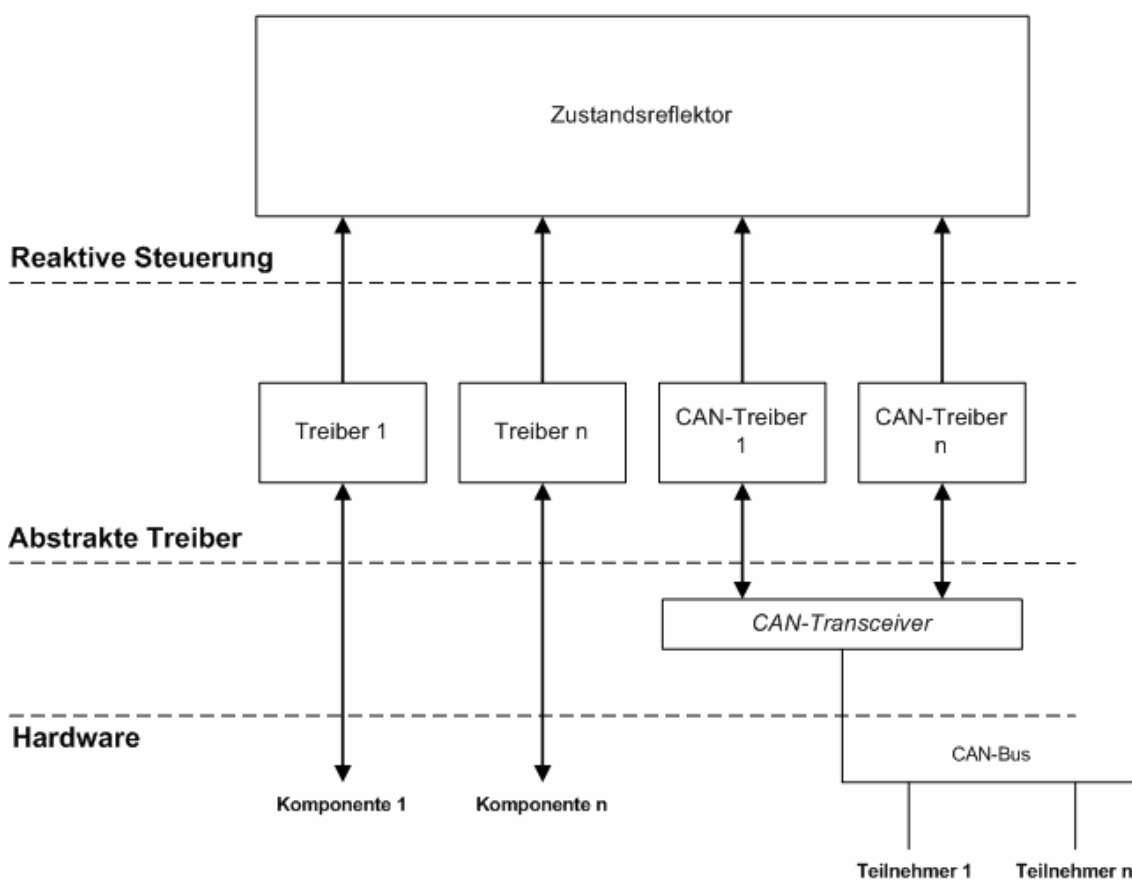


Abbildung 9: Schematische Darstellung der unterschiedlichen Schichten des Gesamtsystems

Es werden zwei Arten von abstrakten Treibern im System unterschieden. Die normalen Treiber und die CAN-Treiber. Die CAN-Treiber verfügen außer einer Schnittstelle zum Zustandsreflektor, für die Übergabe der Sensordaten zusätzlich die Verbindung zum *CAN-Transceiver* und können sich für den Empfang von CAN-Nachrichten anmelden und diese verschicken.

3.4.2 CAN-Transceiver

Um auf einem bestehenden Bussystem senden oder empfangen zu können, ist eine Hardware-Schnittstelle (Transceiver) nötig. Meist wird diese in Form einer PCI-Karte oder eines USB-Adapters angeboten. Jeder Hersteller liefert seine eigenen Treiberdateien und Programmierschnittstellen (meist in der Programmiersprache C). Das Interface *ICANTransceiver* stellt die abstrakte Beschreibung der Treiberfunktionalität dar und erlaubt die Implementierung ohne die Berücksichtigung einer speziellen Hardware.

3.4.3 Abstrakte Aktorenschnittstelle

Um den Zugriff auf die Aktoren zu erleichtern, wurde eine abstrakte Aktorenschnittstelle realisiert. Auf die abstrakte Aktorenschnittstelle wird genauso zugegriffen wie auf echte Aktoren. Hinter der abstrakten Aktorenschnittstelle verbirgt sich jedoch nicht ein physischer Aktor, sondern eine Komponente, an welche PWM-Werte übergeben werden. Diese Komponente (*Engine*) sorgt selbständig dafür, dass die PWM-Werte an die physischen Aktoren übermittelt werden.

3.5 Implementierung

Die Implementierung der Steuerungsarchitektur ist in C++ und weitestgehend objektorientiert realisiert. Dabei wird sowohl von Templates, als auch der Standard-Template-Library (STL) Gebrauch gemacht.

Das System ist in verschiedene Komponenten aufgetrennt. Die Aufteilung stellt eine Kapselung von Problemlösungen dar und ermöglicht damit eine Wiederverwendbarkeit einzelner Softwarekomponenten. Beruhend auf den Basisklassen können einzelne Implementierungen vorgenommen werden, welche die Grundfunktionalitäten der jeweiligen Komponente implementieren.

Basisklasse	Komponente
INavigator	Navigator
IAbility	Fähigkeiten
IInterpreter	Interpreter
IDriver	Abstrakte Treiber
ICANDriver	Abstrakte CAN-Treiber
ICANTransceiver	CAN-Transceiver

Tabelle 1: Basisklassen-Übersicht

3.5.1 Sensordatenerfassung

An dem Vorgang der Sensordatenerfassung sind mehrere Komponenten der Steuerungsarchitektur beteiligt. Für eine erfolgreiche Integration eines Sensors werden ein abstrakter Treiber, der normalerweise die Kommunikation mit der Hardware-Komponente übernimmt, eine Sensor-Datenklasse und ein Container im Zustandsreflektor gebraucht.

Die unterschiedlichen Daten der verschiedenen Sensoren werden in eigenen Klassen gekapselt, die über entsprechende Datenstrukturen zur Aufnahme der jeweiligen Senso-

rendaten verfügen und die dafür benötigten Zugriffsmethoden (Set- & Get-Methoden) bereitstellen. Nachdem ein abstrakter Treiber ein Sensordatum erfasst hat, wird eine Instanz der dazugehörigen Sensor-Datenklasse erstellt und an den Zustandsreflektor übergeben. Dieser erkennt den Typ des Übergabeobjekts und ordnet das Objekt einem dafür angelegtem Container zu.

In der Gleichen weise werden die Datenobjekte der Interpreter übergeben und gespeichert. Im unterschied zu den Hardware-Sensoren werden die Datenobjekte eines Interpreters nicht durch einen abstrakten Treiber sondern durch einen Interpreter erzeugt.

Für die Container wurde eine generische *TDatabase*-Klasse implementiert, die hauptsächlich auf einem Container der STL basiert. Aufgrund der Nutzungsweise dieser Container habe ich mich für einen *Double-ended-queue*²-Container entschieden, der sowohl am Anfang als auch am Ende des dynamischen Arrays ein effizientes Hinzufügen bzw. Entfernen der gesammelten Daten sicherstellt. In der Aktuellen Konfiguration sammelt und verwaltet ein Container bis zu zehn Objekte.

Im aktuellen Zustand sind die Daten der in der Tabelle 2 aufgeführten Sensoren verfügbar.

Sensor	Schnittstelle	Sensordatenklasse	Treiberklasse /Interpreter
Kamera	USB	<i>CAMData</i>	<i>CAMDriver</i>
Beschleunigungssensor und Gyroskop	CAN-Bus	<i>ASCData</i>	<i>ASCDriver</i>
Hall-Sensoren	CAN-Bus	<i>HSCData</i>	<i>HSCDriver</i>
Position Interpreter	Prozessintern	<i>PositionData</i>	<i>PositionInterpreter</i>
Zeitgeber	Prozessintern	<i>TimeData</i>	<i>Timer</i>

Tabelle 2: Implementierte Sensoren

3.5.2 Ereignisverarbeitung

Ereignisse werden in Form von Funktionsaufrufen verteilt. Nachdem ein neues Sensordatenobjekt in einem mit *TDataBase* angelegten Container erfasst wurde, wird durch diesen ein Funktionsaufruf ausgelöst. Dabei wird die Objektreferenz des Sensordatenobjekts als Attribut übergeben damit ein unmittelbarer Zugriff auf die Daten erfolgen kann.

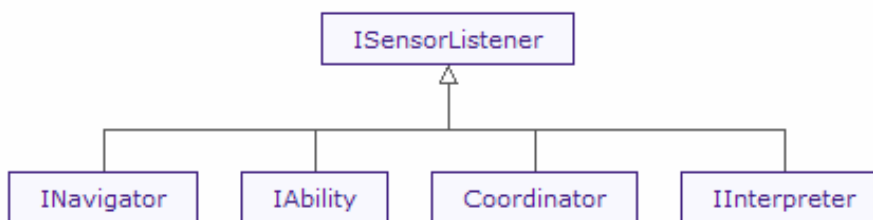
Die Ereignisverarbeitung erfolgt durch späte Bindung (Laufzeitpolymorphie). In der abstrakten Basisklasse *ISensorListener* werden dazu für alle Sensorenklassen *virtuelle* Funktionen in der folgenden Form deklariert:

² „*double-ended-queue*“ ist ein nach zwei Seiten offenes dynamisches Array

```
Class ISensorListener
{
...
virtual onUpdate(CAMData&);
virtual onUpdate(ASCDData&);
virtual onUpdate(HSCData&);
virtual onUpdate(PositionData&);
virtual onUpdate(TimeData&);
...
}
```

Abbildung 10: Ereignisse

Alle Komponenten die auf Ereignisse bzw. Sensordaten angewiesen sind, werden unter Umständen in mehreren Hierarchieebenen von dieser Klasse abgeleitet. Durch Delegation und Laufzeitpolymorphie kann jedes Empfänger-Objekt diese Funktionen verarbeiten und für die eigene Implementierung benutzen. In der folgenden Abbildung sind alle von *ISensorListener* abgeleiteten Klassen dargestellt.

Abbildung 11: Von *ISensorListener* abgeleitete Klassen

Das hier angewandte Verfahren zur Weitergabe neuer Sensordaten entspricht in der Funktionsweise dem Beobachter-Entwurfsmuster [GoF95]. Dabei melden sich alle Interpreter eigenständig, im Zustandsreflektor, für den Empfang der benötigten Daten an. Für die Navigation und die Fähigkeiten übernimmt diese Aufgabe die *coordinator*-Klasse, die im nächsten Abschnitt beschrieben wird.

3.5.3 Koordination

Die Steuerungsarchitektur bietet die Möglichkeit Fähigkeiten zu implementieren. Eine konkrete Implementation einer Fähigkeit besteht aus einer oder mehreren Funktionen für die Sensordatenverarbeitung (siehe Abbildung 10). Diese Funktionen dienen der Sensordatenübermittlung und werden sequentiell aufgerufen, daher ist keine Synchronisation (z.B. mit Hilfe eines Mutex³) innerhalb einer Fähigkeit nötig. Innerhalb der Fähigkeit können mit Hilfe der Sensordaten Stellgrößen für die Aktoren des Fahrzeugs berechnet werden und die Servomotoren gestellt werden.

³ Mutex ist eine Abkürzung für engl. Mutual Exclusion und bezeichnet Verfahren, mit denen verhindert wird, dass nebenläufige Prozesse bzw. Threads gleichzeitig auf die Daten zugreifen und so unter Umständen inkonsistente Zustände herbeiführen.

Eine Anwendung der Steuerungsarchitektur besteht immer aus einem Navigator und einem Satz von Fähigkeiten. Die dafür entwickelten Klassen werden in der *Coordinator*-Klasse instanziiert (siehe Abbildung 12).

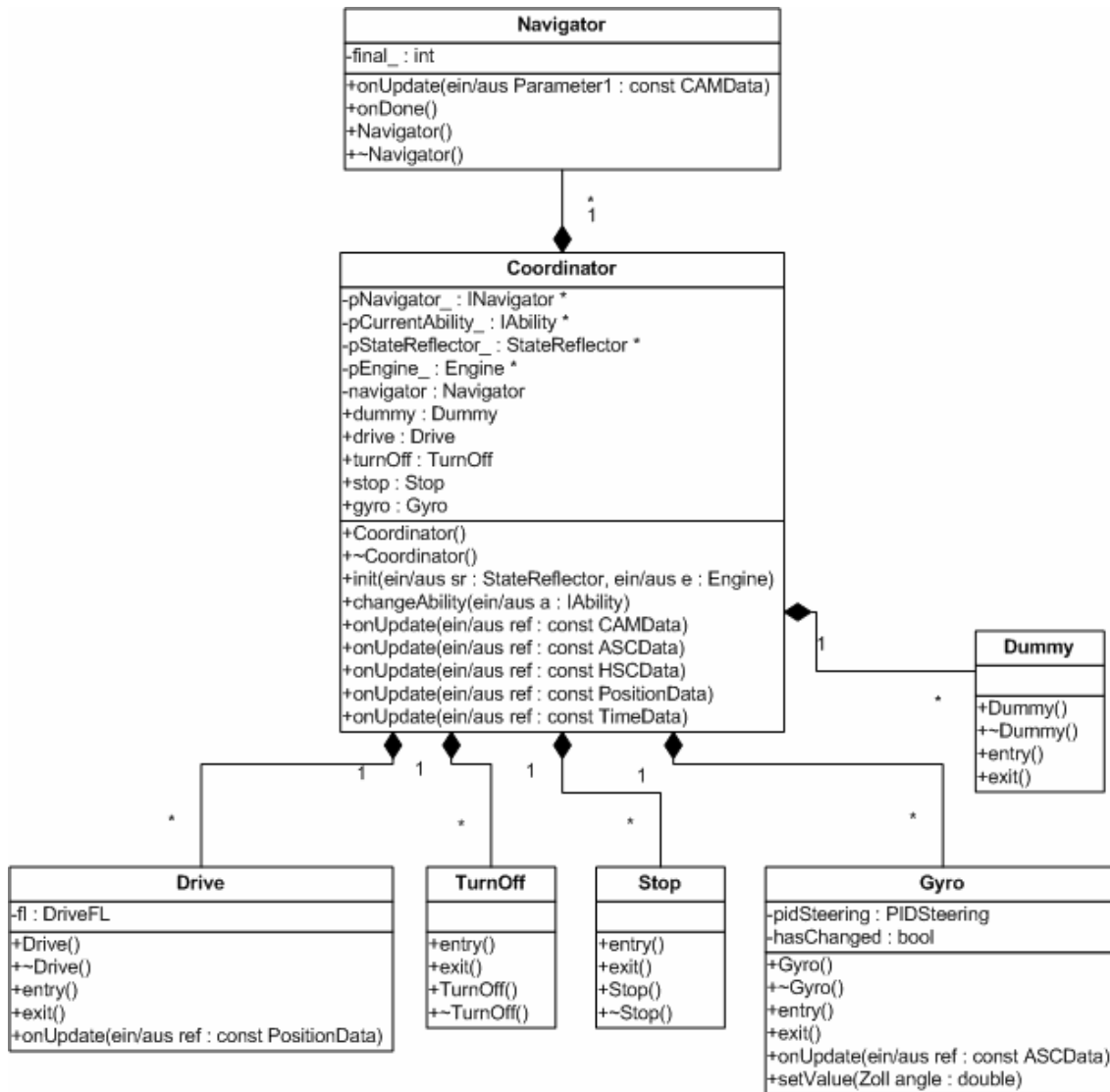


Abbildung 12: Klassendiagramm - Darstellung der implementierten Fähigkeiten und des Navigators im Koordinator

Der Koordinator meldet sich als Einziger für den Empfang aller Ereignisse im Zustandsreflektor an und leitet diese zuerst an den konkreten Navigator, dann an die selektierte Fähigkeit weiter. Die Selektion einer Fähigkeit wurde mit Hilfe eines typisierten Zeigers realisiert, der in der *coordinator*-Klasse gehalten ist. Dieser Zeiger wird mit einer *Dummy*-Fähigkeit initialisiert die keine Ereignisse verarbeitet und keine Steuergrößen an die Aktoren übergibt.

Navigator

Neben dem Zugriff auf den Zustandsreflektor verfügt der Navigator über die *changeAbility(IAbility&)*-Funktion für das Selektieren einer Fähigkeit und eine *onDone()*-Funktion. Die *onDone()* Funktion wird aufgerufen wenn eine Fähigkeit die Beendigung ihrer Ausführung meldet (vgl. „Feedback“ Abschnitt 3.2). Bei der Verwendung der Fähigkeiten in Form abgeschlossener Aktionen wird diese Funktion aus der Fähigkeit heraus aufgerufen.

Fähigkeiten

Durch die Ableitung von *IAbility* bekommen die Fähigkeiten Zugriff auf den Zustandsreflektor, sowie die abstrakte Aktorenschnittstelle. Zudem verfügen sie über die Funktionen *done()* für das Feedback und die Funktionen *entry()* und *exit()*.

Die *entry()*-Funktion wird im Moment der Aktivierung einer Fähigkeit ausgeführt. Unmittelbar vor der Aktivierung einer anderen Fähigkeit wird die *exit()*-Funktion ausgeführt. Eine wichtige Aufgabe dieser Funktionen ist die Vorinitialisierung bzw. das Zurücksetzen der internen Datenstrukturen einer Fähigkeit. Aus dem Beispiel im Abschnitt 3.1 ist ersichtlich, dass eine Fähigkeit ggf. mehrmals aktiviert werden muss. Unter Umständen kann ebenfalls der Navigator die Ausführung unterbrechen, um eine andere Fähigkeit zu aktivieren. Bei einer wiederholten Aktivierung einer Fähigkeit, könnte diese, ohne ein Zurücksetzen der internen Datenstrukturen, mit alten Zwischenwerten mit der Ausführung fortfahren und falsche Stellwerte für die Aktoren Berechnen.

3.5.4 CAN-Bus

Beim CAN-Bus werden die Daten in Frames übertragen, die eine maximale Länge von 8 Byte haben. Jeder Frame wird über seine ID (Identifikator) eindeutig identifiziert. Wie bereits in Abschnitt 3.4.2 erwähnt, stellt das Interface *ICANTransceiver* die abstrakte Beschreibung der Treiberfunktionalität dar. Die Klasse *ICANTransceiver* stellt dazu einerseits die Schnittstelle zum Versenden von CAN-Frames zur Verfügung und bietet andererseits die Möglichkeit sich für den Empfang von CAN-Frames zu registrieren. Die CAN-Frames selbst werden in Objekten der *CANFrame*-Klasse gehalten. Für die Verteilung der *CANFrame*-Objekte wurde das Beobachter-Muster angewandt.

Aktuell wird ein „USB-to-CAN-compact“, ein Adapter der Marke IXXAT eingesetzt. Die Implementierung erfolgte unter der Verwendung der Hersteller-API (VCI30) und befindet sich in der *IXXATTransceiver*-Klasse.

3.6 Einordnung

Die hier vorgestellte Steuerungsarchitektur bedient sich beider in dem Kapitel 2 vorgestellten Architekturen. Die Steuerung und Regelung der Aktoren wird in den Fähigkeiten gekapselt die Teilziele einer positionsmarkenbasierten Fahrt umsetzen sollen. Diese arbeiten reaktiv und werden dazu mit allen Sensorensignalen verbunden. Der Navigator überprüft kontinuierlich bestimmte Bedingungen die festlegen, wann die aktuelle Fähigkeit von einer Nachfolgefähigkeit abgelöst wird. Dabei kann stets nur eine Fähigkeit selektiert werden. Ein gleichzeitiges Ausführen der Fähigkeiten ist nicht möglich. Die Ausführung des Plans entspricht damit der sequentiellen Ausführung der Fähigkeiten. Ferner werden die Verarbeitungsschritte der Navigation und der aktiven Fähigkeit im-

mer streng sequentiell ausgeführt und bestimmen zusammen das Gesamtverhalten des Systems. Der Unterschied zur funktionalen Architektur besteht hier also hauptsächlich darin, dass die Aktionen durch reaktive Fähigkeiten ersetzt wurden, die wie Verhalten implementiert werden können. Die reaktive Arbeitsweise aller Komponenten lässt dennoch eine verhaltensbasierte Steuerung des Fahrzeugs zu, solange auf eine umfassende Weltmodellbildung (vgl. Kapitel 2) verzichtet wird. Eine solche Anwendung wird in Kapitel 5 vorgestellt.

4 Positionsmarkenverfolgung mit Fuzzy-Regeln

Inspiziert durch verschieden Artikel [Tzu03] [Sho05] [Pra04], die verschiedene Anwendungen der Fuzzy-Logic aus dem Bereich autonomer Systeme beschreiben, wurde eine Fuzzy-Regelung für die Positionsmarkenverfolgung entworfen, die in diesem Kapitel vorgestellt wird.

4.1 Einführung in die Fuzzy-Methodiken

Die Regelung komplexer Prozesse mit Hilfe von Unscharfer Logik (*Fuzzy Logic*), die 1965 von Lot A. Zadeh eingeführt wurde [Zad65], ist im Laufe der Zeit zu einer festen Größe in der Regelungstechnik geworden. Die Unscharfe Logik geht von der Annahme aus, dass alle Dinge nur zu einem gewissen Grad zutreffen und erlaubt es strukturiertes Wissen (Erfahrungswissen) in numerischer Form (Regeln) zu kodieren. Es ist also nicht notwendig, für die Regelung eines Systems ein vollständiges mathematisches Modell aufzustellen. Dabei ermöglicht gerade der Mangel an Präzision, in Situationen, in denen unvollständige oder teilweise widersprüchliche Informationen vorliegen, Entscheidungen zu treffen.

In diesem Abschnitt sollen die benötigten Grundlagen kurz erläutert werden. Dabei werden jeweils nur diejenigen Methoden der Unscharfen Logik aufgezeigt, die auch in der vorliegenden Fuzzy-Regelung verwendet wurden. Für eine ausführliche und praxisrelevante Darstellung von Fuzzy-Methoden sei auf [Kah95] verwiesen.

4.1.1 Fuzzy-Mengen und linguistische Terme

Die Unscharfe Logik bietet die Möglichkeit, umgangssprachlich formulierte Sachverhalte mathematisch zu repräsentieren und weiter zu verarbeiten. So kann empirisch gewonnenes, nicht in exakter Form vorliegendes Modellwissen als Grundlage komplexer Regelprozesse dienen.

Eine *Fuzzy-Menge* A über einem Wertebereich X ist definiert als eine geordnete Menge von Paaren

$$A := \{x, \mu_A(x) \mid x \in X\}$$

Wobei die Abbildung

$$\mu_A : X \rightarrow [0,1], \mu_A \in R$$

jedem Element von X einen *Zugehörigkeitsgrad* $\mu_A(x)$ zuordnet. Diese Funktion heißt *Zugehörigkeitsfunktion* von F . Da eine Fuzzy-Menge durch ihre Zugehörigkeitsfunktion vollständig und eindeutig definiert ist, wird der Einfachheit halber μ_A als Bezeichner für eine Fuzzy-Menge verwendet.

Eine *linguistische Variable* ist eine natursprachliche Größe (z.B. "Geschwindigkeit"). Sie wird im Allgemeinen durch eine Reihe so genannter *linguistischer Terme* beschrieben, die Zustände bezüglich der linguistischen Variablen umgangssprachlich beschreiben (z.B. "langsam", "schnell", "sehr schnell"), und denen jeweils eine Fuzzy-Menge

zugeordnet ist. Die Gesamtheit der Fuzzy-Mengen überdeckt dabei den gesamten Wertebereich der Variablen.

Bildet man einen *scharfen*, also numerisch wohl definierten, Wert einer linguistischen Variable (in unserem Beispiel könnte das 20 km/h sein) auf die linguistischen Terme ab, indem man für den scharfen Wert den jeweiligen *Zugehörigkeitsgrad* zu jeder Fuzzy-Menge ermittelt, spricht man von *Fuzzyifizierung*. Ein Beispiel liefert die folgende Abbildung.

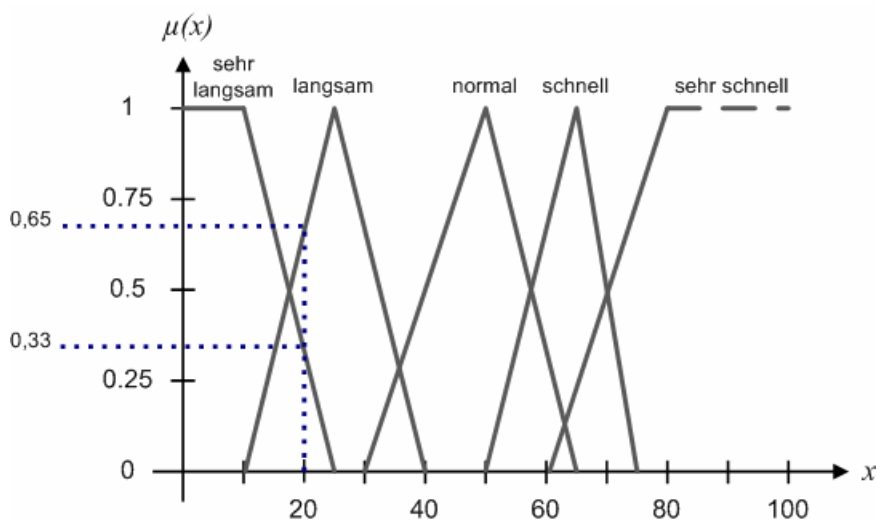


Abbildung 13: Fuzzy-Mengen am Beispiel der linguistischen Variable X: „Geschwindigkeit“

Auf X sind fünf Fuzzy-Mengen definiert, denen die linguistischen Termen *sehr langsam*, *langsam*, *normal*, *schnell* und *sehr schnell* zugeordnet sind. Die Fuzzy-Menge *langsam* beispielsweise überdeckt den Wertebereich von 10 bis 40 km/h. Für den Wert 20 km/h ist eine Fuzzyifizierung skizziert. Sein Zugehörigkeitsgrad zur Fuzzy-Menge *sehr langsam* ist ungefähr 0,33, der zur Menge *langsam* 0,65. Für alle anderen Mengen ist der Zugehörigkeitsgrad 0. Für alle zur linguistischen Variablen gehörenden Fuzzy-Mengen ergibt sich also: $\mu(20 \text{ km/h}) = (0,33; 0,65; 0; 0; 0)$.

4.1.2 Operatoren auf Fuzzy-Mengen

Auf Fuzzy-Mengen kann man, wie auch in der klassischen, binären Logik, logische Operatoren definieren. Es existiert eine Vielzahl unterschiedlicher Möglichkeiten dies zu tun. Wie bereits erwähnt werden hier nur jeweils diejenigen aufgezeigt, die auch im vorliegenden System verwendet wurden. Die Vereinigung $\mu_{A \cup B}$ zweier Fuzzy-Mengen A und B mit den entsprechenden Zugehörigkeitsfunktionen μ_A und μ_B wird definiert als:

$$\mu_{A \cup B}(x) := \max(\mu_A(x), \mu_B(x)).$$

Dies entspricht dem ODER-Operator. Dem UND-Operator entspricht die Schnittmenge:

$\mu_{A \cap B}$, sie ist definiert als:

$$\mu_{A \cap B}(x) := \min(\mu_A(x), \mu_B(x)).$$

Die UND- und ODER-Operatoren veranschaulicht Abbildung 13.

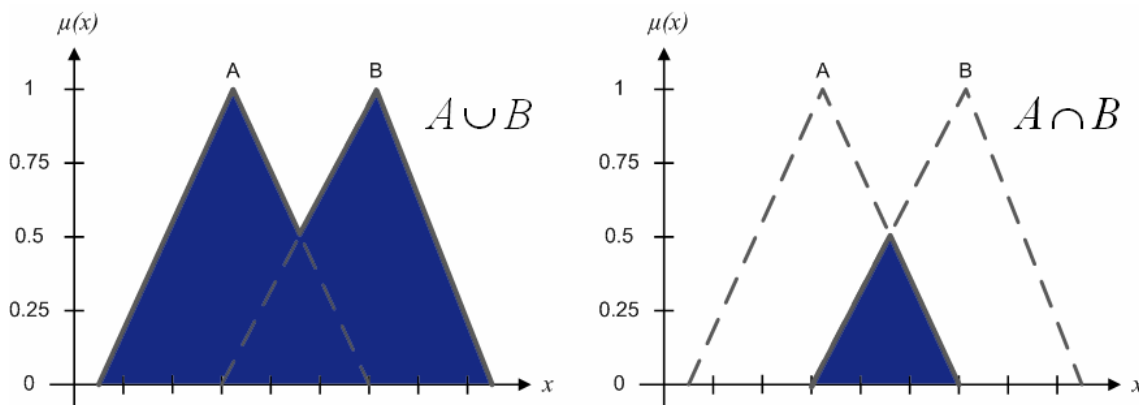


Abbildung 14: Links: Schnitt zweier Fuzzy-Mengen (AND-Operator). Rechts: Vereinigung zweier Fuzzy-Mengen (OR-Operator).

4.1.3 Fuzzy-Implikation und Fuzzy-Inferenz

Will man empirisches Systemwissen bzw. Erfahrungswissen mit unscharfen Methoden modellieren, kann man auf die *Fuzzy-Implikation* zurückgreifen, um mit linguistischen Variablen Wenn-Dann-Regeln in der Form

$$\text{WENN } x = A \text{ DANN } y = B$$

Aufzustellen. Für die verbal formulierte Regel "Wenn die Geschwindigkeit langsam ist, dann ist die Gefahr gering" wäre die Eingangsgröße x die linguistische Variable "Geschwindigkeit", die einen linguistischen Term "niedrig" erfüllen muss, damit die Ausgangsgröße y (die Variable "Gefahr") durch den Term „niedrig“ charakterisiert wird. Im Gegensatz zur klassischen Logik ist die Prämisse nicht entweder wahr oder falsch, sondern kann beliebige Erfüllungsgrade annehmen. Entsprechend ist auch die Konklusion ein unscharfer Wert. Der Erfüllungsgrad einer Prämisse entspricht im einfachsten Fall dem Zugehörigkeitsgrad der entsprechenden Fuzzy-Menge.

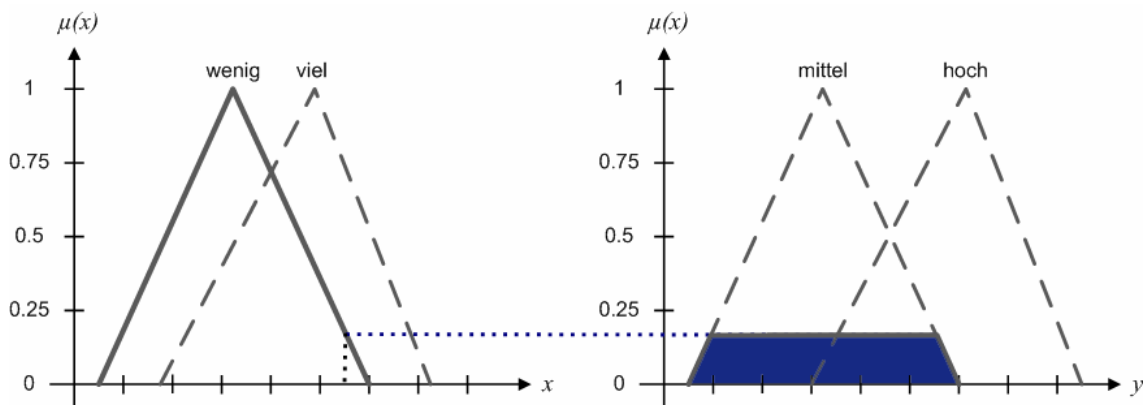
Die Fuzzy-Implikation wird definiert als:

$$\mu_A \Rightarrow B(x, y) := \min(\mu_A(x), \mu_B(y)).$$

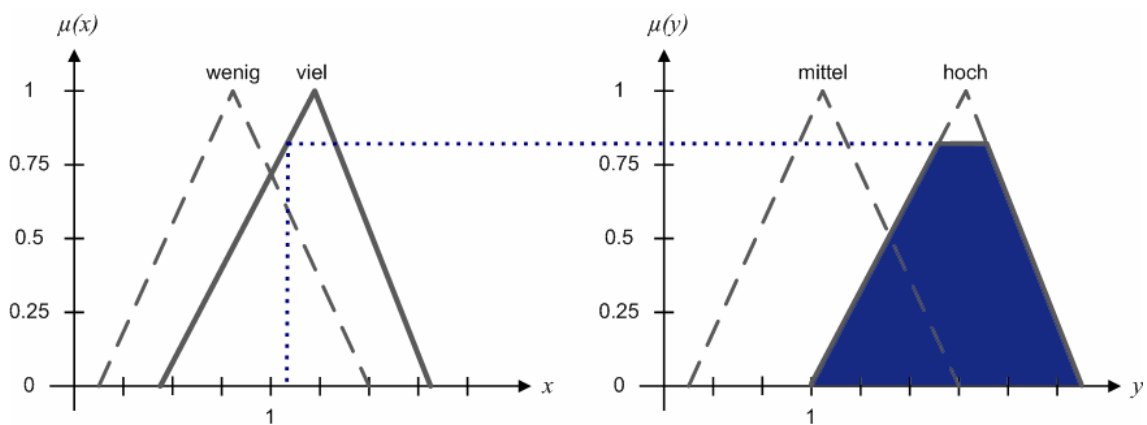
Diese Definition (die so genannte *Mamdani-Implikation*) hat insbesondere die Eigenschaft, dass der Wahrheitsgehalt des Ergebnisses nie höher sein kann als der Wahrheitswert der Prämisse.

Die Auswertung eines Satzes von Regeln bezeichnet man als *Inferenz*. Eine grafische Zusammenstellung eines Inferenzvorgangs zeigt Abbildung 14.

1. WENN x=wenig DANN y=mittel



2. WENN x=viel DANN y=hoch



Resultierende Fuzzy-Menge

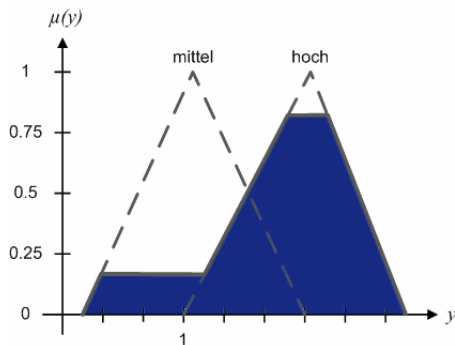


Abbildung 15:Fuzzy-Inferenz mit zwei Regeln über einem Eingangsbereich X mit einem Ergebnisbereich Y.

Die zu den linguistischen Termen „wenig“ und „viel“ zugehörigen Ergebnismengen werden auf der Höhe des jeweiligen Zugehörigkeitsgrades der Eingangsmenge abgeschnitten. Die letztendlich resultierende Fuzzy-Menge ergibt sich durch die Anwendung des Vereinigungsoperators auf die beiden Ergebnismengen.

Sind für eine Ausgangsgröße nach dem Inferenzvorgang mehrere Fuzzy-Mengen aktiv (d. h. der Zugehörigkeitsgrad des Eingangswerts ist größer als 0), werden diese mittels des ODER-Operators vereinigt. Die so entstandene Ergebnismenge muss nun noch defuzzifiziert werden, um einen scharfen Ausgabewert zu erhalten. Eine gebräuchliche

Methode hierfür ist die so genannte *Schwerpunkt-Methode*, bei der sich der Ergebniswert aus der Projektion des Schwerpunktes, der durch die Fuzzy-Menge beschriebenen Fläche auf der Achse des Wertebereichs ergibt.

4.2 Konzeption des Fuzzy-Positionsmarkenreglers

Die Aufgabe der Fuzzy-Regelung ist es das Fahrzeug zu einer Positionsmarke durch Vorgabe der Soll-Geschwindigkeit in km/h und des Soll-Lenkeinschlags in Grad zu regeln. Diese Informationen sollen später dazu genutzt werden mit einer Geschwindigkeits- und einer Lenk-Regelung die Servomotoren zu setzen, deren Ausarbeitung parallel in Diplomarbeit [UN07] erledigt wird und an dieser Stelle nicht weiter erläutert wird.

4.2.1 Aufbau

Die Fuzzy-Regelung soll in jedem Verarbeitungsschritt die aus der Bildverarbeitung gewonnenen Zielinformationen als Eingangsgrößen benutzen und im Ausgang die Soll-Geschwindigkeit und den Soll-Lenkeinschlag bereitstellen. Aufgrund der zwei zu regelnden Größen im Ausgang wurde die Gesamtregelung mit zwei Fuzzy-Reglern realisiert. Der eine übernimmt die Längsregelung und der andere die Querregelung. Die folgende Abbildung stellt diesen Zusammenhang dar.

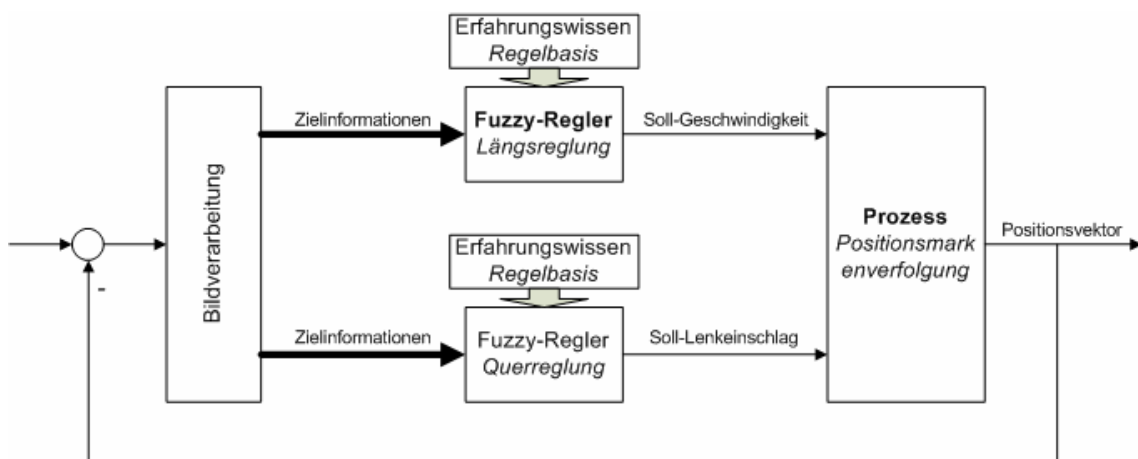


Abbildung 16: Fuzzy-Regelung zur Positionsmarkenverfolgung

Die einzelnen Fuzzy-Regler ermitteln die Soll-Geschwindigkeit und den Soll-Lenkeinschlag. Über der Zeit verändern diese Größen die Position des Fahrzeugs. Im nächsten Verarbeitungsschritt werden mit Hilfe der Bildverarbeitung neue Zielinformationen ermittelt und die Regler berechnen entsprechende Stellwerte.

4.2.2 Grundlage der Regelbasis

Die Zusammenstellung von Regeln für eine Fuzzy-Regelung basiert auf Erfahrungswissen. Der vollständige Regelsatz wird auch Regelbasis genannt und stellt die Übertragungslogik zwischen dem Eingang und dem Ausgang eines Reglers dar. Das besagte Erfahrungswissen wurde zu Beginn in verbaler Form als Leitfaden festgehalten.

Für den Entwurf der Fuzzy-Regelung wurden folgende Regeln aufgestellt:

1. Die Entfernung bestimmt die Geschwindigkeit.
2. Der Führungswinkel bestimmt den Lenkeinschlag.
3. Die Geschwindigkeit bestimmt den Lenkeinschlag.
4. Der Lenkeinschlag bestimmt die Geschwindigkeit.

Zu 1: Bei einer großen Entfernung zur Marke soll das Fahrzeug schnell fahren, ist das Fahrzeug in der unmittelbaren Nähe soll die Geschwindigkeit gegen null gesetzt werden.

Zu 2: Im Fall eines großen Führungswinkels muss stärker gegengelenkt werden als bei einem kleinen Führungswinkel, um auf die Zielbahn zu kommen.

Zu 3 und 4: Je höher die Geschwindigkeit, desto vorsichtiger muss gelenkt werden, sonst droht das Fahrzeug auszubrechen.

Die Regeln sind untereinander abhängig, z.B. benutzt die Regel 3, die Geschwindigkeit, die durch Regel 1 bestimmt wird um den Lenkeinschlag zu beeinflussen. Aufgrund, dass die Geschwindigkeit auf die Entfernung (Regel 1) und der Lenkeinschlag auf den Führungswinkel (Regel 2) zurückführbar sind, werden nur die Entfernung und der Führungswinkel als Eingangsgrößen für die Regelung verwendet.

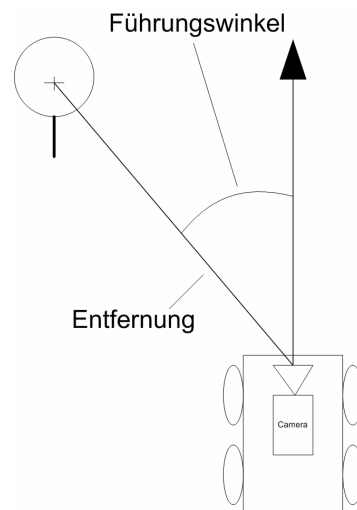


Abbildung 17: Eingänge der Fuzzy-Regelung

Der neue Regelsatz sieht dann so aus:

1. Die Entfernung bestimmt die Geschwindigkeit.
2. Der Führungswinkel bestimmt die Geschwindigkeit.
3. Die Entfernung bestimmt den Lenkeinschlag.
4. Der Führungswinkel bestimmt den Lenkeinschlag.

Dieser verbal formulierte Regelsatz stellt die Grundlage der realisierten Regelung dar.

4.3 Konkrete Umsetzung

Aus den im vorherigen Abschnitt verbal formulierten Regelsatz ergeben sich für beide Regler jeweils zwei Eingangsgrößen: *Entfernung* und *Führungswinkel*. Daher sind die beiden Regler folgendermaßen definiert

Längsregler:

Eingangsgrößen: Entfernung und Führungswinkel.

Ausgangsgröße: Soll-Geschwindigkeit.

Querregler:

Eingangsgrößen: Entfernung und Führungswinkel.

Ausgangsgröße: Soll-Lenkeinschlag.

Um nun zu der entsprechenden Fuzzy-Reglung zu kommen, wurden für alle Eingangsgrößen auf ihrem jeweiligen Wertebereich mehrere Fuzzy-Mengen definiert, die qualitativen Bewertungen der Größe entsprechen.

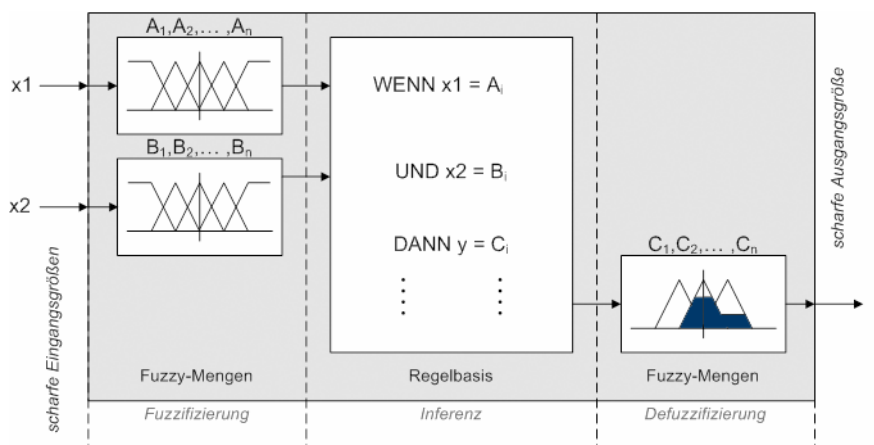


Abbildung 18: Schematische Darstellung eines Fuzzy-Reglers mit zwei Eingangsgrößen, einer Ausgangsgröße

Genauso wurden auch für die Ausgangsgrößen Fuzzy-Mengen definiert. Danach konnten Implikationen in der Form

$$\text{WENN } x1 = A_i \text{ UND } x2 = B_i \text{ DANN } y = C_i$$

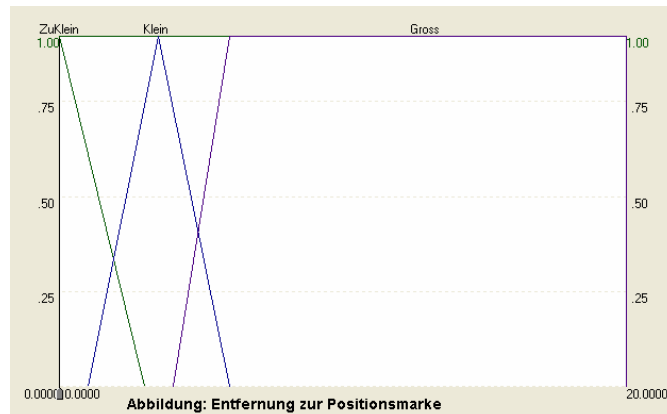
Aufgestellt werden. Für jeden neuen Bildverarbeitungsvorgang konnte nun ein Inferenzschritt durchgeführt und so die Soll-Geschwindigkeit bzw. der Soll-Lenkeinschlag ermittelt werden.

4.3.1 Längsregelung

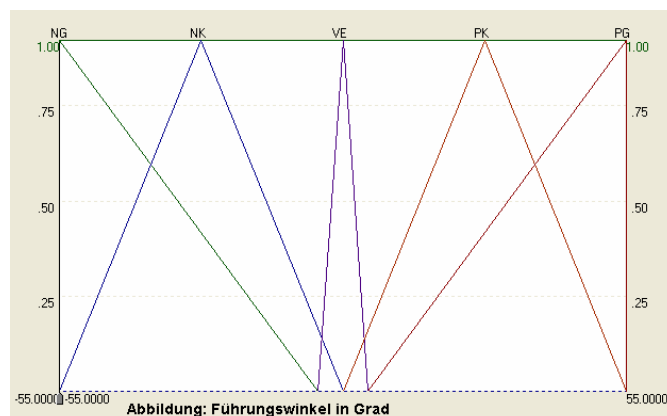
Die Längsregelung regelt die Soll-Geschwindigkeit des Fahrzeugs. Ziel ist es, die Geschwindigkeit vorausschauend zu regeln, wie z.B. eine Geschwindigkeitsabnahme vor einer Kurve oder ein Anhalten des Fahrzeugs im Fall eines zu kleinen Abstands zur Positionsmarke.

Folgende linguistische Variablen wurden für die Längsreglung definiert:

- Die Entfernung zur Positionsmarke. Die Variable kann Werte zwischen 0 und 20 annehmen. Auf den Wertebereich sind drei Fuzzy Mengen aufgeteilt. Der linguistische Term *ZuKlein* soll den Bereich beschreiben in dem die minimale Entfernung zur Marke unterschritten ist und die Geschwindigkeit gegen Null geregelt werden muss. Im mittleren Bereich (*Klein*) soll das Fahrzeug abbremesen, um ggf. eine Kurvenfahrt einzuleiten. Im Bereich (*Gross*) soll versucht werden so schnell wie möglich zur Marke zu kommen.

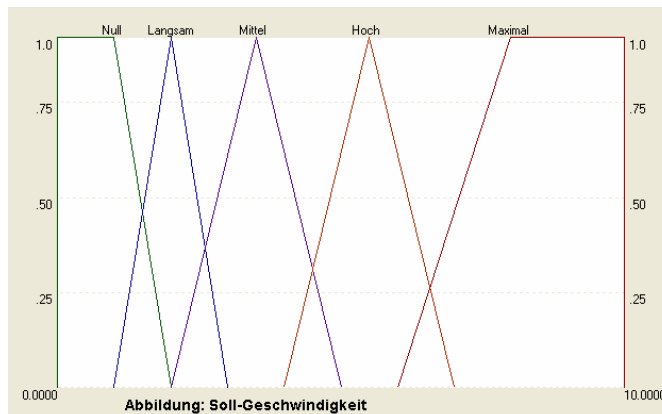


- Der Führungswinkel in Grad. Der Winkel mit dem das Fahrzeug auf die Positionsmarke auftreffen würde. Die Variable kann Werte zwischen -55 und +55 Grad annehmen (siehe Abbildung). Dieser Wertebereich wurde größer ausgelegt als der Sichtbereich der Kamera (ca. +/-37Grad), damit alle Werte verarbeitet werden können. Auf den Wertebereich sind fünf Fuzzy-Mengen aufgeteilt: *NG* (Negativ Groß), *NK* (Negativ Klein), *VE* (Verschwindend), *PK* (Positiv Klein), *PG* (Positiv Groß). Die Größe der Mengen ist unterschiedlich, Winkeländerungen im sehr geringen Bereich können so im Regelsatz gesondert berücksichtigt werden. Hier fallen auch kleinere Unterschiede stark ins Gewicht. Die folgende Abbildung stellt die fünf Fuzzy-Mengen farbig dar.



- Die Soll-Geschwindigkeit in km/h. Die Variable kann vorerst nur Werte zwischen 0 und 10 annehmen. Dadurch wird die maximale Geschwindigkeitsvorga-

be auf 10 km/h begrenzt. Dies soll besonders in der Testphase ein Ausbrechen des Fahrzeugs unterbinden.



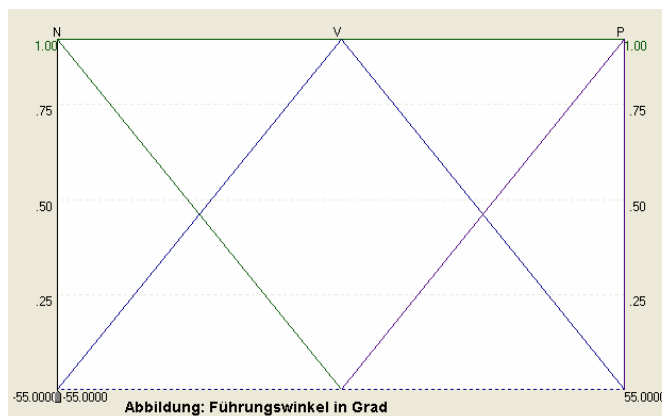
Der erstellte Regelsatz findet sich im Anhang A.1.

4.3.2 Querregelung

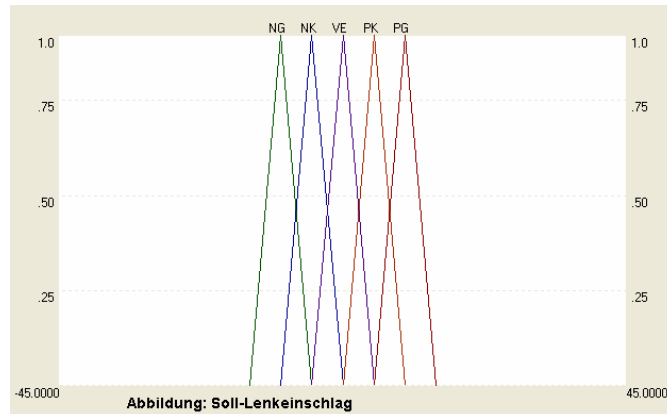
Die Aufgabe der Querregelung ist es den Soll-Lenkeinschlag zu Regeln. Dabei soll das Fahrzeug so gelenkt werden, dass eine möglichst gleichmäßige Bahn zur erkannten Positionsmarke entsteht und diese auf einem unmittelbaren Weg erreicht wird.

Folgende linguistische Variablen wurden für die Längsregelung definiert:

- Die Entfernung zur Positionsmarke. Diese Variable wurde genauso definiert wie bei der Längsregelung (siehe Abschnitt 4.3.1).
- Der Führungswinkel in Grad. Auf den Wertebereich sind drei Fuzzy-Mengen aufgeteilt: *N* (Negativ), *V* (Verschwindend), *P* (Positiv).



- Der Soll-Lenkeinschlag in Grad. Auf Grund der relativ langsamen Bildverarbeitung (ca. 1mal pro Sekunde), darf grundsätzlich kein zu großer Lenkwinkel gestellt werden, da ansonsten das Fahrzeug aus dem Sichtbereich der Marke fährt, bevor eine neue Information eintrifft und ein Gegenlenken stattfinden kann. Daher kann die Variable nur Werte zwischen -15 und plus 15 Grad annehmen.



Der erstellte Regelsatz findet sich im Anhang A.2.

4.3.3 Implementation

Für die Implementierung der Regler wurde auf eine bestehende, frei verfügbare Fuzzy-Logic Bibliothek [IFFLL] zurückgegriffen, die für zeitkritische Anwendungen ausgelegt ist. Diese erlaubt es, mit Hilfe einer FCL⁴-Datei, die erstellten Fuzzy-Mengen und Regeln in maschinenlesbare Form zu bringen und stellt Klassen für die Fuzzy-Inferenz zur Verfügung.

4.4 Simulator

Des Weiteren wurde im Rahmen dieser Arbeit ein Simulator entwickelt, mit dem die aufgestellte Regelung getestet werden konnte. Dieser simuliert die Bewegungen eines Fahrzeugs in einer virtuellen Umgebung. Die GUI des Simulators besteht aus drei Dialogfeldern (siehe Abbildung 18).

Eigenschaften

Virtuelle Karte:

- Die Darstellung erfolgt in einer einstellbaren Skalierung in Pixel/Meter. (Default-Wert: 50 Pixel = 1 Meter).
- Positionsmarken können aufgestellt und ausgerichtet werden. Die Positionierung erfolgt durch Angabe der Entfernung zum Ursprung in Metern. Dabei ist der Ursprung die Startposition des Fahrzeugs.
- Abtastzeit für die Bahnberechnung und Visualisierung kann eingestellt werden (Default-Wert: 20ms).

Kameraeinstellungen:

- Einstellung der Abtastzeit der simulierten Kamera (Default-Wert 1000ms).

⁴ FCL ist die Abkürzung für „Fuzzy Control Language“ und ist eine international genormte Sprache (IEC 61131-7) für die Implementierungen von Fuzzy-Logik in Speicherprogrammierbaren Steuerungen.

Hauptbedienfeld:

- Aktivierung des autonomen Modus.
- Kontrollanzeigen.
- Andere Bedienelemente zu Testzwecken.

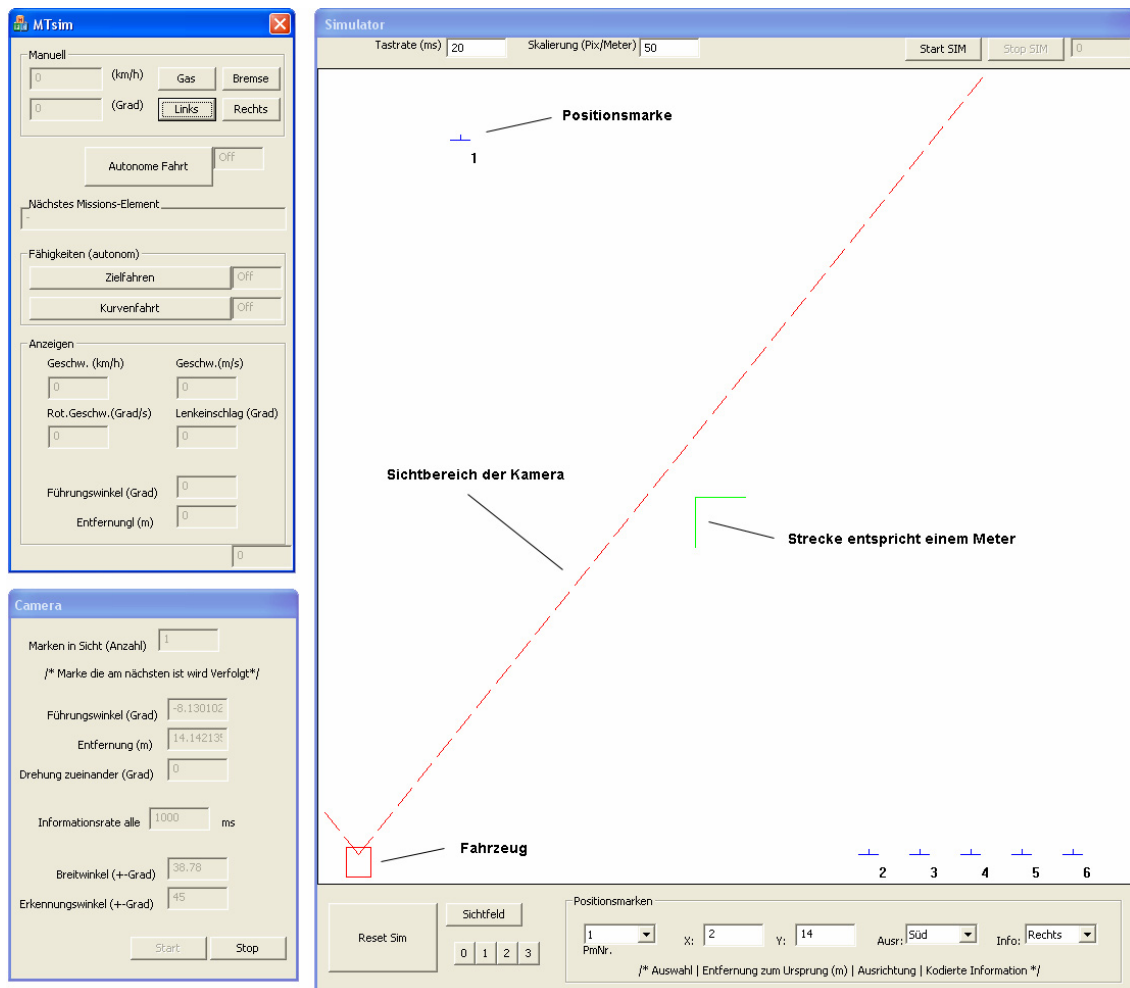


Abbildung 19: GUI –Simulator: Oben links: Hauptbedienfeld. Unten links: Kameraeinstellungen. Rechts: virtuelle Karte

Funktionsweise

Für eine Simulation muss erstmal die virtuelle Kamera gestartet werden. Diese liefert in der zuvor eingestellten Taktrate die Entfernung und den Führungswinkel und speichert diese in einer internen Datenstruktur ab. Wird nun der autonome Modus im Hauptbedienfeld aktiviert, werden die durch die virtuelle Kamera erzeugten Informationen als Eingangsgrößen für die Fuzzy-Reglung genutzt, die Ausgangsgrößen werden intern abgespeichert. Dieser Vorgang wird kontinuierlich mit der für die Kamera gesetzten Taktrate ausgeführt. Zuletzt wird der Prozess der Bahnberechnung gestartet. Dieser liest die intern abgespeicherten Ausgangsgrößen der Regelung aus (Geschwindigkeit und Lenkeinschlag) und berechnet kontinuierlich die neue Position des Fahrzeugs. Bei der Be-

rechnung der Bahn wird vom typischen Vierrad-Modell ausgegangen. In Abbildung 19 ist ein solches Modell zu sehen. Daraus lassen sich folgende, aus der Literatur [Sie-Nou04]

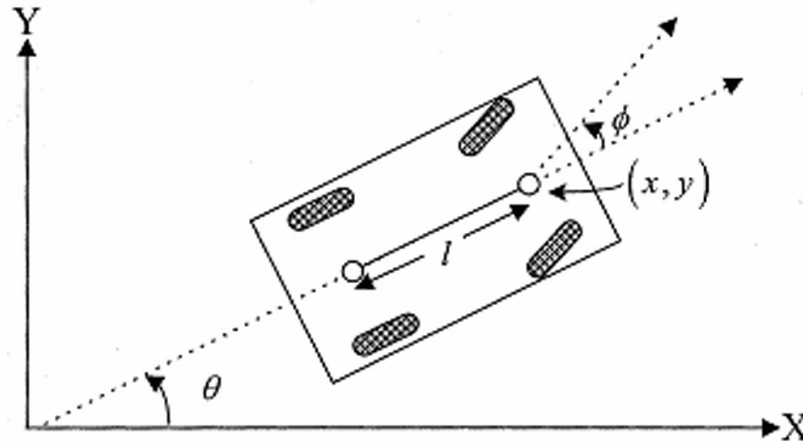


Abbildung 20: Das Vierrad-Modell

bekanntesten Gleichungen für die Vorwärtsbewegung herleiten:

$$\begin{aligned}\dot{x} &= v * \cos(\theta + \phi) \\ \dot{y} &= v * \sin(\theta + \phi) \\ \dot{\theta} &= v * \frac{\sin \phi}{l}\end{aligned}$$

Die nach diesen Gleichungen durchgeführte Berechnung wird alle 20ms wiederholt und das Fahrzeug entsprechend abgebildet.

Da bei einer derartigen Bewegung die Bahn mit dieser Methode durch Geradenstücke angenähert wird entsteht durch die Abtastzeit eine gewisse Abweichung von der tatsächlichen Bahn. Dies wurde zu Gunsten einer einfachen Berechnung in Kauf genommen.

4.5 Simulationsergebnisse

Einfluss der Zielinformationsrate auf das Fahrverhalten:

Der Einfluss der Abtastzeit der Kamera auf das Fahrverhalten wurde erstes in mehreren Simulationen überprüft. Abbildung 20 zeigt vier generierte Trajektorien der gleichen Aufgabe mit unterschiedlichen Abtastzeiten der Kamera. Der Abstand zwischen der Startposition des Fahrzeugs und der Marke beträgt dabei fünfzehn Meter.

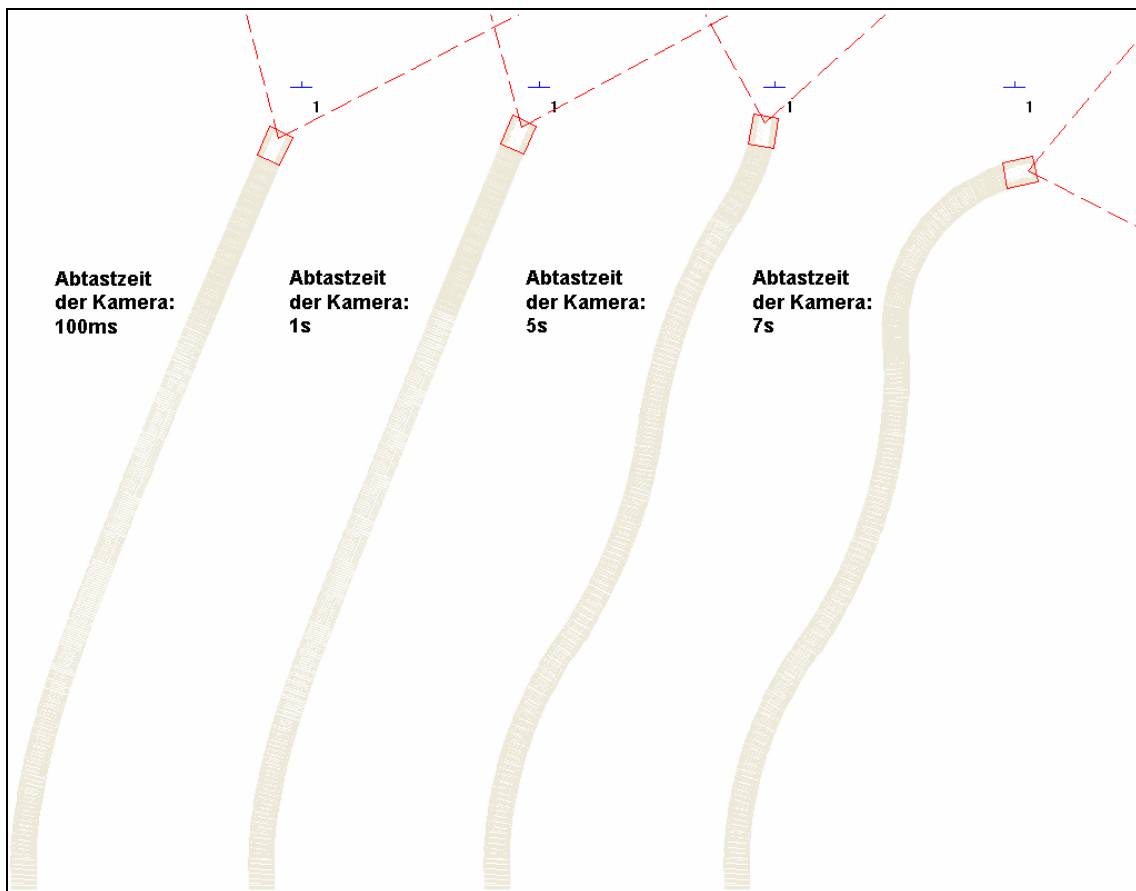


Abbildung 21: Fahrverhalten in Abhängigkeit von der Abtastzeiten der Kamera

Bei einer Abtastzeit von 100ms erzeugt die Simulation eine gleichmäßige Bahn auf die Marke hinzu. Aus dem in Abbildung 21 dargestellten Geschwindigkeitsverlauf sind die unterschiedlichen Bereiche der linguistischen Variable *Entfernung* erkennbar (siehe 4.3.1). Zuerst wird die Entfernung der Menge *Gross* zugeordnet, dies ist an der hohen Geschwindigkeit zu erkennen ($\text{Zeit} < 9\text{s}$). Anschließend wird die Geschwindigkeit auf 2km/h herabgesetzt und gehalten. Hier wurde die Entfernung der Menge *Klein* zugeordnet ($8\text{s} < \text{Zeit} < 17\text{s}$). Zum Schluss wird die Geschwindigkeit gegen Null geregelt da die Entfernung der Menge *ZuKlein* zugeordnet wurde ($> 17\text{s}$).

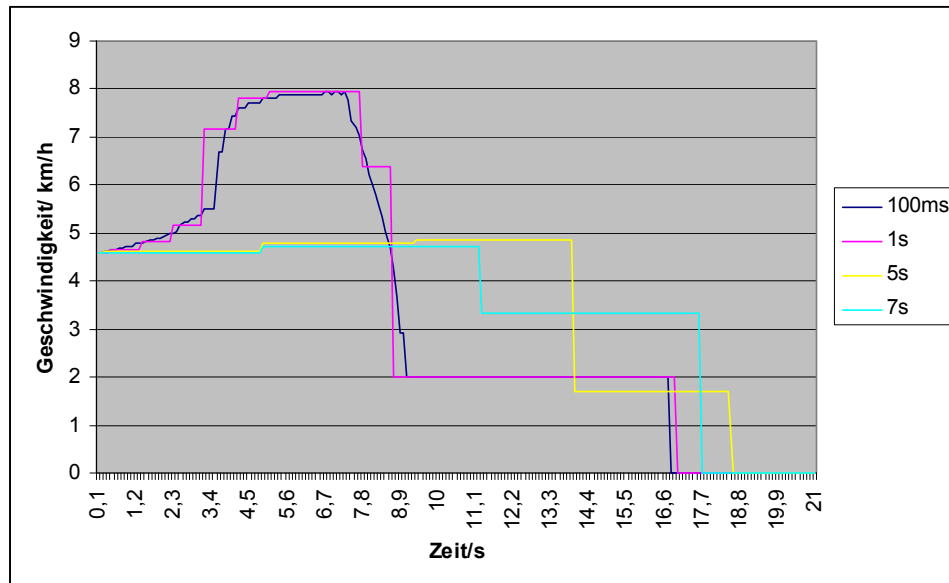


Abbildung 22: Längsregelung bei unterschiedlichen Abtastzeiten der Kamera

Die Bahn wird immer ungleichmäßiger, je größer die Abtastzeit der Kamera ist. Dennoch konnte das Fahrzeug, bei einer Abtastzeit von 5 Sekunden, die Marke erreichen (vgl. Abbildung 20). Dies ist vor allem auf die Querregelung zurückzuführen die in Abhängigkeit vom Führungswinkel unterschiedlich stark gegenlenkt. (siehe Abbildung 22).

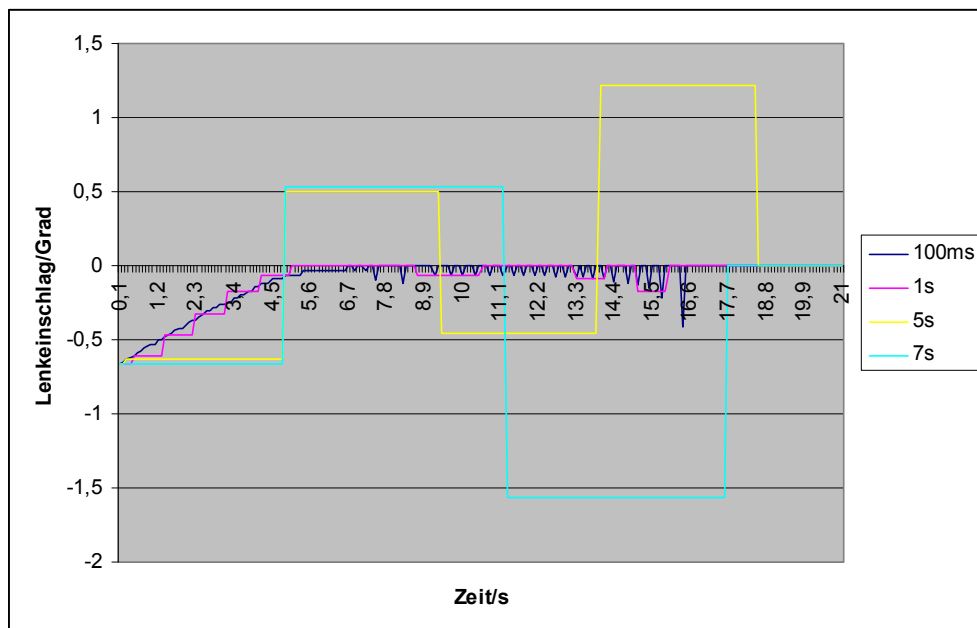


Abbildung 23: Querregelung bei unterschiedlichen Abtastzeiten der Kamera

Einfluss einer falsch positionierten Kamera auf das Fahrverhalten

In Abbildung 23 sind die Trajektorien zwei weiterer Tests zu sehen. Der Abstand der Marke zur Startposition des Fahrzeugs beträgt wieder 15 Meter. Die Abtastzeit der Kamera wurde auf 1 Sekunde eingestellt.

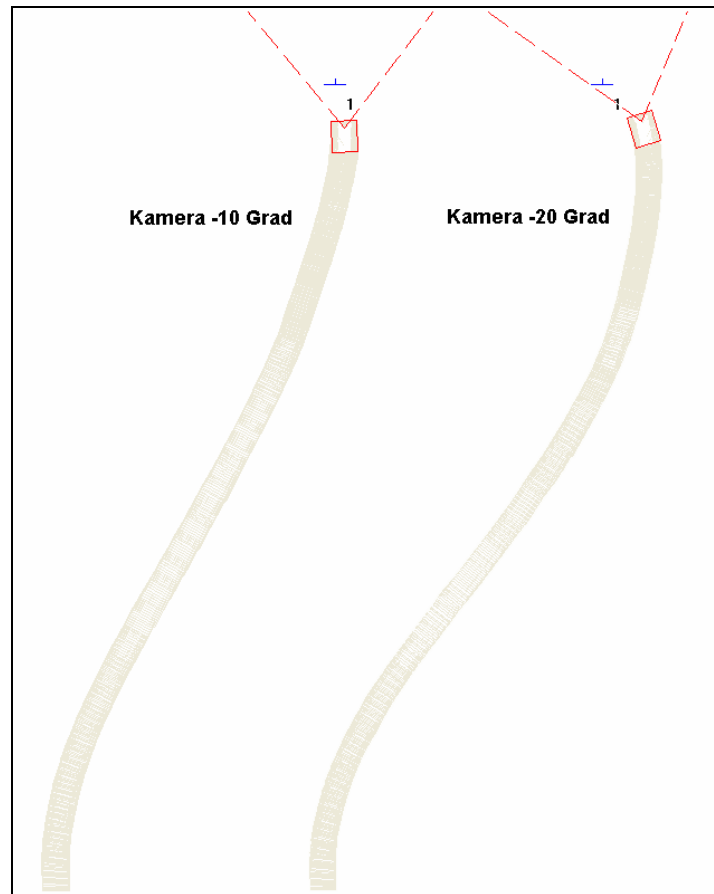


Abbildung 24: Simulation mit schiefer Kamera

Bei dieser Simulation wurde der Führungswinkel, den die Regelung im Eingang bekommt, um 10 bzw. 20 Grad versetzt. Dies entspricht einer um den jeweiligen Winkel versetzten Kamera. Trotz der relativ großen Winkelabweichungen erreicht das Fahrzeug die Marke. In der Abbildung 24 ist der Verlauf der Geschwindigkeit und des Lenkeinschlags für den Fall einer kontinuierlichen Abweichung des Führungswinkels von 20 Grad dargestellt. Beide Regler (Längs- und Querregler) arbeiten zusammen um das Problem der versetzten Kamera zu bewältigen. Dies ist ab der zwölften Sekunde besonders gut zu sehen.

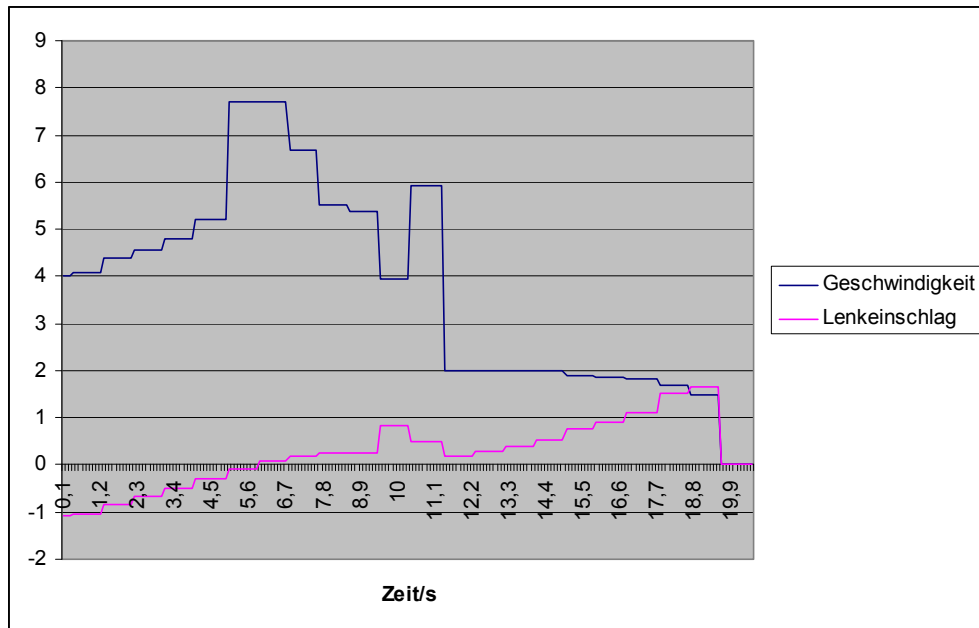


Abbildung 25: Längs- und Querregelung bei versetzter Kamera um 20 Grad

Einfluss einer falsch justierten Lenkung auf das Fahrverhalten:

In Abbildung 25 ist die Simulation mit einer um 1 Grad verstellten Lenkung dargestellt.

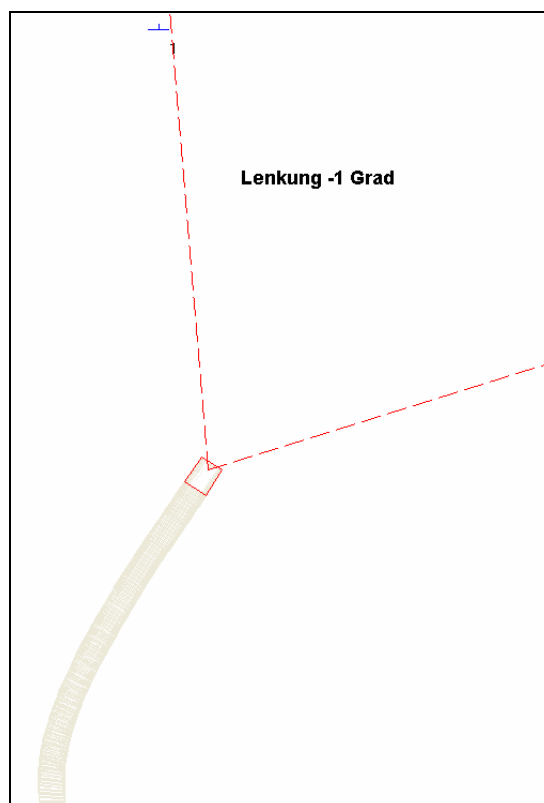


Abbildung 25: Simulation mit einer falsch justierten Lenkung

Die Abtastzeit der Kamera wurde auf eine Sekunde eingestellt und die Anfangsentfernung zur Marke hat 14 Meter betragen. In dieser Situation ist der Querregler nicht in der Lage ausreichend stark entgegen zu lenken und das simulierte Fahrzeug verliert die Marke aus dem Sichtbereich. Abbildung 26 stellt diesen Zusammenhang dar.

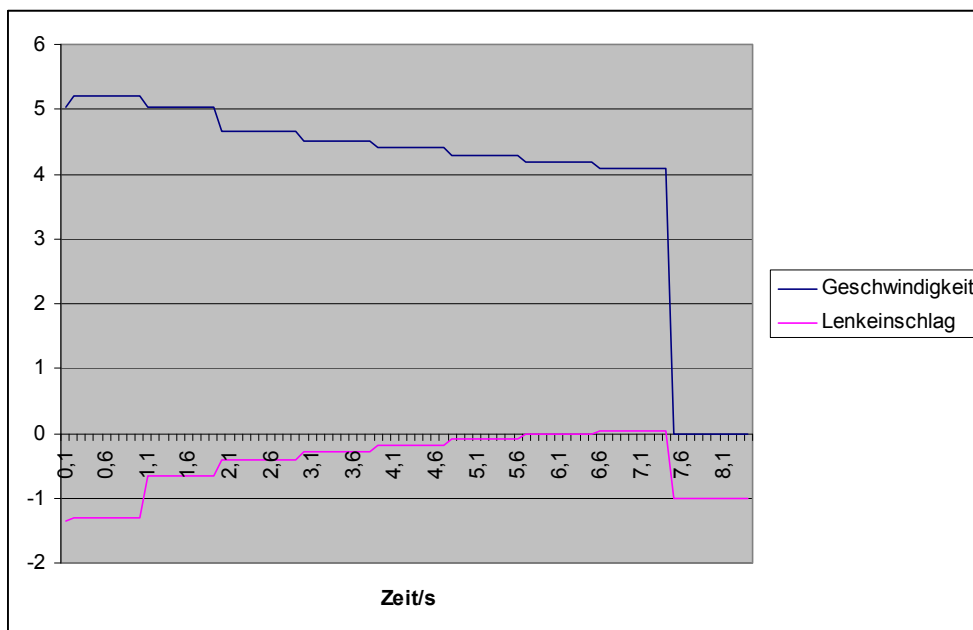


Abbildung 26: Längs- und Querreglung bei einer falsch justierten Lenkung

4.6 Zusammenfassung der Simulationsergebnisse

Durch die Simulation konnte gezeigt werden, dass die entwickelte Fuzzy-Reglung in der Lage ist, anhand der Angabe der Entfernung und des Führungswinkels, die Geschwindigkeit sowie die Lenkung zu steuern. Dabei konnte die Positionsmarke meistens erreicht werden. Eine hohe Informationsrate der Kamera ist ein Kriterium für den Erfolg einer durch diese Fuzzy-Reglung geregelten Fahrt. Diese sollte möglichst hoch und zuverlässig sein damit ein gleichmäßiger Bahnverlauf entsteht. Im Fall einer schiefen Kamera zeigt sich die Fuzzy-Reglung tolerant. Auch mit einem Versatz von 20 Grad konnte die Zielmarke erreicht werden. Bei einer schlecht eingestellten Lenkung konnte die Querreglung leider nicht ausreichend gegenlenken und das Fahrzeug ist aus dem Sichtbereich der Kamera geraten. Dies ist auf die relativ vorsichtige Lenkung zurückzuführen, die aufgrund der relativ langsamen Informationsrate der Kamera so ausgelegt worden ist. (vgl. 4.3.2, Linguistische Variable *Lenkeinschlag*).

5 Praktische Anwendung der Architektur

5.1 Beschreibung

Das Ziel der Anwendung ist es, einen Parcours anhand von Positionsmarken autonom zu bewältigen. Die Tabelle 3 zeigt die dazu implementierten Fähigkeiten.

Name der Fähigkeit	Beschreibung	Parameter	Verwendete Sensordaten
Drive	Positionsmarkenverfolgung	keine Parameter	Distanz, Führungswinkel
TurnOff	Kurvenfahrt	Richtung der Kurvenfahrt (Recht oder Links)	keine
Stop	Anhalten	keine Parameter	keine

Tabelle 3: Die schematischen Fähigkeiten

Die Drive-Fähigkeit implementiert die Fuzzy-Regelung aus Kapitel 4. Aufgrund dessen die aktuell verwendete Bildverarbeitung nur die Distanz, die kodierte Information (Rechts, Links oder Stop) sowie die X und Y Koordinaten, in Bezug auf die eigene Auflösung, der erkannten Marken liefert, musste eine Umrechnung vorgenommen werden, um den Führungswinkel zu bestimmen. Für diese Umrechnung wurde ein Interpreter implementiert.

Name des Interpreters	Beschreibung	Parameter	Verwendeter Sensor
PositionInterpreter	Ermittelt den Führungswinkel	keine Parameter	Kamera

Tabelle 4: Schematische Darstellung des Interpreters

Die Berechnung des Führungswinkels erfolgt dabei durch die lineare Umrechnung der durch die Bildverarbeitung ermittelten X-Koordinate der Positionsmarke auf den Weitwinkel des Kameraobjektives. Die TurnOff-Fähigkeit setzt ausschließlich die Geschwindigkeit auf einen niedrigen Wert und erzeugt einen Lenkeinschlag in die zuvor angegebene Richtung. Dabei findet keine Rückmeldung über den Fortschritt der Kurvenfahrt statt. Die Stop-Fähigkeit stellt die Lenkung Gerade und hält das Fahrzeug an.

Die für diese Anwendung implementierte Navigationskomponente entscheidet anhand der von der Bildverarbeitung gelieferten Informationen welche Fähigkeit in welcher Situation aktiviert wird. Abbildung 25 zeigt die Entscheidungsprozedur des Navigators.

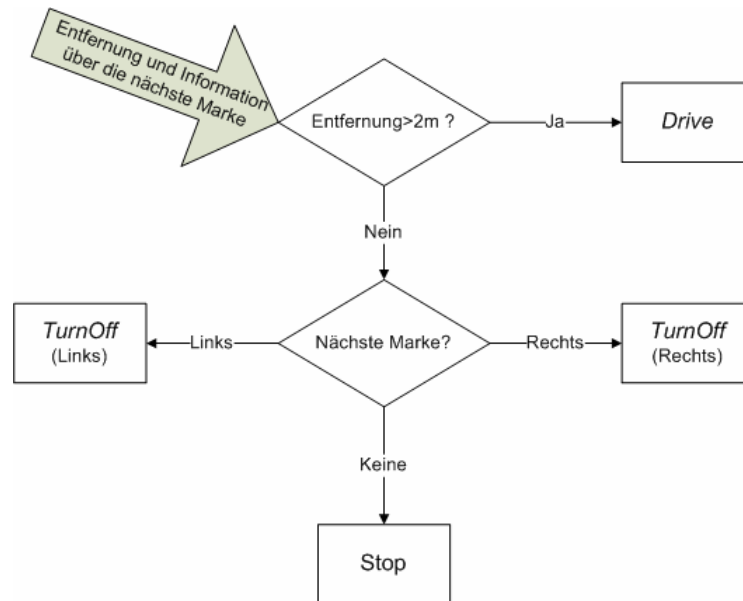


Abbildung 26: Flussdiagramm der Fähigkeitsselektion

Aufgrund der zur Zeit noch fehlenden Geschwindigkeits- und Lenkregelung werden die durch die Fähigkeiten ermittelten Werte linear auf den PWM-Bereich des jeweiligen Servomotor skaliert. Diese Vorgehensweise ist für eine richtige Testfahrt nicht geeignet, da die mechanischen und dynamischen Eigenschaften des Fahrzeugs unberücksichtigt bleiben. Die Anwendung kann aber dadurch unter Laborbedingungen getestet werden.

5.2 Simulation

Um diese Anwendung simulieren zu können wurde eine Schnittstelle zwischen der Architektur und dem in Abschnitt 4.4 vorgestellten Simulator geschaffen, die das Zusammenarbeiten beider Systeme ermöglicht. Die Kameradaten werden von der virtuellen Kamera des Simulators geliefert. Diese werden im Zustandsreflektor gespeichert und lösen den Steuerungszyklus der Steuerungsarchitektur aus. Die durch die Fähigkeiten erzeugten Stellwerte werden an die Simulation übergeben, und werden für die Bahnbe-rechnung benutzt. Auf diese Weise konnte die Anwendung auf richtige Funktionsweise überprüfen werden. In Abbildung ist Simulationsergebnis dargestellt.

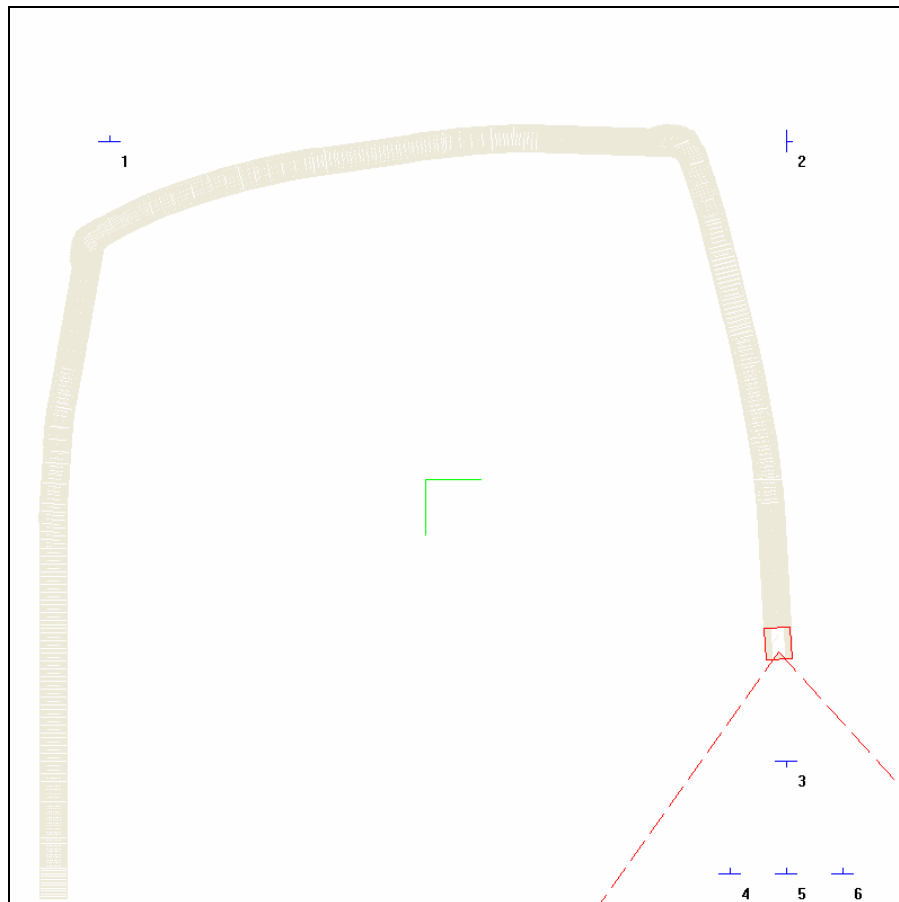


Abbildung 27: Simulation der Anwendung

Die Abtastzeit der Kamera ist bei dieser Simulation auf eine Sekunde eingestellt. In den Positionsmarken 1 und 2 ist die Information über eine weitere Marke in Richtung rechts kodiert. Die Positionsmarke 3 ist die Endmarke.

5.3 Testszzenarien

Unter Laborbedingungen konnte mit ausgeschaltetem Motor die richtige Funktionsweise der Gesamtsteuerung verzeichnet werden. Anstatt, dass das Fahrzeug fährt wurde eine Positionsmarke in unterschiedlicher Entfernung zur Kamera des Fahrzeugs gehalten. Man konnte beobachten wie das Fahrzeug in Abhängigkeit von der Distanz zur Positionsmarke sein Verhalten geändert hat. Bei einem größerem Abstand ($>2\text{m}$) wurde die Drive-Fähigkeit aktiviert und die Lenkung folgte stets der Marke. Die Reaktionszeit lag bei ca. 1 Sekunde. Dies ist aber auf die Bildverarbeitung zurückzuführen. Bei einem Abstand unter 2 Meter konnten die quasi Kurvenfahrt sowie das Anhalten beobachtet werden. Beide Servomotoren wurden durch die Anwendung richtig gesetzt.

Aufgrund der fehlenden Geschwindigkeits- und Lenkregelung war es zum aktuellen Zeitpunkt nicht möglich eine vergleichbare Testfahrt (siehe Abbildung 26) unter realen Bedingungen durchzuführen. Gleichwohl wurde mehrfach der Versuch unternommen eine autonome Positionsmarkenverfolgung des Fahrzeugs zu erreichen indem diese Anwendung in einer vereinfachten Form eingesetzt wurde. Dabei sollte das Fahrzeug mit einer konstanten Geschwindigkeit auf die Marke zufahren und bei einem Abstand unter 2 Metern anhalten. Jedoch konnte die Bildverarbeitung keine ausreichend gute Erkennung der Positionsmarken gewährleisten. Da jedoch ohne die notwendigen Bildinformationen eine autonome Fahrt nicht möglich ist und kein äquivalentes System für den Erhalt entsprechender Daten vorliegt, war eine Durchführung einer solchen Testfahrt nicht denkbar. Die resultierende Folge während der Tests war, dass die Steuerung aufgrund der fehlenden Informationen den Test in der Form beendete, dass das Fahrzeug seinen Betrieb einstellte und auf der jeweiligen Position stehen blieb.

6 Zusammenfassung

Die hier vorgestellte Arbeit beschäftigt sich mit einer hierarchischen Steuerungsarchitektur mit Fuzzy-Regelung zur positionsmarkenbasierten Navigation eines autonomen Modellfahrzeugs. Neben der Vorstellung zweier grundlegender Architekturansätze, wurden Aspekte der Fuzzy-Regelung näher vorgestellt und betrachtet.

Aufbauend auf den theoretischen Erkenntnissen, wurde eine Steuerungsarchitektur entwickelt, die eine Grundlage für die Entwicklung von weiteren Softwaremodulen darstellt. Sie bietet Schnittstellen an, schafft interne Kommunikationswege und legt Richtlinien für die Erstellung weiterer Komponenten fest.

Ein spezielles Augenmerk wurde auf die Modularität des Systems und deren Schnittstellen gelegt, so dass bereits entwickelte Module, aber auch zukünftige, mit der Steuerlogik zusammenwirken können. In diesem Zusammenhang wurde das Konzept der abstrakten Treiber, sowohl auf Software- als auch auf Hardwareebene betrachtet und angewandt. Diese Schnittstellen dienen als Zwischenglied zwischen der realen Hardware und der Softwareapplikationen.

In Anlehnung an die in dieser Arbeit entworfene Steuerungsarchitektur, wurden die notwendigen Softwaremodule, welche die zukünftigen Steuerungsaufgaben übernehmen, entwickelt. Diese Module erfüllen Aufgaben wie Sensordatenerfassung und -verarbeitung. Resultierend aus den so erhaltenen Daten werden in weiteren Arbeitsschritten Entscheidung über die auszuführende Aktion getroffen.

Eine weitere Herausforderung in dieser Arbeit stellte die Aufgabe der Steuerung des Fahrzeugs in Richtung einer Positionsmarke dar. Dies setzte einen gesonderten Ansatz und Strategie für die Steuerung voraus. Als Lösung wurde eine Fuzzy-Regelung vorgestellt, die die Anforderungen dieser Steuerungsaufgabe erfüllt. Diese lieferte erfolgreich Simulationsergebnisse und bot somit eine Basis für Weiterentwicklungen und Optimierungen.

Da aufgrund des derzeitigen Entwicklungsstandes nur bedingt Bilddaten zur Orientierung des Fahrzeuges zur Verfügung standen, wurde ein weiterer Ansatz für das Testen der hier vorgestellten Architektur und Applikationen verfolgt. Es wurde eine zeitgesteuerte Fahrt des Versuchsfahrzeuges implementiert, die eine zeitlich begrenzte Fahrt mit unterschiedlichen Lenk- und Geschwindigkeitsimpulsen beinhaltete. Dieser neue Ansatz bewährte sich in den Test im Labor, sowie im Einsatz im offenen Gelände und verdeutlichte die korrekte Funktionsweise der Steuerungsarchitektur und ihrer implementierten Module. Weiterführend und um auch die nun bereitgestellten Schnittstellen zu anderen Komponenten zu testen, wurde eine Integration eines ebenfalls in Bearbeitung befindlichen Systems vorgenommen, des Gyro-Systems [SG07]. Hier wurde deutlich, dass sich die festgelegten Schnittstellen und Konzepte für die Kommunikation der unterschiedlichen Fahrzeugkomponenten und Systeme im Alltag bewährten.

Obwohl eine autonome Fahrt mittels optischer Orientierung nicht durchgeführt werden konnte, so zeigen doch die alternativ entwickelten Tests und die daraus resultierenden Ergebnisse, dass das Konzept der Steuerungsarchitektur und des somit entstandenen Gesamtsystems aufgeht und erfolgreich für die weitere Entwicklung des Fahrzeuges genutzt werden kann.

Literaturverzeichnis

Bücher und Dokumente

- [Ark98] Arkin R. C.: Behavior-Based Robotics, The MIT Press, Cambridge, Massachusetts, 1998.
- [Bro86] Rodney A. Brooks: A robust layered control system for a mobile Robot, IEEE Transactions on Robotics and Automation, 1986.
- [Bro89] Rodney A. Brooks: A robot that walks: Emergent behavior from a carefully evolved network, Neural Computation, 1989.
- [GoF95] Gamma, Erich; Helm, Richard; Johnson, Ralph und Vlissides, John: Design Patterns: Elements of Object Oriented Software, Addison-Wesley, 1995.
- [JonFly] J. Jones, A. Flynn: Mobile Roboter - Von der Idee zur Implementierung, Addison-Wesley, 1996.
- [Kah95] Kahlert, Jörg: Fuzzy Control für Ingenieure, Friedrich Vieweg & Sohn, Braunschweig, 1995.
- [Kni] T.Knieriemien: Autonome Mobil Roboter, BI-Wissenschaftsverlag, 1991.
- [Pra04] Prahlad Vadakkepat: Fuzzy Behavior-Based Control of Mobile Robots, IEEE Transactions on Fuzzy Systems, 2004.
- [Sho05] Shou-Tao Li: An autonomous mobile robot control method based on hierarchical fuzzy behaviors, Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, 2005.
- [SieNou04] Roland Siegwart, Illah R. Nourbakhsh: Introduction to Autonomous Mobile Robots, MIT Press, 2004.
- [Tzu03] Tzue-Hsend S. Li: Implementation of Human-Like Driving Skills by Autonomous Fuzzy Behavior Control on an FPGA-Based Car-Like Mobile Robot, IEEE Transactions on Industrial Electronics, 2003.
- [Zad65] Zadeh, Lotfi A.: Fuzzy Sets, Information and Control, 1965.

Bachelor-, Diplom- und Doktorarbeiten

- [AK07] Alexander Kant (2007) – Bachelor
Bildverarbeitungsmodul zur Fahrspurerkennung für ein autonomes Fahrzeug
- [BB07] Bachir Blal (2006-2007) – Diplom

- [CP07] Geschwindigkeitsbestimmung mittels Radsensoren im Einsatz eines autonomen Fahrzeuges
Collin Pein (2006-2007) - Bachelor
- [EH07] Geschwindigkeitsregelung mittels eines Beschleunigungssensors im autonomen Modellfahrzeug
Enrico Hensel (2007) – Bachelor
- [MN07] Design und Implementation eines Sicherheitskonzepts für den Betrieb eines autonomen Fahrzeugs
Marko Natzke (2006-2007) – Diplom
- [SG07] System- und Kommunikationsarchitektur zum autonomen Betrieb eines Modellfahrzeuges
Sven Geibert (2007) – Bachelor (in Bearbeitung)
- [TM07] Gyroskopische Kurvenfahrt
Tarik Mousli (2006-2007) – Diplom
- [UN07] Visuelle Navigation eines autonomen Fahrzeugs mit Hilfe von Bildverarbeitung
Umut Nar (2007-2008) – Diplom
- Voraussichtliches Thema: Projektspezifische Qualitätssicherung und Regelung der Fahrzeugsteuerung in Bezug auf Geschwindigkeit und Lenkung

Internet

- [IFFLL] Free Fuzzy Logic Library
<http://ffll.sourceforge.net/fcl.htm>

Anhang A: Regelsätze der Fuzzy-Reglung

Im Folgenden finden sich die Regelsätze, wie sie in den Tests zu der vorliegenden Arbeit verwendet wurde. Die Darstellungsform entspricht der FCL-Norm (vgl. Abschnitt 4.3.3).

A.1 Regelsatz der Längsreglung

```
RULE 0: IF (Entfernung IS ZuKlein) THEN (Geschwindigkeit IS null);  
RULE 1: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS NG) THEN  
(Geschwindigkeit IS Null);  
RULE 2: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS NK) THEN  
(Geschwindigkeit IS Langsam);  
RULE 3: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS VE) THEN  
(Geschwindigkeit IS Langsam);  
RULE 4: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS PK) THEN  
(Geschwindigkeit IS Langsam);  
RULE 5: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS PG) THEN  
(Geschwindigkeit IS Null);  
RULE 6: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS NG) THEN  
(Geschwindigkeit IS Langsam);  
RULE 7: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS NK) THEN  
(Geschwindigkeit IS Hoch);  
RULE 8: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS VE) THEN  
(Geschwindigkeit IS Maximal);  
RULE 9: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS PK) THEN  
(Geschwindigkeit IS Hoch);  
RULE 10: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS PG) THEN  
(Geschwindigkeit IS Langsam);
```

A.2 Regelsatz der Querreglung

```
RULE 0: IF (Entfernung IS ZuKlein) THEN (Lenkeinschlag IS VE);  
RULE 1: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS N) THEN  
(Lenkeinschlag IS NG);  
RULE 2: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS V) THEN  
(Lenkeinschlag IS VE);  
RULE 3: IF (Entfernung IS Klein) AND (Fuehrungswinkel IS P) THEN  
(Lenkeinschlag IS PG);  
RULE 4: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS N) THEN  
(Lenkeinschlag IS NK);  
RULE 5: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS V) THEN  
(Lenkeinschlag IS VE);  
RULE 6: IF (Entfernung IS Gross) AND (Fuehrungswinkel IS P) THEN  
(Lenkeinschlag IS PK);
```

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum

Unterschrift