Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Karlsruhe Institute of Technology

Fakultät für Informatik
Institut für Technische Informatik (ITEC)
Lehrstuhl Industrielle Anwendungen der
Informatik und Mikrosystemtechnik (IAIM)

Diplomarbeit

Verhaltensgenerierung eines kognitiven Automobils mittels eines biologisch motivierten Verhaltensnetzwerks

Daniel Jagszent

01.07.2008 - 31.12.2008

Hauptreferent: Prof. Dr.-Ing. Rüdiger Dillmann
Korreferent: Prof. Dr.-Ing. Jürgen Beyerer
Betreuer: Dipl.-Ing. Joachim Schröder

Ich versichere hiermit, die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Die verwendeten Hilfsmittel und Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 31.12.2008

Daniel Jagszent

Inhaltsverzeichnis

Danksagung

Zuerst möchte ich meinem Betreuer *Joachim Schröder* danken. Ohne seine steten Anregungen und sein außergewöhnliches Engagement wäre diese Arbeit nicht möglich gewesen.

Außerdem möchte ich *Tobias Gindele* für die vielen wertvollen Hinweise und die zahlreichen »Template-Metaprogramming-Sessions« danken, in denen ich viel Neues über C++ und insbesondere die Boost-Bibliothek gelernt habe. Vieles davon fand Einzug in diese Arbeit.

Weiterer Dank geht an die Teammitglieder des *Team AnnieWAY* und den Mitgliedern des *Sonderforschungsbereichs Transregio 28 »Kognitive Automobile«* der Deutschen Forschungsgemeinschaft – namentlich sind hier *Annie Lien*, *Sören Kammel*, *Benjamin Pitzer*, *Moritz Werling* und *Julius Ziegler* zu nennen. Die produktive Zusammenarbeit während der intensiven Zeit der DARPA Urban Challenge 2007 und die Hilfsbereitschaft auch nach der DARPA Urban Challenge 2007 verdienen diesen Dank.

Meinen Eltern und Schwestern danke ich für die Durchsicht dieser Ausarbeitung. Unzählige Rechtschreibfehler konnten so dem Leser erspart werden. Aber auch inhaltliche Verbesserungen wurden durch diese Durchsicht erreicht.

Den Mitarbeitern und Studenten des Lehrstuhls Industrielle Anwendungen der Informatik und Mikrosystemtechnik der Universität Karlsruhe und dem Leiter des Lehrstuhls *Rüdiger Dillmann* danke ich für die freundliche und angenehme Arbeitsatmosphäre unter der diese Arbeit ihr Entstehen finden durfte.

Nicht zuletzt möchte ich meinen Freunden danken, weil sie mir das Abtauchen in die manchmal sehr intensiven Arbeitsperioden während des Erstellens dieser Arbeit verziehen haben und mir immer wieder Stärkung und Zerstreuung gaben, ohne die diese Arbeit wohl nicht möglich gewesen wäre.

1. Einleitung

Laut Angaben der Weltgesundheitsorganisation (WHO 2004) sind im Jahr 2002 ca. 1,1 Millionen Menschen an den Folgen eines Verkehrsunfalls gestorben. Geht man, wie *Evans* in (*Evans* 2004), von den US-Zahlen von 120 Verletzungen pro Todesfall aus und verallgemeinert diese auf die ganze Welt, so ergeben sich 132 Millionen Verletzungen im Jahr 2002 im Straßenverkehr. Ein Großteil der Verunglückten ist dabei sehr jung. In Deutschland sind beispielsweise 29 % aller getöteten Pkw-Insassen im Jahr 2005 zwischen 18 und 24 Jahren gewesen (Destatis 2006). Neben den dadurch ausgelösten individuellen Leid ist dies auch aus volkswirtschaftlicher Sicht ein großer Verlust in einer sonst gesunden und produktiven Altersgruppe.

Der überwältigende Hauptgrund für Unfälle ist dabei mit 86 % Fehlverhalten vom Fahrzeugführer. Nur 2 % der Unfälle hatten technische Mängel des Fahrzeugs als Ursache und 5 % konnten auf die Strassenverhältnisse zurückgeführt werden (Destatis 2006).[1] Auch *Evans* (1985) kommt in einer umfassenderen, internationalen Analyse zu dem ähnlichen Ergebnis, dass der überwiegende Anteil an Unfällen durch den Fahrer verursacht wird.

Betrachtet man diese Zahlen und stellt sie in den Zusammenhang mit dem von allen Experten vorausgesagten steigenden Verkehrsaufkommen, ist mit einer Verschlechterung der aktuellen Situation zu rechnen. Folgerichtig setzen die meisten der modernen Fahrzeugtechniken, die momentan eingeführt werden oder noch in der Entwicklung sind, bei dem Fahrer als dem offensichtlich schwächsten Glied im Verkehrssystem an und entziehen ihm Verantwortung.

So ist beispielsweise *Adaptive Cruise Control (ACC)* die Weiterentwicklung des Tempomaten, die mithilfe von Radarsensoren einen Mindestabstand zum vorausfahrenden Fahrzeug einhalten kann. Sie nimmt dem Fahrer die langweilige und damit fehlerträchtige Aufgabe des Abstandsregelns auf Autobahnen ab. In der nächsten Generation wird ACC auch das Fahren im Stop-and-Go-Verkehr beherrschen.

Weiterhin unterstützen *Einparkhilfen* den Fahrer durch Ultraschallsensoren und Videokameras. Neuerdings übernehmen *automatische Einpark-Assistenten* den ganzen Parkvorgang selbständig.

[1] Die zu 100 % fehlenden Ursachen werden hier nicht näher betrachtet.

Die Beispiele zeigen die Richtung an, in der die Entwicklung neuer Fahrerassistenzsysteme vorangetrieben wird: Sie beginnen als Hilfsmittel für den Fahrer, belassen aber die Verkehrswahrnehmung und die Auswahl des richtigen Fahrverhaltens zunächst beim Fahrer. Die Techniken werden daraufhin weiterentwickelt zu Assistenzsystemen, die auch die Verkehrswahrnehmung und Verhaltensausführung für einen Teilbereich des Autofahrens autonom und mit weniger Fehlern als ein menschlicher Fahrer übernehmen.

An dem Ende dieser Entwicklung könnte ein Fahrzeug stehen, das den Fahrer komplett als Fehlerquelle ausgeschlossen hat: Das autonom fahrende Auto, das die Verkehrswahrnehmung und Verhaltensausführung in jeder möglichen Fahrsituation erfolgreich übernehmen kann.

Um dieses Ziel zu erreichen, forschen viele Forschungsgruppen weltweit im Bereich der Informatik an Problemen der Wissensrepräsentation, dem Maschinenlernen, der Computervision und der Robotik. Eine Kooperation all dieser Teilbereiche und darüber hinaus mit Forschungsbereichen andere Fachrichtungen, wie etwa dem Maschinenbau, ist dafür notwendig.

Auch im Sonderforschungsbereich Transregio 28 »Kognitive Automobile« (SFB/TR28) der Deutschen Forschungsgemeinschaft (DFG)[1] wird daran geforscht. Er ist ein überregionales Gemeinschaftsprojekt mehrerer Institute der Universität Karlsruhe, der Technischen Universität München, der Universität der Bundeswehr München und der Fraunhofer Gesellschaft (IITB in Karlsruhe). Ziel des langjährig angelegten Sonderforschungsbereichs ist es, vollkommen autonom fahrende Fahrzeuge zu entwickeln, die sich in dem zukünftigen Straßenverkehr mit allen anderen Verkehrsteilnehmern, ob autonom oder mit einem menschlichen Fahrer, zurechtfinden. Da dieses Ziel nur ein langfristiges sein kann, wird im SFB/TR28 auch an erweiterten Fahrerassistenzsystemen, wie z.B. der automatischen Erkennung von Straßenschildern, geforscht. Wie oben erwähnt, sind solche Fahrerassistenzsysteme der erste Schritt hin zu einem vollautonomen System.

Die vorliegende Arbeit entstand zum Teil im Rahmen des SFB/TR28 am Institut für technische Informatik der Universität Karlsruhe. Sie wurde darüber hinaus auch teilweise im Rahmen von *Team AnnieWAY*[2] durchgeführt. Team AnnieWAY ist der Beitrag eines Teils des Sonderforschungsbereichs für die *DARPA Urban Challenge 2007* (DARPA 2007, siehe Abschnitt 2.3.2). Diese Zweiteilung spiegelt sich auch teilweise in dieser Arbeit wider, denn in ihr werden zwei Herangehensweisen an das Thema dieser Arbeit – der Verhaltensentscheidung und -ausführung – vorgestellt. Im Rahmen des SFB/TR28 wird ein generischer, flexibler Ansatz eines biologisch motivierten Verhaltensnetzwerkes (siehe Abschnitt 2.1.6) verfolgt. Wohingegen im Rahmen des Team AnnieWAY für die Verhaltensentscheidung und -ausführung auf die feste Struktur eines hierarchischen Zustandsautomaten (siehe Abschnitt 2.2) zurückgegriffen wurde, um in der zur Verfügung stehenden Zeit für die klar umrissenen Aufgaben der *DARPA Urban Challenge 2007* die besten Ergebnisse zu erreichen.

[1] Mehr Informationen zum SFB/TR28 sind unter `http://www.kognimobil.org/` zu finden.
[2] Mehr über das Team findet man unter `http://annieway.mrt.uni-karlsruhe.de/`.

1.1. Aufgabenstellung

Im Rahmen der im SFB/TR28 entwickelten Architektur zur Führung eines autonomen Fahrzeugs (siehe Kapitel 3) ist diese Arbeit in der Verhaltensentscheidung und -ausführung angesiedelt. Der Hauptteil der Arbeit beschäftigt sich mit der Entwicklung eines Verhaltensnetzwerks zur Führung eines kognitiven Automobils. Als Grundlage dient hier der von *Hoffmann* auf das Automobil angepasste Ansatz des biologisch motivierten Verhaltensnetzwerks (*Albiez* 2006; *Hoffmann* 2007, siehe Abschnitt 2.1.6 und 2.1.7). Ein weiterer Aspekt dieser Arbeit besteht in einem theoretischen Vergleich des hier entwickelten, modularen Systems mit einem bestehenden, hochintegrierten System.

Die Aufgabenstellung kann in die folgenden Einzelpunkte unterteilt werden, wobei ein Teil der in ihnen verwendeten Termini erst in Kapitel 2 und 3 näher erläutert wird, um diese Auflistung kompakt zu halten.

1. Der Hauptteil der Arbeit soll sich der Entwicklung eines Softwareframeworks widmen, das den Entwurf eines biologisch motivierten Verhaltensnetzwerks unterstützt. Hierbei sollen besonders die folgenden Punkte beachtet werden:

   a) Das Framework sollte die im SFB/TR28 entwickelte Softwarearchitektur nutzen und die vorherrschende Hardwareplattform unterstützen. Speziell ist hier die gute Anbindung an die im SFB/TR28 entwickelte Echtzeitdatenbank zu nennen.

   b) Das Testen der Funktionstüchtigkeit einzelner Verhalten soll leichter als bisher möglich sein. Insbesondere sollen auch automatisierte Tests unterstützt werden.

   c) Das Verhaltensnetzwerk soll auf eine im SFB/TR28 noch zu entwickelnde einheitliche Simulationskomponente vorbereitet werden. Zur Unterstützung dieser Simulationskomponente muss das Framework die Möglichkeit bieten, die Ausführung unterschiedlicher Teile des Verhaltensnetzwerks durch externe Ereignisse auslösen zu lassen. Außerdem sollte das Framework auch eine Simulation unterstützen, die nicht in Echtzeit ausgeführt wird. Somit können auch aufwendigere Algorithmen auf weniger leistungsfähiger Hardware simuliert werden.

   d) Das Framework soll eine explizite Dokumentation der Netzwerkstruktur überflüssig machen, indem es die implizit im Quelltext des Verhaltensnetzwerks definierten Abhängigkeiten der Verhalten für eine automatische Dokumentation der Netzwerkstruktur ausnutzt.

2. Ein weiterer Aspekt der Arbeit beinhaltet die Überführung des bestehenden Verhaltensnetzwerks auf das neu entwickelte Framework. Hierbei ist eine Anpassung an veränderte Schnittstellen zu anderen Systemkomponenten und veränderten Rahmenbedingungen

vorzunehmen. Insbesondere soll das Verhaltensnetzwerk in den folgenden Punkten angepasst werden:

a) Die enge Koppelung von Bahnplanung/Trajektorienplanung und der Verhaltensentscheidung soll gelöst werden. Dazu muss eine Schnittstelle zu einer neu entwickelten Bahnplanungskomponente geschaffen werden und die reaktive Schicht des Verhaltensnetzwerks muss für diese Schnittstelle überarbeitet werden.

b) Die Verhalten sollten, sofern es sinnvoll ist, die von der Situationsinterpretation aufbereiteten Modelle und nicht mehr direkt die Daten der Wahrnehmungskomponente verwenden. Insbesondere gilt dies für die Repräsentation der Fahrbahnen; die Situationsinterpretation bietet für sie ein fusioniertes Modell.

3. Zuletzt soll ein Vergleich des vom Team AnnieWAY in der DARPA Urban Challenge 2007 verwendeten hierarchischen Zustandsautomaten mit dem im SFB/TR28 verfolgten, modularen Ansatz vorgenommen werden. Da beide Systeme nicht den gleichen Funktionsumfang aufweisen, beschränkt sich der Vergleich auf eine theoretische, strukturelle Analyse der Vor- und Nachteile beider Systeme.

1.2. Überblick

Kapitel 2 führt in das Thema dieser Arbeit ein, indem einige Grundlagen für das Verständnis dieser Arbeit kurz erläutert werden und ein Überblick über den Forschungsstand im Bereich der kognitiven Automobile gegeben wird.

Darauffolgend stellt Kapitel 3 die Systemarchitektur und die verwendeten Konventionen und Hilfsmittel des SFB/TR28 vor, da diese Arbeit sich in diese Systemarchitektur integriert. In dieser Ausführung wird sich bewusst auf die Komponenten der Systemarchitektur beschränkt, die für die Verhaltensentscheidung und -ausführung von Bedeutung sind oder noch sein werden.

Nach diesen Grundlagen stellt Kapitel 4 das in dieser Arbeit entworfene Framework zur Erstellung von biologisch motivierten Verhaltensnetzwerken und das damit entworfene Verhaltensnetzwerk vor. In diesem Kapitel werden auch generelle Anforderungen an den Entwurf erarbeitet.

Kapitel 5 stellt mehrere Testfälle und ihre Ergebnisse vor und evaluiert anhand dieser Testfälle das entwickelte Framework und Verhaltensnetzwerk.

Eine Zusammenfassung der Arbeit und ein Ausblick auf zukünftige Forschungsthemen, die sich an diese Arbeit anschließen können, wird in Kapitel 7 gegeben.

2. Grundlagen

Dieses Kapitel führt in das Thema ein, indem einige Grundlagen für das Verständnis dieser Arbeit kurz erläutert werden und ein Überblick über den Forschungsstand im Bereich der kognitiven Automobile gegeben wird.

Der erste Abschnitt dieses Kapitels führt in den Bereich der Robotersteuerung und gebräuchliche Architekturen zur Robotersteuerung ein. In diesem Abschnitt wird auch auf die in diesem Ansatz verwendete Architektur der biologisch motivierten Verhaltensnetze eingegangen.

Der zweite Abschnitt behandelt kurz hierarchische Zustandsautomaten und ihre UML-Notation und im dritten Abschnitt werden einige kognitive Automobile und ihre Architekturen vorgestellt. Der vierte und letzte Abschnitt dieses Kapitels befasst sich mit den Software-Hilfsmitteln, die zur Implementierung dieser Arbeit herangezogen wurden.

Da aus Platzgründen auf eine Einführung weiterer Konzepte und Themenbereiche, wie etwa aus dem Bereich der Bahnplanung oder der Wissensmodellierung und Wissensrepräsentation, verzichtet wurde, soll hier die Erwähnung zweier Lehrbücher ausreichen, die einen guten Einstieg in die Robotik und die Künstliche Intelligenz und damit auch in diese Themenbereiche ermöglichen: *Russell* und Norvig (2004); *Craig* (2005).

2.1. Robotersteuerungen

Es lassen sich drei grundlegende Paradigmen in der Robotersteuerung identifizieren, die im Folgenden kurz erklärt werden sollen, um dann einige der verbreitetsten und einflussreichsten Architekturen mit ihnen zu klassifizieren und kurz zu erläutern.

2.1.1. Paradigmen

In *Vlacic* (2001, Kapitel 11) oder *Arkin* (1998) werden Robotersteuerungsarchitekturen in die drei Paradigmen *Reaktiv*, *Deliberativ* und *Hybrid* eingeteilt. Abbildung 2.1 zeigt das Spektrum

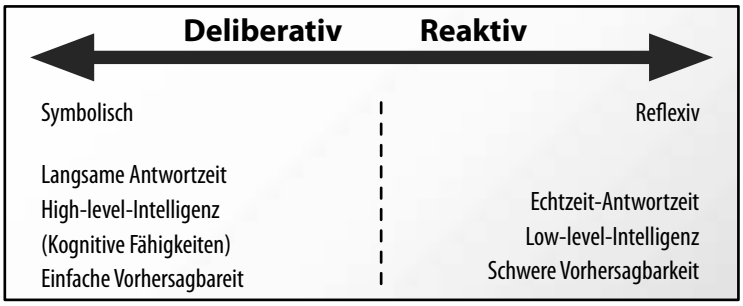


Abbildung 2.1.: Spektrum der Robotersteuerungsparadigmen nach *Arkin* (1998).

der Paradigmen hinsichtlich ihrer Antwortzeit, Intelligenz und Vorhersagbarkeit. Im Folgenden werden die einzelnen Paradigmen genauer vorgestellt.

Deliberatives Paradigma

Das *Deliberative Paradigma* kam schon in der Anfangsphase der Künstlichen Intelligenz auf. In ihm wird eine Wahrnehmungskomponente, eine Entscheidungskomponente und eine Aktionskomponente unterschieden. Die Sensordaten werden in einem ersten Schritt in der Wahrnehmungskomponente benutzt, um ein Modell der Welt zu erstellen. In einem nächsten Schritt wird in der Entscheidungskomponente auf diesem Weltmodell aufbauend ein Plan erzeugt, der dann von der Aktionskomponente in Steuersignale für die Aktorik umgesetzt wird. Ein Zyklus umfasst immer das Durchlaufen all dieser Komponenten in der vorgesehenen Reihenfolge. Da die Modellbildung und die Entscheidungsfindung komplexe Problemfelder sind, hat dieser Zyklus eine langsame Antwortzeit, kann aber mit einem ausreichenden Weltmodell auch komplexe Aufgaben bewältigen. Ein weiterer Vorteil dieses Paradigmas ist die relativ gute Vorhersagbarkeit des Systems. Die klaren Aktionsanweisungen auf dem symbolischen Weltmodell lassen sich leicht ermitteln und somit kann der zukünftige Zustand des Systems auf Basis des aktuellen Weltmodells leicht berechnet werden.

Reaktives Paradigma

Das *Reaktive Paradigma* ist gekennzeichnet durch eine direkte Koppelung der Wahrnehmungskomponente mit der Aktionskomponente. Dieses Prinzip wurde durch *Braitenberg* (1984) in seinem Buch »Vehicles, Experiments in Synthetic Psychology« als eine Reihe von Gedankenexperimenten eingeführt. Durch diese direkte Verbindung können reaktive Systeme in Echtzeit auf neue Sensorinformationen reagieren. Eine direkte Verbindung der Sensorik und Aktorik kann aber nur reflexartige Verhalten von geringerer Intelligenz unterstützen, denn es fehlen die

notwendige Abstraktion und Zustandshaltung, die für höhere Intelligenz notwendig sind. Das Fehlen der Weltmodell-Abstraktion ist allerdings auch ein Vorteil gegenüber den deliberativen Systemen, denn diese haben mit zwei Problemen zu kämpfen, die für reaktive Systeme leicht zu lösen sind:

- Aufgrund ihres komplexeren und rechenaufwendigeren Systems können deliberative Systeme nicht so gut mit sich schnell verändernden Umgebungen zurechtkommen.
- Ist das eingesetzte Weltmodell nicht konsistent, zuverlässig und genau, können Fehler in dem Modell zu schwerwiegenden Fehlentscheidungen der Entscheidungskomponente führen.

Hybrides Paradigma

Hybride Systeme verbinden den Ansatz, der dem deliberativen Paradigma zugrunde liegt mit dem reaktiver Systeme. Diese Verbindung beider Ansätze ist z.B. auch beim Menschen zu beobachten: Auch er hat reaktive Verhaltensausführungen (seine Reflexe) ohne die er nicht lebensfähig wäre. Aber er benötigt genauso auch deliberative, planerische und abstraktere Verhaltensweisen, um bestimmte Probleme lösen zu können. Systeme, die dem hybriden Paradigma folgen, versuchen, die Vorteile der deliberativen und der reaktiven Paradigmen zu vereinen. Sie bestehen meistens aus mehreren Schichten, wobei die verschiedenen Schichten den beiden Grundparadigmen unterschiedlich stark zugeordnet werden können.

2.1.2. Subsumption-Architektur

Abbildung 2.2.: Subsumption-Architektur: Traditionelle Dekomposition der Robotersteuerung in funktionale Komponenten. (Nach *Brooks* 1986, Fig. 1)

Die *Subsumption-Architektur* zur Robotersteuerung wurde von *Brooks* Mitte der 80er Jahre entwickelt und hatte daraufhin einen großen Einfluss auf die nachfolgende Entwicklung von Robotersteuerungsarchitekturen. *Brooks* (1986) analysierte die bis zu diesen Zeitpunkt vorherrschenden deliberativen Ansätze als eine in Abbildung 2.2 gezeigte horizontale Verkettung von funktionalen Komponenten.

Brooks schlug daraufhin einen anderen Ansatz vor: Die Dekomposition der Robotersteuerung basierend auf Verhaltenseinheiten[1] wie sie in Abbildung 2.3 gezeigt ist. Damit hat *Brooks* den ersten Ansatz vorgestellt, der dem reaktiven Paradigma zuzuordnen ist und hat mit diesem Ansatz auch die von *Braitenberg* (1984) inspirierte Sensor-Aktor Kopplung in die moderne Robotik eingeführt.

Brooks (1986) hat durch seine *Subsumption-Architektur* den Begriff der *Verhaltensbasierten Steuerung* geprägt, der nach seiner Arbeit viele moderne Robotersteuerungsarchitekturen zuzuordnen sind.

Abbildung 2.3.: Subsumption-Architektur: Dekomposition der Robotersteuerung basierend auf Verhaltenseinheiten. (Nach *Brooks* 1986, Fig. 2+3)

Die Verhaltenseinheiten sind in mehrere Schichten unterteilt. Die unterste, nullte Schicht stellt ein Basisverhalten – in *Brooks*' Fall das Ausweichen von Hindernissen – zur Verfügung. Erster Schritt der Entwicklung ist es, nur dieses Basisverhalten zu implementieren und zu testen. Nach diesem Schritt existiert ein rudimentärer Roboter, der in folgenden Entwicklungszyklen mit höheren Schichten erweitert werden kann, um komplexere Verhalten, wie etwa das ziellose Umherfahren oder das Kartieren, zu implementieren. Höhere Schichten können dabei:

- auf die Ergebnisse der unter ihnen liegenden Schichten zugreifen und sie gegebenenfalls abändern,
- die Sensordaten der unter ihnen liegenden Schichten unterdrücken, und
- den internen Zustand einer unteren Schicht zurücksetzen.

Brooks (1986) hat vier grundlegende Anforderungen an eine Robotersteuerung identifiziert:

[1] Engl. task-achieving behaviors

1. *Unterstützung vielfältiger Ziele*:[1] Ein Roboter hat oft mehrere Ziele, die sich widersprechen können. Die Robotersteuerung muss mit diesen Widersprüchen zurechtkommen z.B. durch Priorisierung der Ziele.

2. *Unterstützung vielfältiger Sensoren*:[2] Ein Roboter wird sehr wahrscheinlich mehrere Sensoren besitzen, wie z.B. Kameras, Lidar oder Odometriesensoren. Die Daten von den meisten Sensoren sind nicht direkt die gewünschten physikalischen Größen und außerdem sind sie mit Messfehlern behaftet. Unterschiedliche Sensoren können benutzt werden, um die gleichen physikalischen Größen zu messen und es kann vorkommen, dass Sensoren (zeitweise) ausfallen.
   Eine Robotersteuerung muss unter diesen Bedingungen in der Lage sein, Entscheidungen zu treffen.

3. *Robustheit*:[3] Ein Roboter muss robust sein. Er muss in der Lage sein, fehlerhafte Sensoren zu kompensieren und mit sich schnell ändernden Umweltbedingungen zurechtzukommen. Idealerweise sollte er auch mit teilweisen Fehlern in seiner Recheneinheit zurechtkommen.

4. *Erweiterbarkeit*:[4] Wenn dem Roboter mehr Sensoren oder Möglichkeiten hinzugefügt werden, sollte er leicht mit mehr Rechenleistung ausgestattet werden können, um dieses Mehr an Möglichkeiten ausnutzen zu können.

Diese vier Anforderungen werden durch die Subsumption-Architektur erfüllt:

Die erste Anforderung, *mehrere Ziele* parallel verfolgen zu können, wird durch die Aufteilung in Verhaltenseinheiten erreicht. Jede Verhaltenseinheit verfolgt ihr eigenes Ziel, wobei Verhaltenseinheiten höherer Schichten die Ziele von unteren Schichten unterdrücken können. Die zweite Anforderung, Unterstützung *vielfältiger Sensoren*, kann in der Subsumption-Architektur auch oft ohne die vielmals problembehaftete Sensorfusion geschehen, denn in ihr ist es nicht notwendig, eine zentrale Repräsentation der Sensordaten vorzuhalten. Jede Verhaltenseinheit erhält die für sie notwendigen Sensordaten direkt. *Robustheit*, die dritte Anforderung, wird in der Subsumption-Architektur erreicht durch die Unabhängigkeit der unteren Schichten von allen überliegenden. Untere Schichten können auch dann noch rudimentäre Roboterfunktionen aufrechterhalten, wenn andere Schichten ausfallen. Die letzte Anforderung der *Erweiterbarkeit* ist in der Subsumption-Architektur zum einen durch die Möglichkeit des Hinzufügens von höheren Schichten oder weiteren Sensoren ohne niedrigere Schichten zu beeinflussen gegeben. Zum anderen wird die Erweiterbarkeit gefördert durch die relativ schmalbandige Kommunikation zwischen den einzelnen Funktionseinheiten, die dadurch auch leicht auf verschiedene Rechner ausgelagert werden können.

[1] Engl. multiple goals
[2] Engl. multiple sensors
[3] Engl. robustness

Ursprünglich wurden in *Brooks* (1986) die einzelnen Verhaltenseinheiten als Zustandsautomaten realisiert, was aber nur eine mögliche Implementierungsentscheidung darstellt, die die Praktikabilität dieser Architektur unter Beweis stellen soll.

2.1.3. Motor-Schema-Architektur

Arkin (1987) führte einen weiteren verhaltensbasierten Ansatz zur Robotersteuerung ein, die *Motor-Schemas*. Hier wird das Konzept des *Schemas* aus der Psychologie, Neurologie und Gehirnforschung auf die Robotik übertragen.
Ein Schema ist: (*Arkin* 1989, Referenzen zu den ursprünglichen Definitionen siehe dort)

1. Ein Verhaltensmuster *von* Aktionen sowie ein Verhaltensmuster *für* Aktionen.[1]
2. Eine mentale Kodifikation von Erfahrungen, die ein speziell organisierten Weg der kognitiven Perzeption enthält sowie auf komplexe Situationen oder Mengen von Stimuli reagiert.[2]
3. Eine generische Spezifikation eines Rechenagenten.[3]
4. Ein Kontrollsystem, das kontinuierlich das Feedback des zu kontrollierenden Systems beobachtet, um das angemessene Verhaltensmuster für das Erreichen der Ziele des Motor-Schemas zu ermitteln.[4]
5. Ein adaptiver Controller, der eine Identifikationsprozedur benutzt, um die Repräsentation des zu kontrollierenden Objekts zu aktualisieren.[5]

Die drei ersten Definitionen gehen auf Schemas ein, die beiden letzten Definitionen auf Motor-Schemas im Speziellen.

Die Motor-Schema-Architektur ist im Gegensatz zu der Subsumption-Architektur eine kooperative Architektur, in der die einzelnen Motor-Schemas unabhängige Ausgaben (Vektorfelder) erzeugen, die dann von einer zentralen Kooperationsinstanz fusioniert werden (Vektoraddition). Motor-Schemas haben als Eingabe ein oder mehrere *Wahrnehmende Schemas*,[6] die ein

[1] Engl. a pattern *of* action as well as a pattern *for* action.
[2] Engl. a mental codification of experience that includes a particular organized way of perceiving cognitively and responding to a complex situation or set of stimuli.
[3] Engl. a generic specification of a computing agent.
[5] Engl. an adaptive controller that uses an identification procedure to update its representation of the object being controlled.
[6] Engl. perceptual schemas

oder mehrere *Sensorereignisse*[1] beobachten. Um bestmöglich kooperieren zu können, haben die Motor-Schemas eine Kommunikationsmöglichkeit untereinander.

Ein Beispiel aus *Arkin* (1987, p. 267f) soll hier den Informationsfluss, der in Abbildung 2.4 schematisch dargestellt ist, erläutern: Ein Roboter bewegt sich über ein Feld in einer bestimmten Richtung (*Vorwärts*[2] Schema). Das *Finde-Hindernis*[3] Schema sucht konstant nach neuen Hindernissen in den Sensordaten (z.B. Kameradaten), die sich in Richtung des Roboterpfades befinden. Wenn ein Hindernis erkannt wurde, also ein neues Sensorereignis auftrat, wird für dieses Ereignis ein neues wahrnehmendes Schema *Statisches-Hindernis* erstellt. Dieses wahrnehmende Schema ist nur dafür zuständig, die Sensordaten dieser Region zu beobachten. Für andere Sensorereignisse in anderen Regionen des Bildes werden entsprechend neue Instanzen von *Statisches-Hindernis* erstellt. Zusätzlich zum Erstellen von *Statisches-Hindernis* wird auch noch das Motor-Schema *Vermeide-Statisches-Hindernis* erzeugt, das bei hinreichender Gewissheit ein Vektorfeld generiert, welches diesem Hindernis ausweicht. Beide Schemas, das wahrnehmende und das Motor-Schema, werden zerstört, falls sich das Hindernis als Phantom erweist oder es aus dem beobachteten Bereich verschwindet.

Kommunikation zwischen den Motor-Schemas ist notwendig, wenn die rudimentäre Kooperation der Fusion der Ausgaben der Schemas (Vektoraddition von Vektorfeldern) nicht zu einem gewünschten Ergebnis führt. Hier nimmt *Arkin* das Beispielszenario eines blockierten Gehwegs zur Veranschaulichung: Der Roboter vom vorherigen Beispiel wurde instruiert, in einer bestimmten Richtung einem Gehweg zu folgen und Hindernissen auszuweichen. Angenommen, der Gehweg ist komplett durch ein Hindernis blockiert; der Roboter wird dann irgendwann stehen bleiben, weil durch die Vektoraddition seine Sollgeschwindigkeit auf null sinkt. Das *Bleib-auf-Weg*[4] Schema erkennt dies durch Inter-Schema-Kommunikation mit dem Fusionsschema und kann daraufhin temporär sein Vektorfeld so anpassen, dass ein Passieren des Hindernisses jenseits des Gehwegs möglich wird. Sobald ihm die Vektorkraft von *Vermeide-Statisches-Hindernis* anzeigt, dass das Hindernis passiert ist, kann *Bleib-auf-Weg* sein ursprüngliches Vektorfeld wiederherstellen und so den Roboter wieder auf den Gehweg leiten.

Arkin (1998) stellt die folgenden fünf Punkte zusammen, in denen sich die Motor-Schema-Architektur von anderen Ansätzen unterscheidet:

- Die Verhalten haben eine einheitliche Ausgabe (Vektorfeld).
- Die Verhalten kooperieren durch Fusion dieser Ausgaben (Vektoraddition) oder durch Kommunikation untereinander.
- Es besteht keine feste Hierarchie. Die Schemas bilden ein temporäres Netzwerk, das sich kontinuierlich der Umgebung, den Zielen des Roboters, oder seinen zur Verfügung stehenden Ressourcen anpasst.

[1] Engl. sensory events
[2] Engl. move ahead
[3] Engl. find-obstacle
[4] Engl. stay-on-path

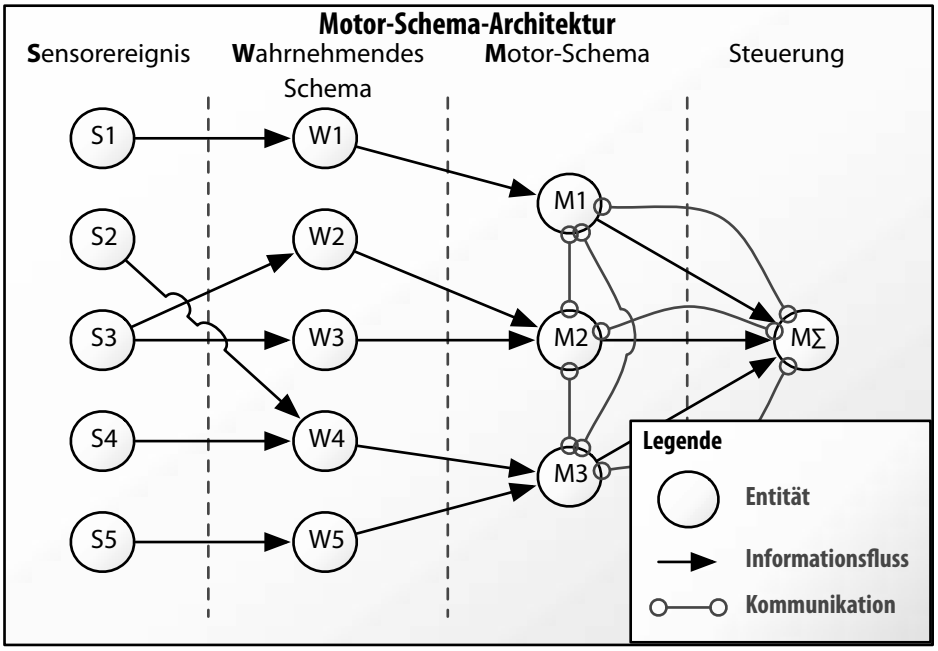


Abbildung 2.4.: Motor-Schema-Architektur: Der Informationsfluss zwischen den Schemas und Sensorereignissen.

- Mehrere Verhalten tragen zu dem Ergebnis der Fusion bei und so ist es nicht leicht zu sagen, welches Verhalten welche Aktion provoziert hat.
- Die Unsicherheit der Wahrnehmung kann in die Ausgabe der Verhalten mit einfließen.

2.1.4. EMS-Vision/4D-Ansatz

Am Institut für Systemdynamik und Flugmechanik (ISF) der Universität der Bundeswehr München wurde eine hierarchische, verhaltensbasierte, hybride Architektur zur autonomen Fahrzeugführung entwickelt (*Pellkofer* 2003; *Siedersberger* 2003). Abbildung 2.5 zeigt eine auf die Verhaltensauswahl fokussierte Übersicht über die Architektur und ihre Komponenten:

- *Missionsplanung*
  Die Missionsplanung generiert eine Liste von weitreichenden Missionszielen und hält diese für die Verhaltensentscheidung vor.
- *Szenenbaum*
  Der Szenenbaum hält in einer baumartigen Struktur 3D-Objekte und ihnen zugeordnete Zustandsgrößen vor.

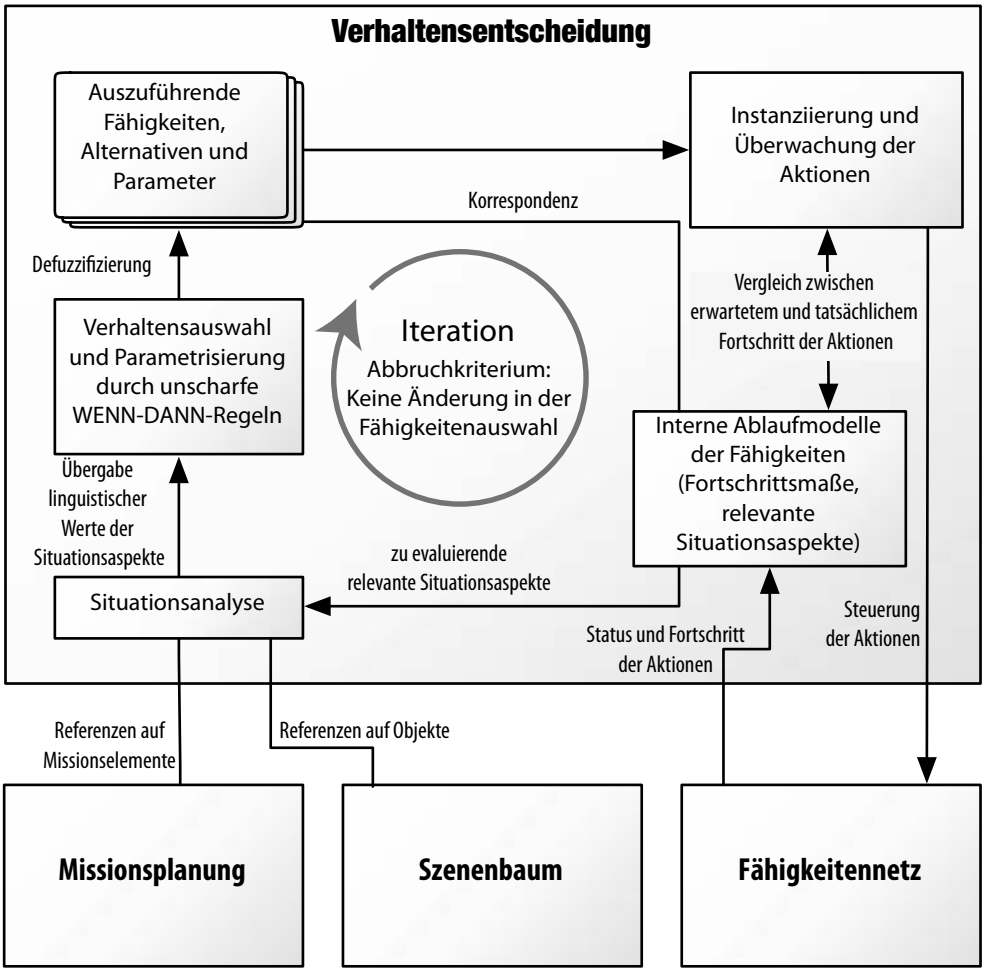


Abbildung 2.5.: Die am ISF verwendete Architektur zur Verhaltensentscheidung. (nach *Pellkofer* 2003, Abb. 3.1)

- *Verhaltensentscheidung*
  Die Verhaltensentscheidung enthält in einem iterativen Prozess eine Situationsanalyse und Verhaltensauswahl. Durch die iterative Verfeinerung werden nur solche Situationsaspekte ausgewertet, die auch wirklich von den Verhalten benötigt werden. Die Iteration wird abgebrochen, wenn sich durch die Anwendung der unscharfen WENN-DANN-Regeln auf den Situationsaspekten keine neuen aktiven Verhalten ergeben. Diese aktiven Verhalten, die aus dem dynamisch veränderbaren Fundus des Fähigkeitennetzs stammen, werden von der Verhaltensentscheidung erzeugt und überwacht.

- *Fähigkeitennetz*
  Das Fähigkeitennetz ist ein Netzwerk der aktuell verfügbaren Aktionen. Das Netzwerk ist

hierarchisch geordnet, wobei mit steigender Hierarchieebene die Entscheidungskompetenz und der Abstraktionsgrad des bei der Entscheidung verwendeten Wissens zunimmt. Jede Fähigkeit in diesem Netzwerk, auch Experte genannt, besitzt ein Fortschrittsmaß, mit dem überliegende Experten und Module sich ein symbolisches Bild der Situation machen können.

2.1.5. *Darvin* und Situationsgraphenbäume

In Karlsruhe wurde unter Prof. Hans-Hellmut Nagel am Fraunhofer Institut für Informations- und Datenverarbeitung (IITB) die Versuchsplattform *Darvin*[1] entwickelt. Sie ist mittlerweile in der dritten Generation. Auf dieser Plattform werden Algorithmen entwickelt, die im Rahmen von Fahrerassistenzsystemen den Fahrer unterstützend zur Seite stehen. Das Ziel einer vollständig autonomen Fahrzeugführung wird momentan nicht verfolgt.

Über eine Kombination von Bildfolgenauswertung mit logischer Auswertung auf begrifflicher Ebene soll dem System ein Verständnis für die aktuelle Verkehrssituation vermittelt werden (DARVIN 2008). Als Verhaltensbeschreibungsverfahren kommen Situationsgraphenbäume zum Einsatz (*Gerber* 2000; *Nagel* und Arens 2003, 2005). Situationsgraphenbäume setzen sich aus Situationsgraphen zusammen, die das Wissen über eine erwartete zeitliche Abfolge von Situationen beschreiben. Situationsgraphen wiederum bestehen aus Situationsschemata – generische Beschreibungen des Zustands eines Verkehrsteilnehmers – und Prädiktionskanten zwischen diesen Situationsschemata. Durch Spezialisierung werden Situationsgraphen zu Situationsgraphenbäumen verbunden.

2.1.6. Biologisch motivierte Verhaltensnetzwerke

Wie die Motor-Schemas sind auch die *biologisch motivierten Verhaltensnetzwerke* eine Entwicklung basierend auf Erkenntnissen der Biologie. *Albiez* (2006) hat die Lokomotionskontrolle in der Natur studiert und zusammen mit Erkenntnissen aus der Neurologie daraus eine technische Umsetzung abgeleitet. Er schließt aus dieser Untersuchung die folgenden fünf Thesen für ein Verhaltensnetzwerk, das sich biologische Erkenntnisse zunutze macht: (*Albiez* 2006, Kapitel 5, Erläuterungen eingefügt)

1. *Verwendung einer einheitlichen Schnittstelle für die einzelnen Verhaltensbausteine.*

2. *Anordnung dieser Bausteine in einer schwachen und zeitlich veränderbaren, das heißt temporalen, Hierarchie.*
   Albiez hat den Begriff der *Hierarchie* formaler definiert, hier mag die Erklärung genügen, dass ein schwach-temporalhierarchisches Verhaltensnetzwerk ein in Ebenen geordnetes Netzwerk ist, das auch direkte Verbindungen der Verhaltensbausteine von höheren

[1] Abk. Driver Assistance using Realtime Vision for INnercity areas.

Ebenen in niedere erlaubt und dabei die Ebenen zwischen ihnen überspringt (*schwach*). Außerdem ist es kein fixes Netzwerk, sondern die Verbindungen zwischen den Verhaltensbausteinen ist zeitlich veränderbar (*temporal*).

3. *Verwendung einer kinematotopischen Topologie.*
   Die horizontale Anordnung innerhalb einer Ebene des Netzwerks sollte sich an der Kinematotopie ausrichten. D.h. sie sollte sich an der Kinematik des zu steuernden Robotersystems ausrichten.

4. *Nutzung der Arbeit der einzelnen Verhaltensbausteine zur Koordination und zur Bestimmung des Fokus'.*
   Um auch in großen Netzwerken und komplexen Robotersteuerungen noch eine effektive Koordination der Verhalten zu erreichen, sollte sich der Fokus des Netzwerks auf die (unscharfe) Untermenge der zu einer Aufgabe benötigten Bausteine setzen können.

5. *Verwendung der internen Zustände der Verhalten als propriozeptive Signale.*
   Interne Zustände einzelner Bausteine sollten genau so wie Sensordaten den anderen Bausteinen als Eingangsquelle dienen können.

Im Folgenden wird die auf diese Thesen aufbauende verhaltensbasierte Robotersteuerung skizziert, die *Albiez* für 4- und 6-beinige Laufmaschinen entwickelt hat. Dafür wird zuerst ein einzelner Baustein des Verhaltensnetzwerks genauer betrachtet, um danach auf die Kooperation unter ihnen einzugehen und abschließend die Besonderheiten der verwendeten Netzwerkstruktur zu erläutern.

Verhaltensbaustein

Ein Verhaltensbaustein im Sinne eines biologisch motivierten Verhaltensnetzwerks ist in *Albiez* (2006, Definition 6.1) definiert als:

Verhaltensbaustein *Ein* Verhaltensbaustein *ist die funktionale Grundeinheit der Verhaltensnetzwerke. Er wird beschrieben als ein Sechstupel der Form*

$$\begin{aligned} V &= (\vec{e}, \vec{u}, \iota, \hat{r}, \hat{a}, F) \\ mit: \vec{u} &= F(\vec{e}, \iota) \\ a &= \hat{a}(\vec{e}, \vec{u}) \\ r &= \hat{r}(\vec{e}) \end{aligned}$$

wobei r als Reflexion*, a als* Aktivität*, ι als* Motivation *und F als* Übertragungsfunktion *bezeichnet werden. $\vec{e} = e_1, \ldots, e_n$ umfasst alle sensorischen Eingaben des Verhaltens und $\vec{u} = u_1, \ldots, u_m$ bezeichnet die vom Verhalten generierten Aktorausgaben.*

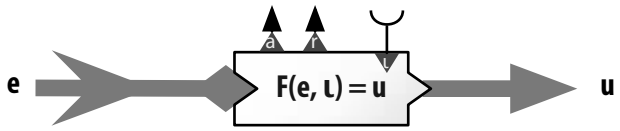


Abbildung 2.6.: Biologisch motivierte Verhaltensnetze: Das Schema eines Verhaltensbausteines, wie es in den folgenden Abbildungen verwendet wird. a = Aktivität, r = Reflexivität, ι = Motivation, e = Eingangsdaten, u = Ausgangsdaten

Die Verhaltensbausteine, im Folgenden auch *Verhalten* genannt, besitzen mit dieser Definition eine einheitliche Schnittstelle, durch die sie leichter in ein Netzwerk integriert werden können. Der Hauptbestandteil eines Verhaltens ist seine Übertragungsfunktion F, die anhand der Eingaben $\vec{e}$ und der Motivation ι (welche als spezielle Eingabe aufgefasst werden sollte) die Aktorausgaben $\vec{u}$ berechnet. Die Motivation ι ist ein standardisierter Eingang, den alle Verhalten besitzen. Wobei $\iota \in [0, 1]$. ι gibt den Grad der gewünschten Aktivierung an: $\iota = 0$ bedeutet das das Verhalten nicht aktiv sein soll. $\iota = 1$ bedeutet maximale Aktivierung.

Die Reflexion $r \in [0, 1]$ eines Verhaltens wird auch Zielbewertung genannt. Je niedriger dieser Wert, desto zufriedener ist das Verhalten mit der aktuellen Situation.

Die Aktivität $a \in [0, 1]$ eines Verhaltensbausteins spiegelt die Quantität der Änderung wider, die ein Verhalten vornimmt. Ein Wert von 0 bedeutet keine, ein Wert von 1 maximale Änderung.

Bei den meisten Verhalten gibt es eine Korrelation von Reflexion und Aktivität in der Art, dass ein hoher Wert für die Reflexion auch einen hohen Wert für die Aktivität induziert und umgekehrt ein niedriger Reflexionswert eine niedrige Aktivität verursacht.

Es sind aber auch Verhalten denkbar, die auch bei geringer Reflexion (also hoher Zufriedenheit) eine hohe Aktivität aufweisen, weil sie zur Erhaltung des gewünschten Zustands viel Aufwand aufbringen müssen.
Albiez nennt Gangart-Verhalten eines Laufroboters als Beispiel, denn diese Verhalten müssen immer eine hohe Aktivität aufweisen, um die Beine eines Laufroboters zu bewegen.

Die Aktivität a eines Verhaltens besitzt natürlich eine Korrelation mit der Motivation ι des gleichen Verhaltens, da ι die gewünschte Aktivierung angibt. Aus einer Motivation $\iota = 0$ resultiert also eine Aktivität $a = 0$. Die Reflexion r eines Verhaltens ist dagegen unabhängig von seiner Motivation ι – sie spiegelt auch bei deaktiviertem Verhalten die Zufriedenheit des Verhaltens mit der aktuellen Situation wider.

Koordination der Verhaltensbausteine

Der vorherige Abschnitt hat einen elementaren Baustein des Netzwerks vorgestellt, dieser Abschnitt befasst sich nun mit der Koordination der Verhaltensbausteine untereinander.

Eine Koordination der Verhalten ist notwendig, weil es mehrere Verhalten geben kann, die Ausgaben für die gleichen Aktoren erzeugen. Ohne bewusste Koordination dieser Verhalten würde der Aktor zum Teil gegensätzliche und nicht sinnvolle Steuersignale erhalten. Grundsätzlich können Verhalten auf zwei unterschiedliche Arten zusammenarbeiten, die beide von dem Verhaltensnetzwerk unterstützt werden sollten, da sie jeweils unterschiedliche Einsatzszenarien darstellen:

- *Kooperative Koordination*
  Die zu koordinierenden Verhalten tragen alle zu dem Ergebnis bei. Eine hier nicht näher spezifizierte Fusion der Aktorausgaben ist für eine kooperative Koordination notwendig.

- *Kompetitive Koordination*
  Eines der zu koordinierenden Verhalten erlangt die Überhand und ist der alleinige Beitragende zu dem Ergebnis.

In *kompetitive Koordination* können wiederum zwei Spezialfälle für die Koordination der Zusammenarbeit zweier Verhalten der gleichen semantischen Hierarchieebene identifiziert werden:

- *Komplementäre Verhalten*

- *Redundante Verhalten*

Auf diese beiden Spezialfälle wird später eingegangen – dort wird auch genauer spezifiziert, was komplementär und redundant in diesem Zusammenhang heißt.

Fusionsknoten Der Ansatz von *Albiez* führt einen gesonderten Baustein ein, der die beiden Koordinationsarten beherrscht: den *Fusionsknoten*. Abbildung 2.7 stellt diesen und seine Verbindung zu den Verhaltensbausteinen grafisch dar.

Ein Fusionsknoten verbindet die Ausgaben von zwei oder mehr Verhaltensbausteinen, welche Ausgaben für denselben Aktor generieren. Er erhält die Aktorausgaben aller zu koordinierenden Verhaltensbausteine und deren Aktivität als Eingabe und berechnet aus diesen eine fusionierte Aktorausgabe.

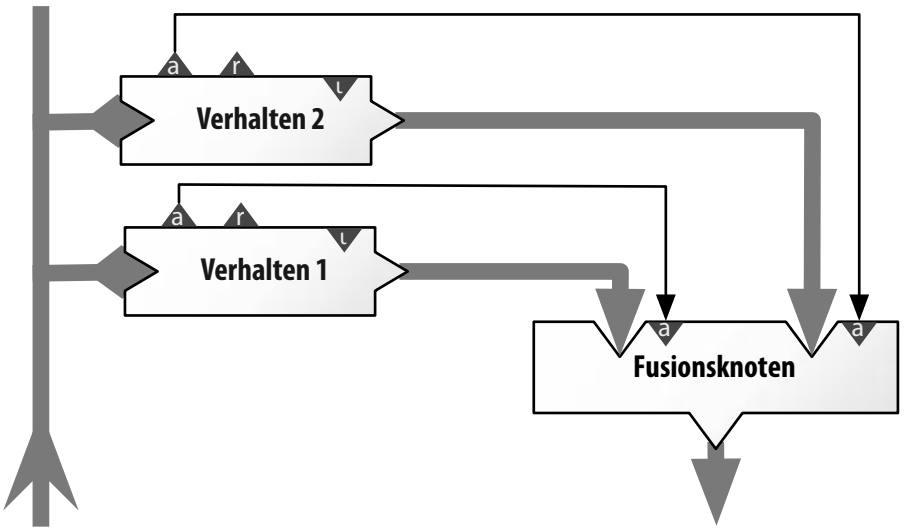


Abbildung 2.7.: Fusionsknoten zum Zusammenführen von Verhaltensausgaben. Fusionskriterium ist die Aktivität der beteiligten Verhalten.

Es werden passive Knoten (*Fusionsknoten erster Ordnung*) und aktive Knoten (*Fusionsknoten zweiter Ordnung*) unterschieden: Ein passiver Fusionsknoten besitzt selbst keinen eigenen Aktivitätswert und kann deshalb nur als direkter Eingang eines Aktors dienen und nicht weiter im Netzwerk verschaltet werden. Ein Fusionsknoten zweiter Ordnung stellt eine fusionierte Aktivität zur Verfügung und kann somit z.B. wieder als Eingang eines anderen Fusionsknoten dienen.

Kopplung von Verhalten Eine weitere Koordinationsmöglichkeit ist die *Kopplung* von Verhalten: Zwei Verhalten sind gekoppelt, wenn sie sich durch ihre Ausgaben wechselseitig beeinflussen. Gekoppelte Verhalten stehen dabei auf derselben semantischen Hierarchiestufe. Das beschränkt die Kopplung auf die beiden oben erwähnten Spezialfälle der komplementären und redundanten Kopplung.

Komplementär sind zwei Verhalten, wenn sich deren Funktion gegenseitig ausschließt. Die komplementäre Kopplung ist in dem Netzwerk kenntlich dadurch, dass die Aktivität des einen Verhaltens ein Sensoreingang des anderen Verhaltens ist.

Redundant ist ein Verhalten, wenn es die gleiche Funktion erfüllt, wie ein anderes Verhalten. Dieses redundante Verhalten erhält als zusätzliche Eingabe die Aktivität und Reflexion des anderen Verhaltens und kann somit die Funktion bei Bedarf übernehmen.

Kopplungen sollten sparsam eingesetzt werden, denn sie widersprechen dem Komponentengedanken eines Verhaltensbausteins: Der Baustein kann dann nicht mehr eigenständig entworfen werden sondern es müssen Wechselbeziehungen zwischen Bausteinen der gleichen semantischen Hierarchiestufe beachtet werden.

Netzwerk

Einzelne Bausteine des Verhaltensnetzwerks und Kooperationen zwischen ihnen wurden eingeführt. Dieser Abschnitt behandelt die Netztopologie und es werden einige Begriffe erklärt und generalisiert, die in den vorherigen Abschnitten als Vorgriffe benutzt wurden.

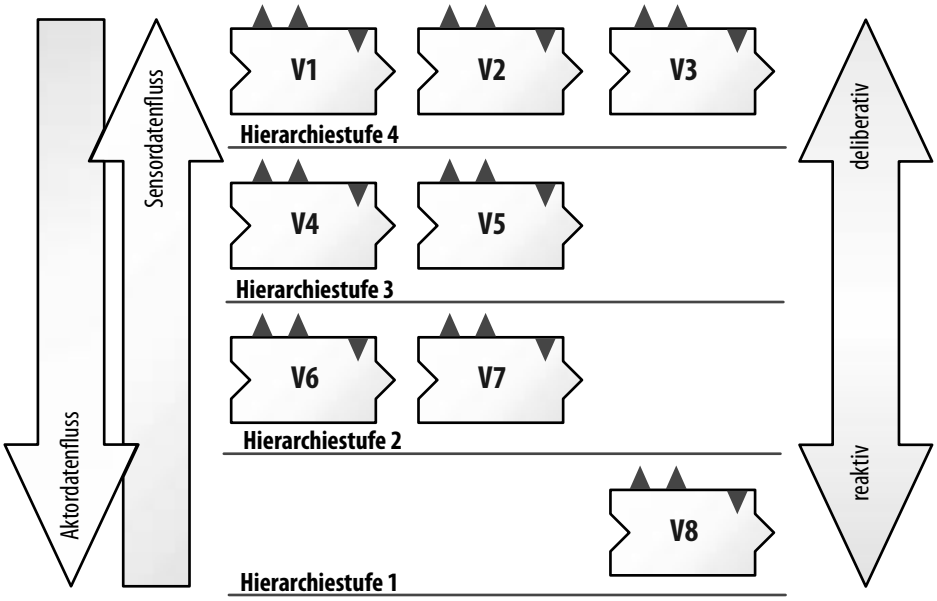


Abbildung 2.8.: Verteilung von mehreren Verhalten V_i auf Hierarchiestufen und Visualisierung des Datenflusses und der Reaktionseigenschaft im Netzwerk.

Ein *Verhaltensnetzwerk* ist die Anordnung mehrerer Verhalten in einen hierarchischen, gerichteten Netzwerkgraphen. Die Knoten des Graphen sind die Verhaltensbausteine und Fusionsknoten, die Kanten des Graphen repräsentieren den Datenfluss zwischen ihnen. Sensordaten-Kanten sind von unten nach oben gerichtet und Aktordaten-Kanten von oben nach unten. Der Aufbau des Graphen wird als Topologie bezeichnet. Die Hierarchien ergeben sich aus der Ähnlichkeit der Semantik der Verhalten, wobei die Hierarchien nach dem Grad der Reaktionseigenschaft ihrer Verhalten[1] sortiert sind. Abbildung 2.8 veranschaulicht dies.

[1]Die Reaktionseigenschaft eines Verhaltens reicht von reaktiv bis deliberativ.

Wenn im vorherigem Text »Sensor« und »Aktor« erwähnt wurde, ist natürlich auch die in der Robotik gebräuchliche Bedeutung dieser Begriffe gemeint. Im Rahmen eines Verhaltensnetzwerks nach *Albiez* sind diese Begriffe aber weiter gefasst: Die Reflexion und Aktivität eines Verhaltens werden als *virtuelle*[1] *Sensoren* aufgefasst. Die Motivation als *virtueller Aktor*.

Erst mit diesen erweiterten Begriffen der Sensoren und Aktoren ist eine Hierarchisierung in semantischen Ebenen überhaupt möglich. Ohne sie müssten z.B. höher liegende Verhalten auch direkte Aktorsignale als Ausgabe besitzen, was die Verhalten und ihre Vernetzung unnötig kompliziert machen würde und auch nicht dem Abstraktionsgrad der Funktionen dieser Verhalten entspräche.

In einem biologisch motivierten Verhaltensnetzwerk bleiben die Verbindungen zwischen den Netzwerkknoten – also die Netzwerkstruktur an sich – konstant. Die temporäre Veränderung des Netzwerks wird durch die dynamische (De-)Motivation der Verhalten erreicht. Dies hat den Vorteil, dass alle möglichen Abhängigkeiten der Verhalten untereinander explizit in der Netzwerkstruktur enthalten sind; sie sich also nicht in der Implementierung der einzelnen Verhalten verstecken.

2.1.7. Biologisch motiviertes Verhaltensnetzwerk zur Steuerung eines kognitiven Automobils

Hoffmann (2007) hat die in Abschnitt 2.1.6 vorgestellte Architektur auf kognitive Automobile übertragen und angepasst. Er verzichtet auf die in *Albiez* (2006, siehe Abschnitt 2.1.6) eingefügte kinematotopische Topologie des Netzwerks und lehnt sich bei der semantischen Hierarchiesierung der Verhalten an die folgende, in der Fahrerverhaltensmodellierung[2] gebräuchlichen (*Michon* 1985), Struktur an:

- *Reaktive Schicht*
  Alle Verhalten, die *Fertigkeiten* oder *Handlungsprimitive* abbilden, stellen die unterste, die reaktive Schicht des Verhaltensnetzwerks dar. Verhalten auf dieser Schicht haben eine hohe Reaktivitätseigenschaft. D.h. sie stellen eher reflexartige, von einem menschlichen Fahrer unbewusst ausgeführte, Verhalten dar. Ein Beispiel für eine solches Verhalten ist das Ausweichen bei einem plötzlich auftauchenden Hindernis. Ein menschlicher Fahrer macht dies reflexartig ohne zu überlegen.

- *Taktische Schicht*
  Die Verhaltensbausteine der *Taktischen Schicht* – auch *Handlungsfähigkeiten* oder *Handlungen* genannt – besitzen einen höheren Zeithorizont und ihre Reaktivitätseigenschaft

[1] Die Sensoren und Aktoren werden als virtuell bezeichnet, weil sie nicht physisch präsent sind.
[2] Engl. driver behavior modelling

ist deliberativer als die der reaktiven Schicht. Beispielsweise ist ein Verhalten zum Umfahren eines statischen Hindernisses hier angesiedelt. Auch für einen menschlichen Fahrer ist das Umfahren eine bewusste Entscheidung, die erst nach Einschätzung der aktuellen Lage durchgeführt wird und in mehrere Phasen unterteilt werden kann.

- *Strategische Schicht*
  Verhalten dieser Schicht werden *Verhaltensfähigkeiten* oder *Verhaltensweisen* genannt. Dies sind deliberative Verhalten, die in einem großen Zeithorizont und einer hohen Abstraktionsstufe agieren. Diese Verhalten setzen die Verkehrsregeln durch, indem sie die unterliegenden Verhalten entsprechend (de-)motivieren. Diese Verhalten werden von außen angestoßen und erhalten als Eingabe typische GPS[1]-Navigationssystem-Anweisungen wie „nächste Möglichkeit links einbiegen“ oder „der Straße folgen“. Da ein Navigationsystem nur immer eine dieser Anweisung zur Zeit stellt, sind alle Verhalten auf dieser Ebene komplementär. D.h. nur eines kann zur einem bestimmten Zeitpunkt aktiv sein.

Zusätzlich zu dieser Unterteilung führt *Hoffmann* (2007) noch *Zwischenschichten* ein, die bei Bedarf zwischen die Hauptschichten eingefügt werden können. Dies ist notwendig, da in einem biologisch motivierten Netzwerk nach *Albiez* (2006) Verhalten auf der gleichen Ebene nur über die im vorherigen Abschnitt erklärten Kopplungen miteinander verbunden sein können, *Hoffmann* aber z.B. Verhalten der taktischen Ebene identifiziert hat, die auf andere Verhalten der selben Ebene einwirken müssen, um ihr Ziel zu erreichen.

Fusionsarten

Hoffmann geht auf die folgenden Fusionsarten, als speziell für den Anwendungsbereich in einem kognitiven Automobil angepasst, ein:

- *Beschränkte Summenfusion*
  Bei dieser Fusion werden alle Eingabewerte aufsummiert. Negative Werte sind zulässig. Ist diese Summe kleiner als das Minimum der Werte oder größer als das Maximum, wird sie auf den jeweiligen Wert beschränkt.

- *Maximumfusion*
  Der maximale Eingabewert wird als Ausgabe gewählt.

- *Minimumfusion*
  Der minimale Eingabewert wird als Ausgabe gewählt.

[1] Global Positioning System

- *Master-Slave-Fusion*
  Ein Eingangsverhalten ist der *Master*. Sollte der Aktivitätswert dieses Verhaltens von Null abweichen, so wird der Eingabewert dieses Verhaltens als Ausgabe gewählt. Ist der Wert Null, wird der Eingabewert des *Slave*-Verhaltens durchgereicht. Diese Art der Fusion ist vor allem zur Anbindung von sicherheitsrelevanten Verhalten, deren Ausgaben alle Ausgaben von anderen Verhalten überschreiben sollen, sinnvoll.

Topologie

Abbildung 2.9 zeigt die Topologie des Verhaltensnetzwerks, die *Hoffmann* (2007) für ein kognitives Automobil entwickelt hat und auf die in dieser Arbeit aufgebaut wird. Eine genauere Betrachtung der einzelnen Verhalten und ihrer Beziehungen untereinander soll hier aus Platzgründen entfallen. In Kapitel 4 werden unter anderem die Änderungen, die an dieser Struktur vorgenommen wurden, erläutert.

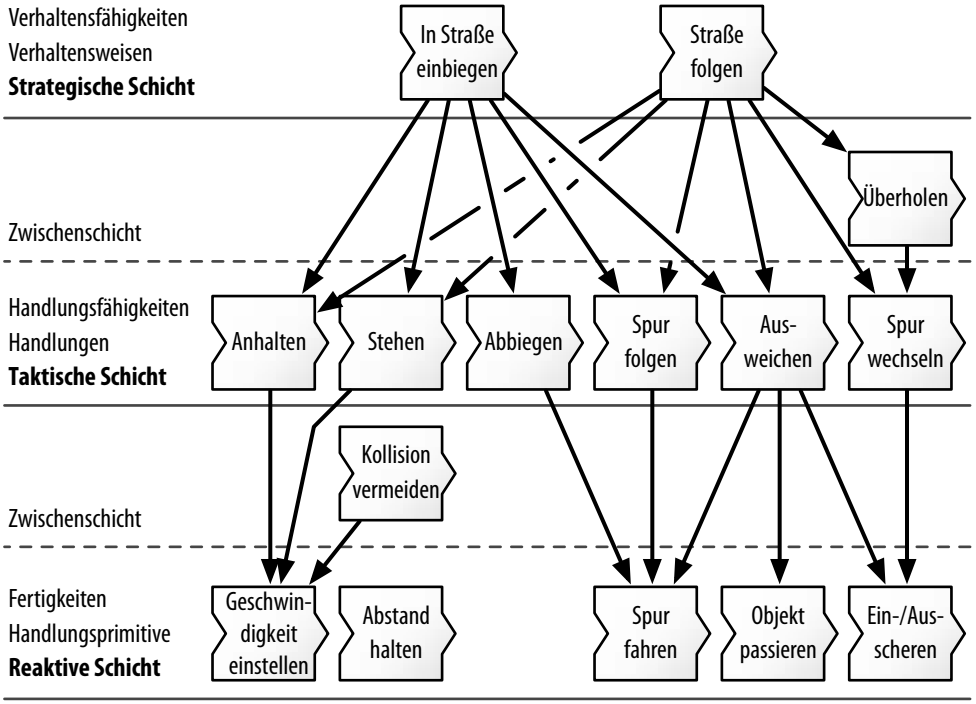


Abbildung 2.9.: Topologie des biologisch motivierten Verhaltensnetzwerkes zur Steuerung eines kognitiven Automobils nach *Hoffmann* (2007). Eine Verbindung zwischen zwei Verhalten drückt eine Kann-benutzen-Beziehung aus.

2.2. Hierarchische Zustandsautomaten

In diesem Abschnitt soll aus zwei Gründen auf hierarchische Zustandsautomaten nach *Harel* (1987) eingegangen werden: Zum einen als Beispiel einer Methode, ein einzelnes Verhalten eines Verhaltensnetzwerks zu implementieren, und zum anderen, weil hierarchische Zustandsautomaten die Grundlage für den Planungsteil des Team-AnnieWay-Ansatzes darstellen auf den im Kapitel 6 noch genauer eingegangen wird.

Ein endlicher Zustandsautomat ist ein gerichteter Graph, der ein dynamisches Verhalten formal grafisch beschreibt. Hier soll auf die UML-Notation (*Booch* u. a. 1997a, b) eingegangen werden, in der die Knoten im Graphen Zustände und die Kanten die Übergänge zwischen ihnen beschreiben. Die Kanten sind mit dem auslösenden Ereignis und mit eventuellen einschränkenden Bedingungen annotiert.

Abbildung 2.10 zeigt einen solchen einfachen endlichen Zustandsautomaten für ein fiktives Gerät, dessen alleinige Funktion es ist, kaputt zu gehen, sobald es im Regen steht. Ein ausgefüllter Kreis kennzeichnet den Startzustand, ein ausgefüllter Kreis umgeben von einem anderen Kreis kennzeichnet den Endzustand. Ein Pfeil ohne Annotation stellt einen spontanen Zustandsübergang dar. So wird z.B. in Abbildung 2.10 sofort am Anfang vom Startzustand in den Zustand *Funktionsfähig* gewechselt. Die Annotation eines Pfeils enthält zuerst das auslösende Ereignis („es regnet“) und optional in Klammern eine einschränkende Bedingung, die erfüllt sein muss, damit dieser Übergang ausgewählt werden kann („Gerät im Freien“).

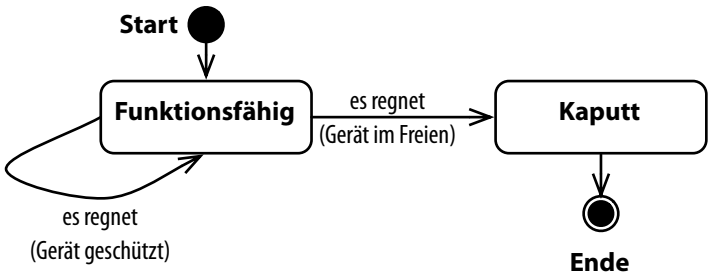


Abbildung 2.10.: Einfaches Beispiel eines endlichen Zustandsautomaten in UML-Notation mit zwei Zuständen und einem Ereignis.

Diese einfache Methode wird bei größeren Systemen sehr schnell unübersichtlich. *Harel* (1987) hat deswegen mehrere Erweiterungen vorgeschlagen, die es erlauben, auch komplexere Systeme mit endlichen Automaten zu beschreiben. Von ihnen sollen hier die zwei wichtigsten vorgestellt werden, die auch ihren Einzug in die UML-Notation schafften: Hierarchisierung und Nebenläufigkeit.

2.2.1. Hierarchisierung

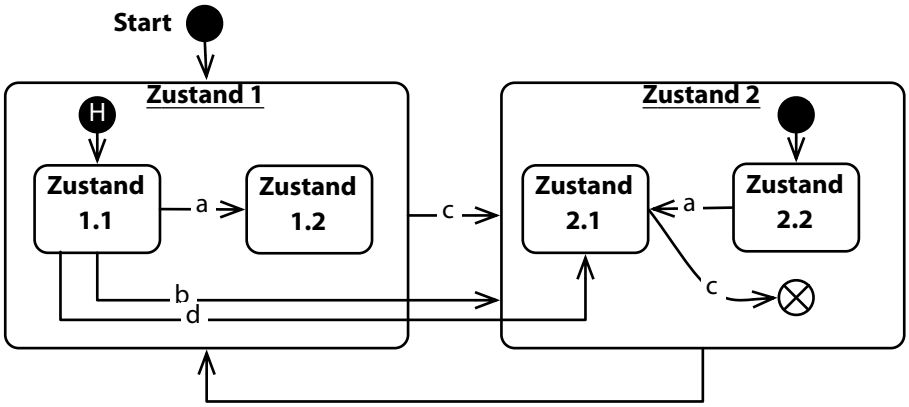


Abbildung 2.11.: Beispiel eines hierarchischen Zustandsautomaten in UML-Notation.

Abbildung 2.11 zeigt ein Beispiel, das alle zur Hierarchisierung notwendigen Konstrukte enthält. Hauptunterschied ist, dass ein Zustand mehrere andere Zustände enthalten kann. Übergänge können alle Zustände jeder Verschachtellungsebene verbinden. So existiert z.B. ein Übergang zwischen *Zustand 1.1* und *Zustand 2* genauso wie zu *Zustand 2.1*.

Jeder Zustand mit Unterzuständen besitzt einen Startzustand, der aktiviert wird, wenn der Überzustand aktiviert wird. So wird z.B. im *Zustand 1.1* beim Ereignis b zum *Zustand 2* gewechselt. Da *Zustand 2* ein verschachtelter Zustand ist, wird in ihm der Startzustand aktiviert, der wiederum mit einem spontanen Übergang in *Zustand 2.2* wechselt.

Ein X in einem Kreis kennzeichnet einen hinausführenden Zustand innerhalb eines verschachtelten Zustands. Wird dieser Zustand aktiviert, so wird der Überzustand verlassen. In der Abbildung 2.11 wird beispielsweise von *Zustand 2.1* mit dem Ereignis c der komplette *Zustand 2* verlassen. Aufgrund des spontanen Übergangs zwischen *Zustand 2* und *Zustand 1* wird dieser nun wieder aktiviert.

Ein Startzustand, der mit einem H gekennzeichnet ist, enthält eine Historie des letzten Zustands, der vor dem Verlassen des Überzustands aktiv war und aktiviert diesen Zustand anstatt den Startzustand. Zur Veranschaulichung betrachten wir in Abbildung 2.11 die Ereignisfolge $a\ c\ a\ c$. Das Ereignis a wechselt von dem zum Anfang aktiven *Zustand 1.1* in *Zustand 1.2*. Das Ereignis c wechselt zu *Zustand 2*. Da der *Zustand 1* verlassen wird, speichert der spezielle Historien-Startzustand den aktuellen Unterzustand *Zustand 1.2*. Die Ereignisse a und c lassen den Automaten wieder in den *Zustand 1* wechseln. Nun wird aber nicht, wie beim ersten Mal, *Zustand 1.1* aktiviert sondern der gespeicherte *Zustand 1.2*.

2.2.2. Nebenläufigkeit

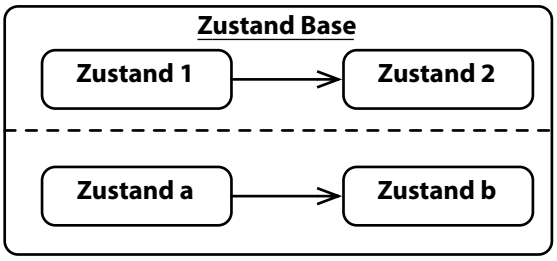


Abbildung 2.12.: Beispiel eines nebenläufigen Zustandsautomaten in UML-Notation.

In einem komplexeren System können oft Zustandsgruppen identifiziert werden, die orthogonal zueinander sind. D.h. die unterschiedlichen Gruppen beeinflussen sich nicht gegenseitig. Abbildung 2.12 zeigt zwei solche nebenläufige Gruppen. Die erste Gruppe wird durch *Zustand 1* und *Zustand 2* gebildet. Die zweite Gruppe durch *Zustand a* und *Zustand b*. Eine gestrichelte Linie trennt die beiden Gruppen grafisch voneinander in ihrem gemeinsamen Überzustand *Zustand Base*. Dieser Überzustand besitzt zwei aktive Unterzustände und zwei Startzustände, in jeder seiner Zustandsgruppen einen.

2.3. Kognitive Automobile

Die vorliegende Arbeit befasst sich mit kognitiven Automobilen, die als Weiterentwicklung mobiler Roboter angesehen werden können. Zunächst wird in diesem Abschnitt ein kurzer historischer Überblick gegeben und dann im Folgenden auf verschiedene Realisierungen kognitiver Automobile im Rahmen der DARPA Urban Challenge 2007 näher eingegangen.

2.3.1. Historischer Überblick

Als einer der ersten mobilen Roboter kann der im *Stanford Research Institute* entwickelte *Shakey* (*Nilsson* 1969) angesehen werden. Dieser brachte wertvolle Erkenntnisse im Bereich der Computervision und Robotersteuerung. Shakey war mobil im Laborumfeld und konnte über seine Umwelt Schlussfolgerungen ziehen und so beispielsweise farblich markierte Objekte erkennen und verschieben. Dazu besaß er eine Kamera, einen optischen Distanzmesser und Berührungssensoren. Er war über Funk mit leistungsfähigeren Computern verbunden, um die nötige Rechenleistung für seine Aufgaben zu erhalten. Bei der Entwicklung von Shakey wurden einige Konzepte und Algorithmen entwickelt, die auch heute noch breite Anwendung finden. So wurde z.B. der A*-Graphen-Suchalgorithmus im Rahmen der Shakey-Entwicklung von *Hart u. a.* (1968) publiziert.

1977 wurde in Tsubaka in Japan am *Mechanical Engineering Laboratory* das erste autonom fahrende Automobil gebaut. Es besaß ein Stereokamerasystem und konnte bei guten Bedingungen ca. 50 Meter bei 30 km/h autonom fahren. Um dies zu erreichen, wurde spezielle Hardware entwickelt, die Helligkeitsänderungen in den Bildern der Kameras direkt in digitale Pulse umwandelte. Hindernisse und Straßenmarkierungen wurden als bestimmte Abfolge dieser Pulse erkannt. (*Tsugawa* u. a. 1979; *Moravec* 1999)

Nach diesen ersten Versuchen wurde in den frühen 80ern in den USA und Deutschland unabhängig voneinander an Automobilen geforscht, die alle Komponenten inklusive der benötigten Computer an Board hatten, um autonome Fahrten vorzunehmen. In Deutschland fokussierte man sich am Lehrstuhl von *Ernst Dieter Dickmanns* an der Universität der Bundeswehr München auf Autobahnfahrten (*Meissner* 1982). In den USA wurde im Rahmen der DARPA Initiative *On Strategic Computing* im Unterprogramm *Autonomous Land Vehicles* (*Lowrie* u. a. 1985, im Folgenden ALV) mehrere Forschungsvorhaben finanziert. Unter anderem ein größeres Versuchsfahrzeug der Martin-Marietta Corporation (jetzt Lockheed Martin Corporation) und ein Fahrzeug (*NavLab 1*) an der Carnegie Mellon University (im Folgenden CMU, *Dowling* u. a. 1987; *Goto* und Stentz 1987). Beide Fahrzeuge legten den Fokus auf langsames Fahren (maximal 20 km/h) auf unbefestigten Straßen. Auf beiden Seiten des Atlantiks waren diese frühen Prototypen Kastenwägen oder kleine Busse. Die Größe war nötig, um die erforderliche Rechenleistung und die teilweise sehr aufwendigen Umbauten transportieren zu können. *NavLab 1* z.B. wog ca. 5 t. *NavLabs* Entwurfsgrundlage für die Modellierung und Planung basierte schon auf einem hierarchischen System, wie es in Abbildung 2.13 abgebildet ist. Die Architektur sah mehrere Abstraktionsstufen und auf den Abstraktionsgrad angepasste Planungs- und Interpretationsmöglichkeiten vor.

1987 wurde der an der Universität der Bundeswehr in München (im Folgenden UBM) entwickelte *VaMoRs*[1] 20 Kilometer auf einem freien Abschnitt der Autobahn bei Geschwindigkeiten bis zu 100 km/h erfolgreich getestet. Das darauf folgende EUREKA[2]-Projekt *Prometheus*[3] (1987–1994) hat in Europa mehreren Forschungsgruppen die Entwicklung von Versuchsträgern finanziert. Unter anderem geht daraus der an der Universität der Bundeswehr in München entwickelte *VaMP*[4] hervor. *VaMP* – mittlerweile war die Technik in einer Mercedes S-Klasse unterzubringen – hat 1994 mehrere autonome Fahrten im Pariser Verkehr absolviert. Insgesamt mehr als 1000 km mit Höchstgeschwindigkeiten von bis zu 130 km/h und dem automatischen Überholen von langsameren Verkehrsteilnehmern.

1995 fanden bei beiden Forschergemeinden größere Systemtests statt: Die CMU ließ ihren *NavLab 5*, eine Weiterentwicklung im Rahmen ihrer NavLab Reihe, in dem Projekt *no hands across America* 4.585 Kilometer von der Ostküste zur Westküste Amerikas fahren. 98 % der Zeit

[1] Versuchsfahrzeug für autonome Mobilität und Rechnersehen.
[2] Europäische Initiative zur Förderung anwendungsnaher Forschung: http://www.eureka.be/, http://www.eureka.dlr.de/.
[3] PROgramme for an European traffic of Highest Efficiency and Unprecedented Safety, (*Braess* und Reichart 1997a, b).
[4] Nachfolger von VaMoRs. Versuchsfahrzeug für autonome Mobilität und Rechnersehen im Pkw.

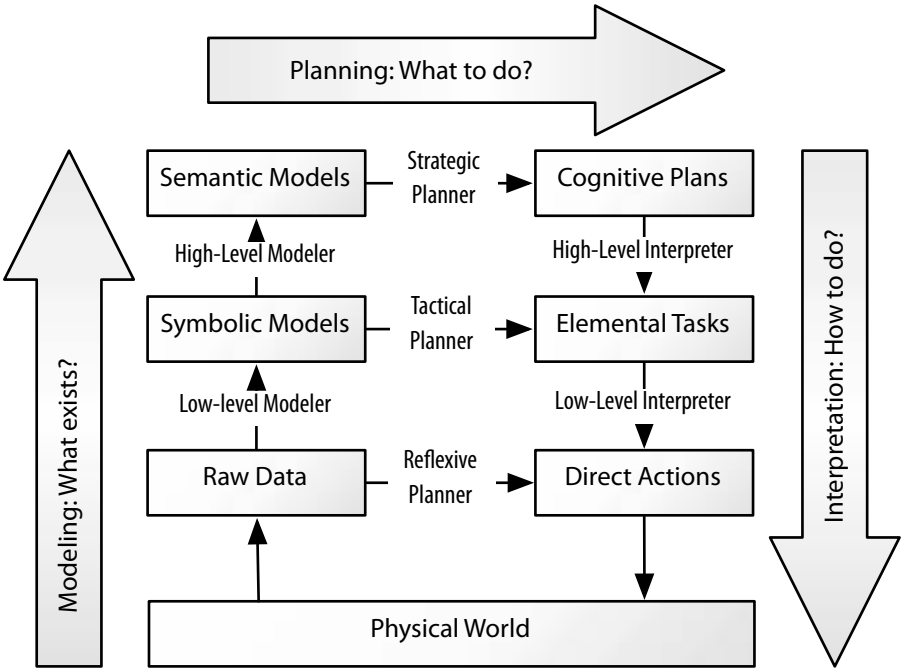


Abbildung 2.13.: Die hierarchische Systemarchitektur der Modellierungs- und Planungskomponente von *NavLab 1* (aus *Dowling* u. a. 1987).

davon übernahm RALPH[1](*Pomerleau* 1995), ein an der CMU entwickeltes und trainiertes neuronales Netz, die Querregelung. Gas und Bremse wurden von menschlichen Fahrern gesteuert. Der Systemtest der Forscher um *E.D. Dickmanns* am Institut für Systemdynamik und Flugmechanik der UBM bestand aus einer 1687 km langen Autobahnfahrt von München nach Dänemark und zurück. 95 % der Zeit fuhr *VaMP* vollkommen autonom mit Höchstgeschwindigkeiten von bis zu 180 km/h. *Dickmanns* 4D-Ansatz und Blickrichtungssteuerung der Kameras waren die Forschungsschwerpunkte. *Pellkofer* (2003) und *Siedersberger* (2003) entwickelten für diesen Ansatz ein komplexes Verhaltensnetzwerk mit baumartig aufgebauter Objektdatenbasis als explizitem Wissensspeicher, das in dieser Arbeit im Abschnitt 2.1.4 genauer vorgestellt wird.

Auch andere Forschergruppen haben ähnliche Systemtests durchgeführt, z.B. das italienische ARGO-Projekt – ein Nachfolgeprojekt des pan-europäischen Prometheus Projekts. 1998 fuhr das ARGO-Fahrzeug 2000 km im Rahmen des *MilleMiglia in Automatico* Tests auf italienischen Autobahnen. Ca. 95 % der Zeit davon fuhr das GOLD[2]-System autonom. GOLD ist ein modulares System, das den Schwerpunkt auf die Computervision legt. (*Broggi* u. a. 2001)

[1]Engl. Rapidly Adapting Lateral Position Handler
[2]Engl. Generic Obstacles and Lane Detection

2004 wurde die erste DARPA Grand Challenge, ein Roboter-Rennen in der Mojavewüste im Süden Kaliforniens durch unstrukturiertes Terrain, ausgetragen. Aber erst in der zweiten DARPA Grand Challenge 2005 gelang es vier Teilnehmern, die 212 km lange Strecke in unter 10 Stunden zu bewältigen. Gewinner des Rennens und damit von 2 Millionen Dollar war die Universität Stanford mit ihrem Roboter *Stanley*, der den Fokus auf probabilistische Verfahren legte und die Unsicherheiten der Wahrnehmung mit in die Planungskomponente aufnimmt. (*Thrun* u. a. 2007b)

Dieser historische Überblick soll im Rahmen dieser Arbeit ausreichen. Ausführlichere und auf die Computervision fokussierte historische Überblicke findet der geneigte Leser z.B. in (*Moravec* 1999; *Dickmanns* 2002; *Turk* u. a. 1988; *Schmidhuber*).

2.3.2. DARPA Urban Challenge 2007

Im Folgenden wird die *DARPA Urban Challenge 2007* (DARPA 2007, im Folgenden UC07) näher beschrieben. Die UC07 ist ein Rennen für autonom fahrende Fahrzeuge, das am 3.11.2007 in Victorville, Kalifornien stattfand. Es ist das mit Abstand größte Ereignis dieser Art mit einer Beteiligung fast aller führenden Forscherteams auf diesem Gebiet.

Als solches ist es gut geeignet, um einen Überblick über den aktuellsten Stand der Forschung zu geben. Außerdem war das Team AnnieWAY – wie eingangs erwähnt – eines der teilnehmenden Teams der UC07 und auch deshalb soll die UC07 und einige der anderen Teams hier vorgestellt werden.

In der UC07 müssen die teilnehmenden Roboter autonom verschiedene Missionen in einem urbanen Umfeld durchführen und dabei die kalifornischen Verkehrsregeln einhalten. Die Missionen bestehen aus einer Listen von zu überfahrenden Kontrollpunkten oder durchzuführenden Parkmanövern.

89 Teams haben sich für das Rennen angemeldet, 53 wurden in die Vorentscheidungen aufgenommen und von ihnen sind 35 zu dem Halbfinale (*National Qualification Event* vom 26. bis 31.10.2007) zugelassen worden. Im Halbfinale mussten sie auf verschiedenen Teststrecken ihre Fahrtüchtigkeit in einem nachgestellten urbanen Umfeld unter Beweis stellen. Jede Teststrecke hatte einen anderen Schwerpunkt und bei zwei von ihnen haben professionelle Stuntfahrer Straßenverkehr simuliert. Auf der Teststrecke A wurde das Einfädeln in den fließenden Verkehr verlangt. Teststrecke B legte den Schwerpunkt auf das Navigieren in einem umfangreichen Straßennetz mit statischen Hindernissen und Teststrecke C hat das Kreuzungsverhalten der Roboter getestet.

Das Halbfinale haben 11 teilnehmende Teams erfolgreich absolviert und konnten somit am Finale teilnehmen:

- AnnieWay (Universität Karlsruhe, Forschungszentrum Karlsruhe, TU München, Bundeswehr Universität München)
- Ben Franklin (University of Pennsylvania, Lehigh University)
- CarOLO (TU Braunschweig)
- Cornell (Cornell University)
- Intelligent Vehicle Systems (Honeywell)
- MIT (Massachusetts Institute of Technology)
- Stanford Racing (Stanford University)
- Tartan Racing (Carnegie Mellon University)
- Team Oshkosh (Oshkosh Truck Corporation)
- Team UCF (University of Central Florida)
- Victor Tango (Virginia Tech)

Im Finale mussten diese 11 Teams jeweils 3 Missionen mit insgesamt ca. 96 km Wegstrecke in unter 6 Stunden bewältigen. 6 Teams haben dies geschafft und nach einer Analyse ihrer Zeiten und ihres Fahrverhaltens haben die ersten drei Plätze Tartan Racing, Stanford Racing und Victor Tango belegt.

Der Schwerpunkt lag bei diesem Rennen nicht unbedingt auf Schnelligkeit sondern, das erste Mal, auf dem Einhalten der Verkehrsregeln. Herausfordernd waren auch die strikten Regeln, die z.B. ein menschliches Eingreifen, wie es in den im vorherigen Abschnitt erwähnten Systemtests bei schwierigen Situationen die Regel war, ausschlossen.

Im Folgenden werden die drei Gewinner-Teams der *UC07* vorgestellt und ihre Systemarchitektur, soweit sie bekannt ist, zusammengefasst. Hier soll noch darauf hingewiesen werden, dass mit hoher Gewissheit in naher Zukunft genauere Beschreibungen der Systemarchitekturen der Gewinner-Teams als Konferenzbeiträge oder Journalbeiträge veröffentlicht werden. Die folgenden Beschreibungen stützen sich hauptsächlich auf technische Berichte, die Teil des UC07-Auswahlverfahrens waren.

Tartan Racing

Tartan Racing ist das Team der Carnegie Mellon University um William „Red“ Whittaker. Abbildung 2.15 zeigt das autonome Fahrzeug des Teams mit Namen *Boss*, das die *DARPA Urban Challenge 2007* gewann. Tartan Racings Systemarchitektur unterteilt das Problem des autonomen Fahrens in die folgenden Unterprobleme (siehe Abbildung 2.14 und *Whittaker* u. a. 2007)

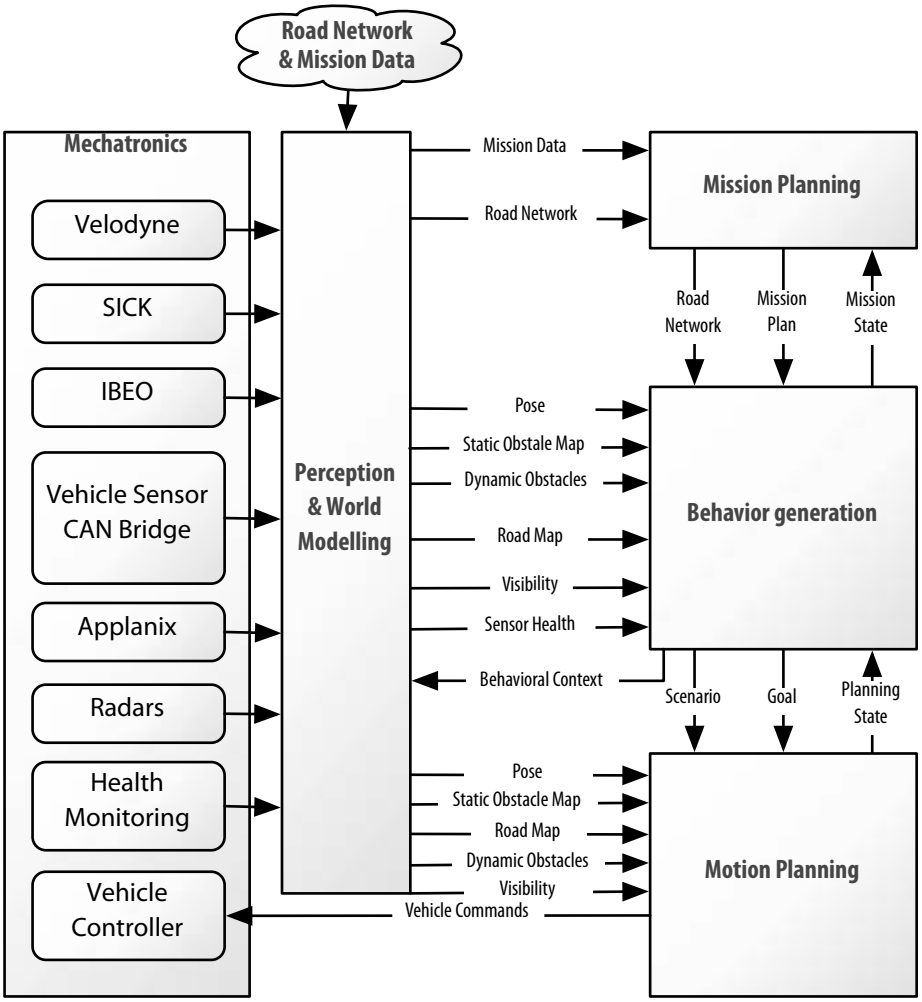


Abbildung 2.14.: Die Architektur von *Boss* ist in fünf Bereiche unterteilt: Mission Planning, Behavior Generation, Motion Planning, Perception and World Modelling, and Mechatronics. Abbildung nach *Whittaker* u. a. (2007).

Missionsplannung[1] Die Missionsplannung wird auf einem gerichteten und gewichteten Graphen des Straßennetzwerks durchgeführt, wobei jede Spur gesondert im Graphen repräsentiert ist. Die Gewichte der Kanten sind eine Kombination verschiedener Faktoren wie z.B. der erwarteten Zeit, die voraussichtlich benötigt wird, um die Kante zu passieren. Für die Missionsplanung wird von jedem Knoten des Graphen der an Kosten minimale Weg zu dem nächsten zu überfahrenden Checkpoint berechnet. Neue Umwelteinflüsse wie neu erkannte Blockaden oder Kreuzungen führen zu einer Änderung des Graphen und einer Neuberechnung der minimalen Wege. Aufgrund der relativ geringen Größe des Graphen kann diese Berechnung während der Fahrt im Hintergrund laufen.

[1]Engl. mission planning

Abbildung 2.15.: *Boss* – das autonome Fahrzeug von *Tartan Racing*, dem Team der Carnegie Mellon University, das die *DARPA Urban Challenge 2007* gewann.

Verhaltensgenerierung[1] Aufgrund der gewünschten Position und der aktuellen Gegebenheit werden Fahrkontexte[2] ausgewählt, um das Problem der Verhaltensgenerierung einzugrenzen. Die obersten Fahrkontexte sind *Straße folgen*, *Kreuzung behandeln* und *Zonenposition erreichen* (z.B. beim Einparken aber z.B. auch für U-turns). Innerhalb dieser Fahrkontexte gibt es Unterverhalten wie z.B. *Straße entlangfahren* oder *defensiver Spurwechsel*. Abbildung 2.16 zeigt exemplarisch den Fahrkontext *Kreuzung behandeln* und seine Unterverhalten. Es wurde leider nicht näher darauf eingegangen, wie diese Unterverhalten weiter strukturiert sind.

Bewegungsplanung[3] Ein Segment des aktuellen Missionsplans ist Grundlage der Bewegungsplanung. Mit ihrer Hilfe werden die Trajektorien für die Fahrzeugsteuerung generiert, was sowohl im *Straße-folgen* Fall wie auch beim Navigieren in Zonen mit Hilfe eines von *Howard* u. a. (2006) entwickelten Algorithmus zur dynamischen Berechnung von fahrbaren Trajektorien für lokale Ziele geschieht. Im *Straße-folgen*-Fall werden dabei die lokalen Ziele anhand eines Spurmarkierungsalgorithmus ermittelt. Im *Navigieren-in-Zonen*-Fall werden die lokalen Ziele durch einen Anytime D* Suchalgorithmus (*Likhachev* u. a. 2005) erstellt. Der Anytime D* arbeitet hier auf einem Gitter aus Fahrzeugposition und Ausrichtung und findet mit Hilfe von offline mit einem akkuraten Fahrzeugmodel erstellten Trajektorien von einem Knoten zum nächsten.

Wahrnehmung und Weltmodellierung[4] Diese Komponente interpretiert die Informationen von den verschiedenen Sensoren des autonomen Automobils und fusioniert sie zu einem einheitli-

[1] Engl. behavior generation
[2] Engl. driving contexts
[3] Engl. motion planning
[4] Engl. perception and world modeling

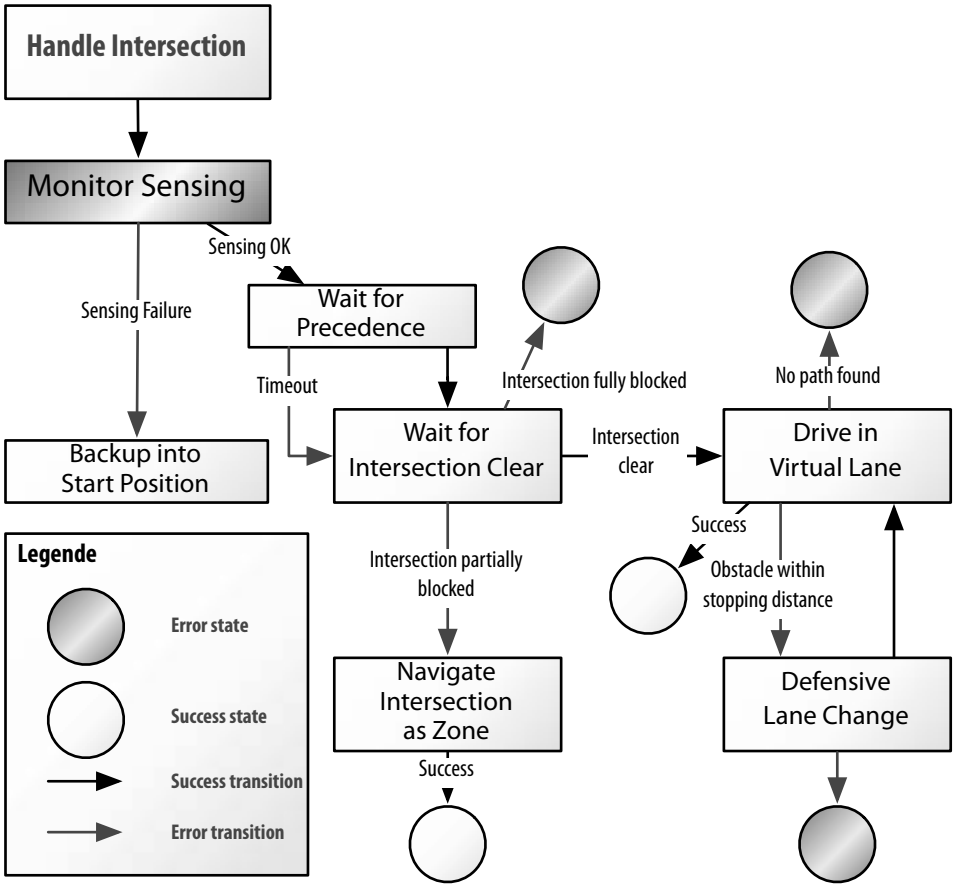


Abbildung 2.16.: Tartan Racing: Schematische Repräsentation des *Kreuzung behandeln* Fahrkontexts.

chen Model der Welt. Mit dieser Komponente können beweglichen Objekte, statische Objekte und Fahrbahnmarkierungen erkannt werden und sie stellt die aktuelle Fahrzeugposition und den Fahrzeugzustand den anderen Komponenten bereit. Bemerkenswert an dieser Komponente ist die in ihr erfolgende mehrstufige Klassifikation und Datenfusion, die die Daten von unterschiedlichen Lidar-, Video- und Radar-Sensoren zu einem einheitlichen Model der Welt fusionieren kann.

Mechatronik[1] In dieser Komponente sind alle Sensoren und Aktoren des autonomen Automobils angesiedelt. Sie umfasst auch die sonstige Elektronik, die für eine autonome Fahrt nötig ist. *Boss* besitzt 9 unterschiedliche Arten von Sensoren, die ihm eine umfassende 360 Grad Sicht ermöglichen.

[1]Engl. mechatronics

Stanford Racing Team

Abbildung 2.17.: *Junior* – das autonome Fahrzeug vom *Stanford Racing Team*, dem Team der Stanford University, das den zweiten Platz bei der *DARPA Urban Challenge 2007* gewann.

Die Missionsplanung von *Junior*, dem kognitiven Automobil des Stanford Racing Teams um Sebastian Thrun, unterscheidet sich nicht wesentlich von der Missionsplanung von *Boss*. Die Verhaltensgenerierung von *Junior* unterscheidet Makro- und Mikro-Plan. Makropläne bilden diskrete Entscheidungen wie Spurwechsel oder Abbiege-Entscheidungen ab, Mikropläne führen kontinuierliche Prozesse wie das Halten eines lateralen Spuroffsets aus. Der Pfadplaner kann alle möglichen Makro- und Mikropläne aus dem umliegenden Straßennetz generieren und weist jedem Plan-Paar verschiedene Kosten zu. Er berücksichtigt dabei auch statische und dynamische Hindernisse. Statische Hindernisse werden umfahren, dynamische Hindernisse werden durch das Herabsetzen der eigenen Geschwindigkeit vermieden. Das Plan-Paar mit den minimalen Kosten wird ausgewählt und ausgeführt. Vorfahrtsregeln werden beachtet, indem die Überschneidungen des ausgewählten Pfades mit anderen Spuren auf Gegenverkehr überprüft werden.

Pfadplanung auf Parkplätzen wird beim Standford Racing Team von einem speziellen, mehrschichtigen Pfadplaner für unstrukturiertes Gelände behandelt. Eine Kombination aus dynamischer Programmierung auf einem groben 3D-Raster und des Rapidly-Exploring Random Tree Algorithmus (*LaValle* 1998) wählt den zu fahrenden Pfad aus.

Weitere Informationen – auch zu den hier aus Platzgründen nicht erwähnten Wahrnehmungskomponenten *Juniors*, sind in *Thrun* u. a. (2007a) zu finden.

Abbildung 2.18.: *Odin* – das autonome Fahrzeug des *Victor-Tango*-Teams der VirginiaTech University. *Odin* erzielte den dritten Platz bei der *DARPA Urban Challenge 2007*.

Victor Tango

Der in *Reinholtz* u. a. (2007) vorgestellte Ansatz des *Victor Tango* Teams unterteilt die Architektur von *Odin* – dem kognitiven Automobil dieser Gruppe – in ähnliche Bereiche wie die Systemarchitektur von Tartan Racing. Abbildung 2.18 zeigt *Odin*, Abbildung 2.19 einen Überblick der verwendeten Systemarchitektur: Auch sie verfolgt das hybride Paradigma einer verhaltensbasierten Architektur. Die Module *Routenplanung*,[1] *Fahrverhalten*[2] und *Bewegungsplanung*[3] gehören dementsprechend verschiedenen Reaktivitätsstufen an: Die Routenplanung ist am meisten deliberativ bis hin zu der stark reaktiven Bewegungsplanung. Auf diese drei Komponenten soll sich die folgende Beschreibung auch konzentrieren. Mehr über die Wahrnehmungskomponenten und die Anbindung an die Auto-Technik kann in (*Reinholtz* u. a. 2007) nachgelesen werden.

Routenplanung Die Routenplanungskomponente von *Odin* beschränkt sich auf die Auswahl der optimalen Abfolge von Straßensegmenten, um eine gegebene Mission zu erfüllen. Dafür wird ein Graph mit allen Exit-Knoten[4] aufgebaut und mit der zu erwartenden Fahrzeit annotiert. Auf diesem Graphen wird mit Hilfe des A*-Algorithmus ein optimaler Pfad gefunden. Dynamische Straßenblockaden und die tatsächlich gemessene Geschwindigkeit des Verkehrs auf den Straßensegmenten finden bei der Annotation Berücksichtigung.

[1] Engl. route planner
[2] Engl. driving behaviors
[3] Engl. motion planner
[4] Exit-Knoten sind solche Knoten in der Straßennetzbeschreibung der DARPA, die ein Straßensegment mit einem anderen verbinden.

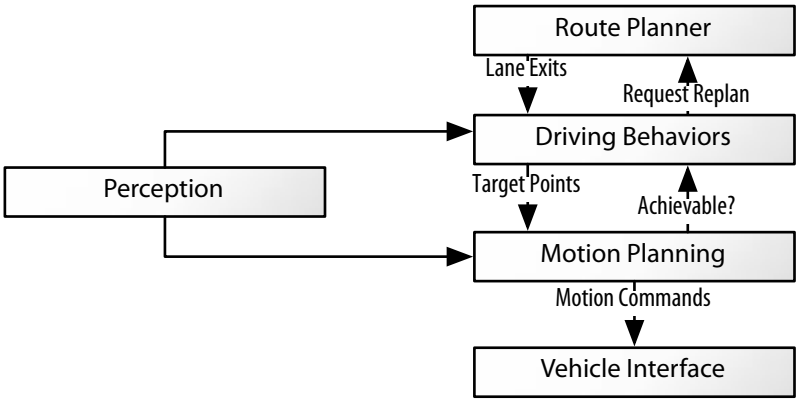


Abbildung 2.19.: Vereinfachte Systemarchitektur für *Odin*, dem kognitiven Automobil des *Victor Tango* Teams.

Fahrverhalten Diese Komponente besteht aus mehreren unabhängigen Verhalten, die über eine angepasste Winner-Takes-All-Strategie miteinander kooperieren. Die Verhalten können drei unterschiedliche Arten von Ausgaben produzieren: Zielpunkte,[1] gewünschte Geschwindigkeit und gewünschte Fahrspur. Die Winner-Takes-All-Strategie wählt jeweils in diesen drei Bereichen das aktive Verhalten aus, das momentan am wichtigsten ist. So ist z.B. die Fahrspurentscheidung, für ein anstehendes Rechtsabbiegen auf der rechten Spur zu bleiben, wichtiger als der Spurwechsel, um ein langsames vorausfahrendes Fahrzeug zu vermeiden. Neben dieser Priorisierung gibt es keine andere Ordnung, wie z.B. Hierarchisierung, unter den unabhängig voneinander arbeitenden Verhalten.

Bewegungsplanung Die Bewegungsplanung erzeugt aus den Ausgaben der Fahrverhalten die nötigen Steuerinformationen für die Fahrzeugsteuerung. Sie kann aber auch der Fahrverhalten-Komponente signalisieren, wenn gewünschte Ziele nicht zu erreichen sind, weil sie z.B. durch Hindernisse belegt sind. Kern dieser Komponente ist eine Suche nach durchführbaren Pfaden, die vom aktuellen Fahrzeugzustand ausgehend mögliche zukünftige Steuergrößen sucht, um zum gewünschten Ziel zu gelangen.

Das System des Team AnnieWAY wird in Kapitel 6 im Rahmen des Vergleichs der Planungskomponente des Team AnnieWAY mit dem in dieser Arbeit entwickelten Verhaltensnetzwerk näher erläutert. Eine Systembetrachtung weiterer Teams soll im Rahmen dieser Arbeit nicht vorgenommen werden.

[1] Engl. target points

2.4. Software-Hilfsmittel

Dieser Abschnitt stellt die verwendeten Software-Bibliotheken und Software-Frameworks vor, die bei der Umsetzung dieser Arbeit verwendet wurden. Eine weitergehende Einführung in diese Software-Hilfsmittel kann hier aber aus Platzgründen nicht erfolgen.

2.4.1. Qt

Trolltechs Qt[1] ist ein plattformunabhängiges Framework, das es Entwicklern ermöglicht, grafisch aufwendig gestaltete Programme zu entwickeln, ohne sich auf eine Plattform festlegen zu müssen. Um die Plattformunabhängigkeit zu erreichen, unterstützt das Qt Framework aber nicht nur grafische Elemente, sondern auch eine Vielzahl unterschiedlicher nicht grafischer Bereiche, die sich auf den verschiedenen Plattformen unterscheiden. Hierunter sind z.B. die Parallelisierung oder Dateizugriffe einzuordnen. Abbildung 2.20 zeigt einen Überblick über das Qt Framework und einige seiner wichtigsten Bibliotheken. Das Qt Framework enthält neben diesen Anwendungsbibliotheken auch Hilfsmittel, die das Übersetzen von Programm-Quelltext in ausführbaren Maschinencode plattformunabhängig realisieren.

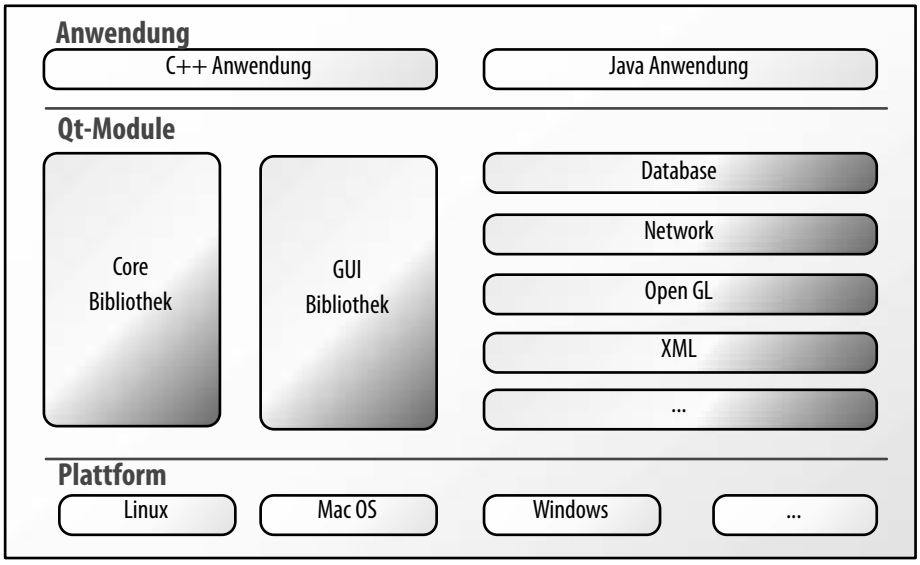


Abbildung 2.20.: Qt Framework: Eine Übersicht.

[1] http://trolltech.com/products/qt

2.4.2. Computational Geometry Algorithms Library

Die *Computational Geometry Algorithms Library* (CGAL)[1] ist eine Open Source C++-Template-Bibliothek, die mathematische Datenstrukturen und Algorithmen in effizienter Weise implementiert. Sie bietet durch C++-Templates die Möglichkeit, diese Datenstrukturen und Algorithmen mit den unterschiedlichsten Datentypen zu benutzen. Eine herausragende Eigenschaft der Bibliothek ist es, auch mit exakten Datentypen verwendbar zu sein und damit auch exakte Ergebnisse liefern zu können. Mit dem entsprechenden Datentyp ist es damit möglich, mathematische Funktionen ohne Fehler zu berechnen. Mehr über CGAL findet der geneigte Leser in *CGAL Editorial Board* (2007).

2.4.3. Boost

Boost[2] ist eine Sammlung von C++-Template-Bibliotheken, die sehr gut mit der C++-Standard-Bibliothek zusammenarbeiten. Viele der enthaltenen Bibliotheken sind für eine Eingliederung in die Standard-Bibliothek vorgesehen. Die 87 Bibliotheken können in so unterschiedliche Kategorien wie z.B. Zeichenketten- und Textverarbeitung, generische Container und Algorithmen, Tests, Template-Metaprogrammierung und viele mehr eingeordnet werden.

2.4.4. OpenCV

OpenCV[3] ist eine C/C++ Open-Source Bibliothek von *Intel* für die Bildverarbeitung. Sie hat eine effiziente Implementierung von vielen Bildverarbeitungsalgorithmen und -datenstrukturen. Auch neuere Forschungsergebnisse werden in ihr implementiert. Außerdem enthält die Bibliothek auch grundlegendere Hilfsmittel wie Bildformat-Konverter oder effiziente Bildausgabe-Module.

[1] http://www.cgal.org/
[2] http://www.boost.org/
[3] http://www.intel.com/technology/computing/opencv/index.htm

3. Systemarchitektur des SFB/TR28[1]

In diesem Kapitel soll die Systemarchitektur des Sonderforschungsbereich/Transregio 28 »Kognitive Automobile« (SFB/TR28) und die in diesem entwickelten Hilfsmittel vorgestellt werden. Der Schwerpunkt liegt dabei auf den Systemkomponenten, die für die Verhaltensentscheidung relevant sind. Abbildung 3.1 zeigt eine grobe Übersicht über die aktuell verfügbaren Komponenten und ihre Verbindungen. Im Folgenden werden diese Komponenten kurz vorgestellt. Es wird auch auf zwei Komponenten eingegangen, die noch in der Planungsphase sind und deshalb in Abbildung 3.1 nicht berücksichtigt wurden.

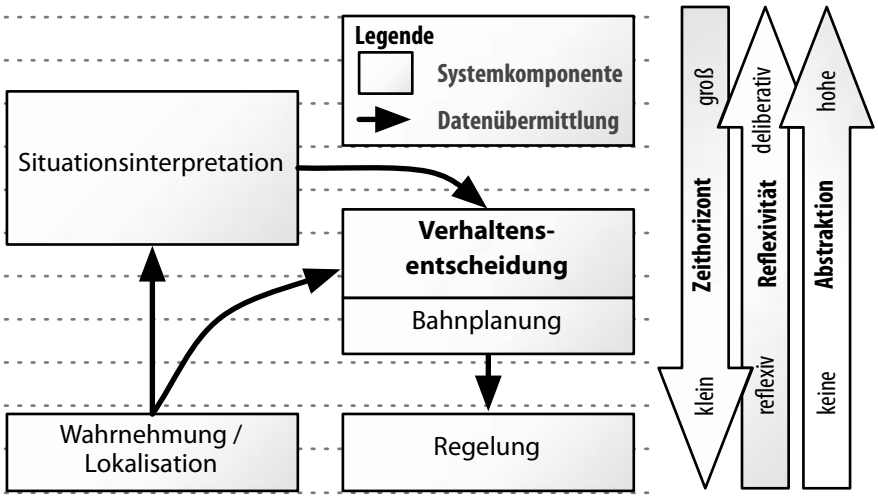


Abbildung 3.1.: Vereinfachte Systemarchitektur aktuell integrierter SFB/TR28 Teilkomponenten. Die vertikale Position der Systemkomponenten spiegelt ihre Einordnung innerhalb der Skalen am rechten Rand wider.

3.1. Konventionen und Hilfsmittel

Ein komplexes System wie das des SFB/TR28 mit vielen beitragenden Parteien braucht eine gemeinsame Basis, um eine effektive Zusammenarbeit zu ermöglichen. In dem nun folgenden

[1] Außer den im Text erwähnten Quellen wurde dieses Kapitel nach Angaben auf http://www.kognimobil.org/ erstellt.

Abschnitt wird diese Basis, die aus Konventionen und gemeinsam benutzten Komponenten besteht, soweit sie die Verhaltensgenerierung betrifft, vorgestellt.

3.1.1. Echtzeitdatenbank

Im SFB/TR28 wird eine Echtzeitdatenbank (im Folgenden RTDB[1]) zum Datenaustausch und zur Kommunikation der Komponenten untereinander benutzt. Es ist eine objektorientierte, nicht relationale und nicht persistente Datenbank, die andere Komponenten mithilfe einer C/C++-Schnittstelle verwenden können.

Die RTDB benutzt Shared-Memory[2], um die Daten an die angeschlossenen Prozesse zu übertragen. Über die Schnittstelle kann ein Prozess Objekte nach bestimmten Kriterien suchen, aktuelle Daten für ein Objekt anfordern, oder auf Aktualisierungen der Daten eines Objekts warten. Außerdem ist jedes Datum mit einem Zeitstempel ausgestattet und die RTDB hält für die Objektdaten jeweils eine Historie vor, so dass der Zustand eines Objekts zu einem bestimmten Zeitpunkt abgerufen werden kann. Die RTDB ist schnell genug, um die umfangreichen Sensordaten der Kameras und Laserscanner in Echtzeit an die sie benötigenden Prozesse zu verteilen. Es ist möglich, den Datenbankinhalt mitzuschneiden und später nochmals abzuspielen. Mehr über die RTDB kann der interessierte Leser in *Goebl* und Färber (2007) erfahren.

Die RTDB wurde nicht für verteilte Anwendung konzipiert, sie enthält also nicht die dafür benötigten verteilten Kommunikations- und Replikationsmechanismen, mit der Prozesse verschiedener Rechner auf die gleiche Datenbank zugreifen können. Um trotzdem Daten über Rechnergrenzen hinweg benutzen zu können, existiert eine Netzwerkbrücke, mit der einzelne Objekte über das Netzwerk übertragen werden können. Die Brücke publiziert ein Objekt von einer Datenbank in eine ansonsten unabhängige Datenbank auf einem anderen Rechner.

Gemeinsame Daten in mehreren Komponenten benutzen zu können, ist nicht genug. Die Daten müssen auch gleich interpretiert werden, um konsistente Ergebnisse zu erhalten. Dafür sind Konventionen nötig, die komponentenübergreifend Gültigkeit haben. Einige der im SFB/TR28 getroffenen Konventionen werden im Folgenden vorgestellt, weil sie für die Entwicklung der Verhaltensentscheidung von Bedeutung sind.

3.1.2. Referenzpunkt und fahrzeugfeste Koordinatensysteme

Viele Problemvereinfachungen gehen von einem punktförmigen Fahrzeug aus. Die Lokalisation errechnet z.B. nur die Koordinaten eines Fahrzeugpunktes und die Ausrichtung des Fahr-

[1] vom englischen Real-Time-DataBase
[2] Über Adressraum-/Prozessgrenzen hinweg gemeinsam benutzbarer Arbeitsspeicher.

zeugs in diesem Punkt. Um z.B. genaue Kollisionsdetektionen durchführen zu können, muss die dafür zuständige Komponente wissen, um welchen Punkt relativ zum Fahrzeug es sich handelt.

Im SFB/TR28 wurde deshalb ein Referenzpunkt definiert, der für jegliche punktförmige Repräsentation herangezogen werden sollte. Dieser Referenzpunkt befindet sich, wie in Abbildung 3.2 gezeigt, auf der Rückseite des Befestigungspunktes des Innenspiegels.

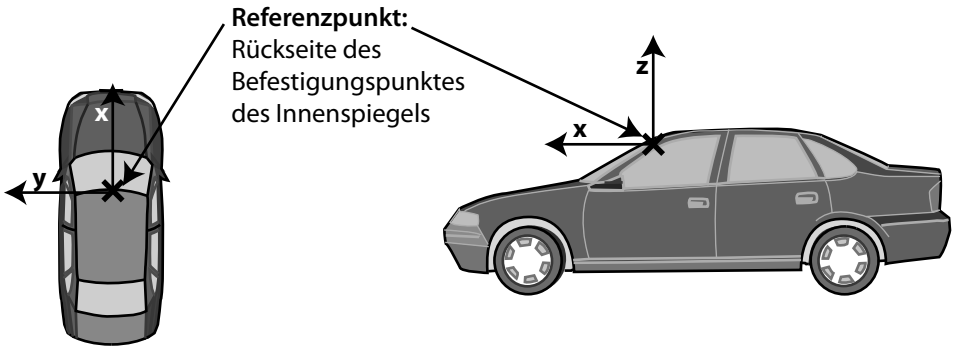


Abbildung 3.2.: Der Referenzpunkt im SFB/TR28 und das fahrzeugfeste Koordinatensystem, das an ihm aufgespannt wird.

Der Referenzpunkt spannt das fahrzeugfeste Referenzkoordinatensystem auf, ein immer an das Fahrzeug gebundenes Koordinatensystem. Dessen Abszisse zeigt von dem Innenspiegel nach vorne, die Ordinate nach links und die Applikate nach oben. Es handelt sich also um ein normales, rechtshändiges Koordinatensystem.

Jeder Sensor besitzt sein eigenes Koordinatensystem: Kameras z.B. ein rechtshändiges Koordinatensystem mit der Kamera als Ursprung. Der Hauptsensor, ein 3D-Lidar (siehe Abschnitt 3.2), benutzt ein polares Koordinatensystem mit dem Sensor als Ursprung. Diese Koordinatensysteme sollten aber nur innerhalb der Wahrnehmungskomponenten benutzt werden, ausserhalb nur das Referenzkoordinatensystem oder die im Folgenden beschriebenen nichtfahrzeugfesten Koordinatensysteme.

Eine Ausnahme zu dieser Konvention ist die Schnittstelle zwischen Bahnplanung und Regelung. Hier wird aus regeltechnischen Gründen als punktförmige Repräsentation des Autos der Mittelpunkt der Hinterachse des Fahrzeugs verwendet.

3.1.3. Globale und lokale Koordinatensysteme

Im SFB/TR28 werden verschiedene nicht-fahrzeugfeste Koordinatensysteme verwendet, da jedes Vorteile in bestimmten Bereichen hat. Auf der Erde global eindeutige Koordinaten werden durch das übliche, sphärische Koordinatensystem WGS84 mit Angabe des Breitengrades und des Längengrades als Winkel vom Mittelpunkt der Erde angegeben. Der Nullmeridian, der

Längengrad, der als nullter Längengrad definiert wurde, führt durch Greenwich, England. Es wird bei der Benutzung dieses Koordinatensystems angenommen, dass die Höhe, die sich das Auto über dem Boden befindet, vernachlässigt werden kann. Es wird also nur ein Punkt auf der Erdoberfläche mit der Angabe des Breiten- und Längengrads eindeutig bestimmt. Abbildung 3.3 zeigt grafisch, wie die Längengrade und Breitengrade auf einer Kugel verlaufen. Das Koordinatensystem WGS84 approximiert die Erde als Ellipsoid, d.h. eine Kugel die zu den Polen hin abgeflacht ist.

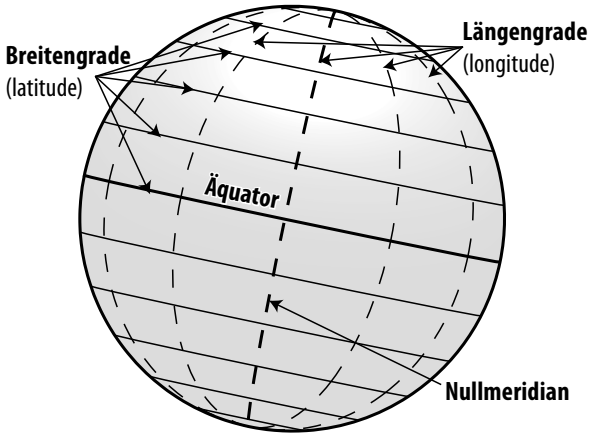


Abbildung 3.3.: Sphärisches, globales Koordinatensystem.

Da Distanzberechnungen in diesem sphärischen Koordinatensystem sehr aufwendig sind – für sie muss der Großkreis durch die zwei Punkte berechnet werden und die Bogenlänge auf diesem Großkreis bestimmt werden – wird im SFB/TR28 auch noch ein zweites, nicht-fahrzeugfestes aber kartesisches Koordinatensystem verwendet: das sog. Lokalkoordinatensystem. Dieses lokal um einen Ursprung feste Koordinatensystem ist eine Abwandlung des UTM[1]-Systems; deshalb soll hier zuerst das UTM-System kurz erläutert werden. Im UTM-System wird das Erdellipsoid in 60 vertikale Streifen von jeweils 6 ° aufgeteilt. Jeder dieser Streifen wird mittels eines Zylinders approximiert und auf diesem Zylinder wird ein kartesisches Koordinatensystem definiert.[2] Das UTM-System besteht also nicht nur aus einem Koordinatensystem, sondern ist eine Sammlung von kartesischen Koordinatensystemen, die in ihrer Gesamtheit jeden Punkt auf der Erde abdeckt. Eine Koordinate ist im UTM-System also nur eindeutig, wenn auch das zugehörige Koordinatensystem, die so genannte UTM Zone, mit angegeben wird. Das im SFB/TR28 benutzte Lokalkoordinatensystem verwendet einen Fixpunkt, der im WGS84-Koordinatensystem angegeben wird. Dieser Fixpunkt wird einmalig in UTM-Koordinaten umgerechnet und dient dann als Verschiebungsoffset für die Umrechnung aller weiteren Punkte in das kartesische Koordinatensystem. Ein Beispiel soll diese Umrechnung verdeutlichen: Der

[1] engl. Universal Transverse Mercator

[2] Dies ist eine vereinfachte Darstellung, UTM unterteilt den Ellipsoiden noch in weitere Zonen und behandelt die Polkappen verschieden. Hier mag aus Platzgründen aber diese vereinfachte Darstellung genügen. Unter http://www.koders.com/cpp/fid56D52408FAC344874E65BF9A1C54F3731C96A39B.aspx ist eine vollständige Umrechnung vom sphärischen Koordinatensystem ins UTM-System als C++-Quelltext zu finden.

Punkt $49°1'22,5084''N\ 8°25'54,7248''E$ [1] befindet sich auf dem Testgelände der Universität Karlsruhe und soll hier als Fixpunkt des Lokalkoordinatensystems dienen. Umgerechnet in das UTM-System ergibt sich die Koordinate 5430159,059N (y) 458463,971O (x) in Zone 32U. Die WGS84-Koordinate $49°1'19,7112''N\ 8°25'47,46''E$ ist in UTM-Koordinaten 5430073,786N 458316,374O Zone 32U. In Lokalkoordinaten ist das 85,273 (y) 147,597 (x). Dieses Beispiel zeigt den Vorteil des Lokalkoordinatensystems: Um den Fixpunkt produziert es nur kleine Zahlenwerte; die numerische Genauigkeit bei Berechnungen in dem Lokalkoordinatensystem ist deshalb höher. Ein Nachteil des Lokalkoordinatensystems ist aber, dass es nur sinnvoll angewendet werden kann, wenn sich alle Koordinaten innerhalb der gleichen UTM-Zone (100 km ×100 km) wie der Fixpunkt befinden, da UTM-Koordinaten unterschiedlicher Zonen nicht kompatibel sind. Außerdem erbt es die verzerrende Darstellung in Ost-West Richtung, die schon in das UTM-System aufgrund der Approximation der Kugelzonen mittels Zylinder eingeführt wurde.

Das globale Koordinatensystem wird immer dann verwendet, wenn ein Punkt auf der Erde langfristig eindeutig bestimmt werden muss. Digitales Kartenmaterial ist deshalb z.B. in diesem Koordinatensystem abgelegt. Da das globale Koordinatensystem aber ein sphärisches ist, sind häufig benötigte Operationen, wie z.B. die Bestimmung der (euklidischen) Distanz zwischen zwei Punkten, mathematisch aufwendige Operationen. Deshalb werden die Punkte innerhalb eines begrenzten Bereichs einmalig in das kartesische Lokalkoordinatensystem umgerechnet, in dem solche Operationen durch einfache mathematische Sätze, wie z.B. den Satz des Pythagoras, hergeleitet werden können.

3.1.4. Simulation und Visualisierung

Da Testen auf dem kognitiven Automobil zeitaufwendig ist und der Versuchsträger auch nicht immer verfügbar ist, wurden im SFB/TR28 mehrere Simulations- und Visualisierungskomponenten erstellt, mit denen die anderen Komponenten getestet werden können, ohne sie im Fahrzeug ausführen zu müssen. Daneben kann mithilfe der Visualisierungskomponente, die den Datenbankinhalt grafisch anzeigen kann, auch im Fahrzeug der Systemzustand visualisiert werden, was die Fehlersuche vereinfacht. Beide Komponenten arbeiten auch mit Datenbankmitschnitten, so ist eine Fehlersuche im Nachhinein möglich.

Das Fahrzeug wird in der Simulation durch ein lineares Einspurmodell (siehe *Mitschke* und Wallentowitz 2004, Kap. 18) modelliert. Es kann also ein geschlossener Regelkreislauf simuliert werden, der aufgrund des verwendeten Modells in Standardsituationen und bei moderaten Geschwindigkeiten nahe an das reale Fahrverhalten herankommt. Sensordaten werden im aktuellen Simulationsansatz nicht simuliert. Um trotzdem den Regelkreislauf schließen zu können, werden Aufnahmen von realen Sensordaten in die RTDB gespielt.

[1] Visualisierung des Testpunkts auf Google Maps: http://tinyurl.com/5b3dcf.

Abbildung 3.4.: Bildschirmabbild einer der Visualisierungskomponenten, die eine Datenbankaufnahme des Finales der DARPA Urban Challenge darstellt.

3.2. Wahrnehmung

Eine Vielzahl von Sensoren ist notwendig, um den aktuellen Zustand des kognitiven Automobils und seiner Umgebung wahrzunehmen. Die Wahrnehmungskomponente ist dafür zuständig, die anfallenden Sensordaten auszuwerten. Dazu besitzt sie eine Vielzahl von spezialisierten Unterkomponenten, z.B. zur Fahrbahndedektion oder zur Hindernisdetektion. Die erkannten Daten werden von den einzelnen Unterkomponenten an die Interpretationskomponente weitergereicht, um dort eine Gesamtsicht der aktuellen Situation zu bilden.

Die Lokalisation erfolgt mithilfe von GPS[1] und Odometrie-Daten sowie einer INS.[2] Diese Industriekomponente fusioniert GPS-Position, Odometriedaten und gemessenen Beschleunigungen zu einer absoluten Positionsbestimmung, die auch bei Fehlmessungen wie z.B. fehlerhafter oder ungenauer GPS-Positionen eine kontinuierliche Positionsberechnung ermöglicht.

Tabelle 3.1 zeigt eine Übersicht über die vorhandenen externen Sensoren und ihr Einsatzgebiet. Im SFB/TR28 wird auch an verteilter Wahrnehmung geforscht. Durch Auto-zu-Auto Kommunikation wird dadurch virtuell die Wahrnehmungsreichweite des kognitiven Automobils erweitert: Es kann damit auch Objekte wahrnehmen, zu denen von seinem Standpunkt aus keine Sichtverbindung besteht.

[1] engl. Global Positioning System

[2] engl. Inertial Navigation System. Eine kommerzielle Komponente, die eine robuste Positionsbestimmung ermöglicht.

Sensor	Genauigkeit	Einsatzgebiet
OXTS RT3003 INS	20 cm Positionsgenauigkeit 0,1 ° Ausrichtungsgenauigkeit	Lokalisation
Velodyne HDL-64E	0,09 ° horizontale Genauigkeit 5 cm Distanzgenauigkeit bis 100 m	Hinderniserkennung Spurerkenung
Sick LMS 291	1 cm Distanzgenauigkeit bis 80 m	Hinderniserkennung im Parkplatzumfeld
Stereovision	640×480 Pixel SW-Bild, 25 Bilder/s	Spurerkennung

Tabelle 3.1.: Externe Sensoren des kognitiven Automobils und ihr Einsatzgebiet.

3.3. Situationsinterpretation

Die Situationsinterpretationskomponente analysiert die aktuelle Szene als Ganzes und zieht daraus Schlüsse. Sie fusioniert die verschiedenen Wahrnehmungsdaten zu einem konsistenten Modell der aktuellen Verkehrssituation und identifiziert in diesem Modell mithilfe von fallbasiertem Schließen Beziehungen zwischen den Objekten. Diese Beziehungen – z.B. Auto X fährt auf Spur Y oder Auto X hat Vorfahrt vor Auto Y – werden den anderen Komponenten, speziell der Verhaltensentscheidung, durch die Datenbank zur Verfügung gestellt.

Eine andere Aufgabe der Situationsinterpretation ist es, die Missionsplanung durchzuführen und den Fortschritt innerhalb der Mission zu beobachten. Die Mission wird als Abfolge von Wegen auf einer digitalen Karte geplant. Den anderen Modulen wird diese Mission in einem ähnlichen Abstraktionsniveau zur Verfügung gestellt, wie es von einem handelsüblichen GPS-Navigationsgerät bekannt ist, z.B. „An der nächsten Kreuzung rechts abbiegen“ oder „Dem Straßenverlauf folgen“.

Eine tiefergehende Behandlung der Interpretationskomponente kann der geneigte Leser in *Gindele* (2007) finden.

3.4. Verhaltensentscheidung und Bahnplanung

Die Verhaltensentscheidung- und Bahnplanungskomponente ist die Systemkomponente, die aus den rohen und von der Interpretation aufbereiteten Sensordaten und der aktuellen Zielvorgabe ein angemessenes Verhalten des kognitiven Automobils erzeugt und dieses Verhalten der Regelung über eine allgemeine, nicht Auto-spezifische Schnittstelle kommuniziert.

Diese Komponente muss genauso reflexartig auf sicherheitskritische Ereignisse reagieren, sowie deliberative Entscheidungen treffen können. Deshalb ist sie mit einer hybriden, hierarchischen und verhaltensbasierten Architektur modelliert – einem biologisch motivierten Verhaltensnetzwerk (siehe Abschnitt 2.1.6 und *Albiez* 2006; *Hoffmann* 2007).

3.5. Regelung

Die Regelungskomponente setzt die von der Bahnplanung stammenden Vorgaben auf das Fahrzeug und seine Elektronik in angepasste Steuersignale um. Die momentane Schnittstelle zwischen Bahnplanung und Regelung besteht aus einer Reihe von Hinterachs-Punkten in Lokalkoordinaten, der gewünschten Geschwindigkeit für das Abfahren dieser Punkte, und die Information, welche Geschwindigkeit das Auto am Ende dieser Punktfolge erreicht haben soll. Aufgrund dieser Eingaben regelt die Komponente die Beschleunigung und die Lenkradstellung des Fahrzeugs mithilfe eines orbital tracking controllers, so dass eine minimale Abweichung zur Vorgabe erreicht wird. Näheres über die aktuell implementierte Regelung kann in *Werling* und Groll (2008); *Kammel* u. a. (2008, Abschnitt 11) nachgelesen werden.

3.6. Zukünftige Systemkomponenten

In diesem Abschnitt werden momentan noch nicht integrierte Systemkomponenten, die Einfluss auf die Verhaltensentscheidung und Bahnplanung haben, kurz erwähnt. Abbildung 3.5 stellt die Systemarchitektur mit diesen Erweiterungen dar.

3.6.1. Verteilte Verhaltensentscheidung

Im SFB/TR28 wird auch an der Kooperation von mehreren kognitiven Automobilen geforscht. Durch Kommunikation zwischen den Fahrzeugen (car-to-car communication) soll ein einzelnes kognitives Automobil in die Lage versetzt werden, Situationen besser einschätzen zu können, als es das mit seinem lokalen Wissen könnte. Ein Beispiel hierfür ist das frühzeitige Signalisieren eines Staus an heranfahrende Fahrzeuge. Es sollen aber auch kooperative Verhaltensentscheidungen getroffen werden können, die manchmal nötig sind, um Gefahrensituationen zu entkommen. Z.B. in dem in Abbildung 3.6 dargestellten Szenario, in dem ein plötzlich entgegenkommendes Auto das überholende Auto zwingt sein Überholmanöver abzubrechen. Das verteilte Verhalten veranlasst die teilnehmenden Fahrzeuge an dem Überholmanöver jeweils passende Ausweichmanöver durchzuführen, so dass das überholende Auto wieder sicher in den fließenden Verkehr einscheren kann.

3.6.2. Safety Assessment

Diese Komponente kann als virtueller Copilot angesehen werden, der kontinuierlich die Wahrscheinlichkeit eines Unfalls berechnet, indem er die von der Verhaltensentscheidung geplante

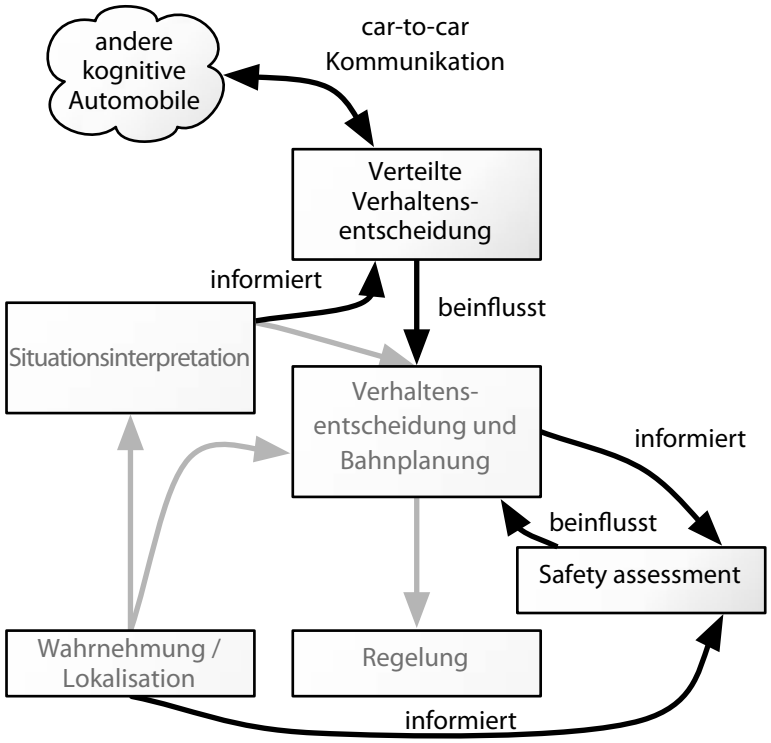


Abbildung 3.5.: Verteilte Verhaltensentscheidung und Safety Assessment sind zwei momentan noch nicht integrierte Teilbereiche der SFB/TR28 Systemarchitektur, die mit der Verhaltensentscheidungskomponente interagieren. Die aktuell vorhandenen Komponenten und ihre Beziehungen sind in der Grafik ausgegraut dargestellt.

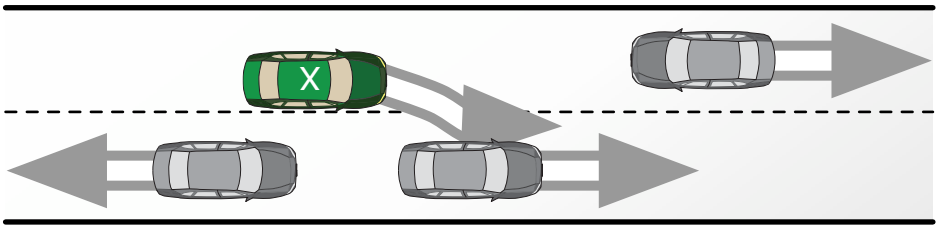


Abbildung 3.6.: Beispielszenario für eine verteilte Verhaltensentscheidung. Das mit X gekennzeichnete Auto bricht ein Überholmanöver wegen plötzlich aufkommenden Gegenverkehrs ab. Alle Autos stimmen ihre Verhalten ab, so dass die Situation aufgelöst werden kann.

Trajektorie und die wahrscheinlichen Trajektorien aller anderen Verkehrsteilnehmer in Betracht zieht. Wird eine unsichere Trajektorie erkannt, kann dies der Verhaltensentscheidung mitgeteilt werden, damit diese eine sicherere Trajektorie berechnet.

3.7. Hardware

Die Softwarekomponenten des SFB laufen in dem aktuellen Versuchsträger auf einem zweifachen AMD Opteron Dual-Core Computer in dem jeder der vier Kerne mit 2.2 GHz getaktet ist. Das System verfügt über 4 GB RAM und eine High-End Grafikkarte. Die Regelung, die einzige Komponente, die direkt per CAN-Bus[1] mit dem Auto und den Aktuatoren kommuniziert, ist auf eine Spezialhardware ausgelagert.

Kammel u. a. (2008) erläutert die Hardwarearchitektur genauer. Abbildung 3.7 ist dort entnommen und zeigt die komplette Hardwarearchitektur zum Zeitpunkt der *DARPA Urban Challenge 2007*.

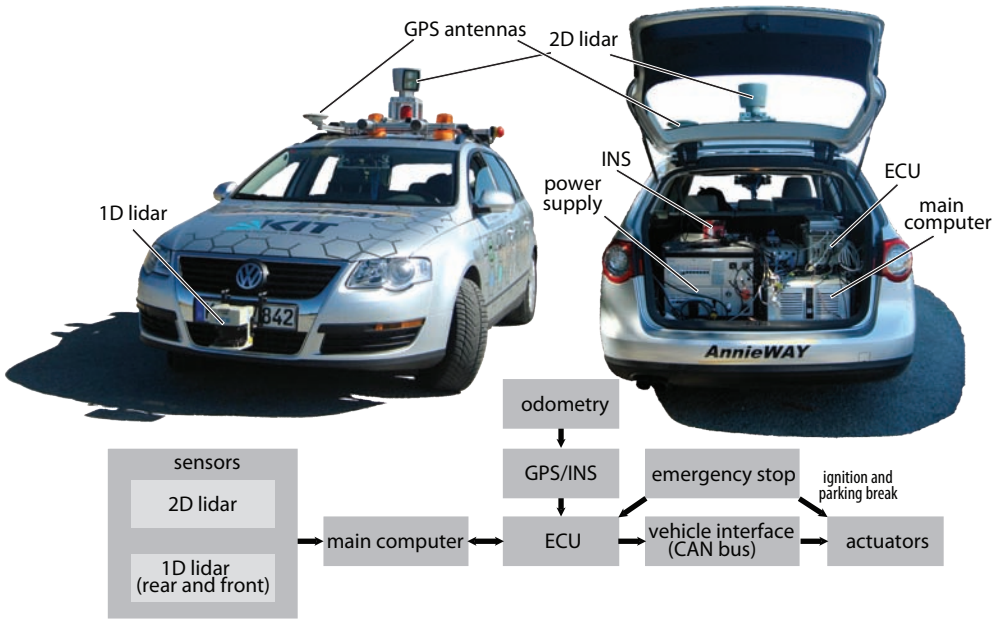


Abbildung 3.7.: Hardware-Architektur des Versuchsträgers zum Zeitpunkt der *DARPA Urban Challenge 2007* (aus *Kammel* u. a. 2008, Figure 2).

[1]Controller Area Network – asynchrones, serielles Feldbussystem.

4. Verhaltensnetzwerk

Dieses Kapitel stellt das in dieser Arbeit entworfene Framework für ein biologisch motiviertes Verhaltensnetzwerk und das damit entworfene Netzwerk vor. Die Anforderungen an den Entwurf werden im ersten Abschnitt des Kapitels identifiziert um im Weiteren das mit diesen Anforderungen entwickelte Framework vorzustellen. Abschließend wird das mit dem Framework entwickelte Verhaltensnetzwerk und dessen Verhalten beschrieben.

4.1. Anforderungen

Die von *Brooks* (1986, siehe Abschnitt 2.1.2) identifizierten, allgemeinen Anforderungen an eine Robotersteuerung gelten auch für das in dieser Arbeit beschriebene Framework für Verhaltensnetzwerke. In diesem Abschnitt werden die Anforderungen genauer erläutert und auf die in Kapitel 3 dargelegte Systemarchitektur abgestimmt. *Robustheit* wird hier – anders als bei *Brooks* – in drei einzelne Anforderungen unterteilt: Daten-Fehlertoleranz, Rechen-Fehlertoleranz und Anpassungsvermögen.

Nach den Brooks'schen Anforderungen an eine Robotersteuerung werden noch zwei erweiterte Anforderungen an das Framework vorgestellt.

4.1.1. Brooks'sche Anforderungen

Anforderung 1 (Unterstützung vielfältiger Ziele). *Das kognitive Automobil hat vielfältige Ziele mit sehr unterschiedlichen Zeitskalen und teilweise gegenläufigen Agenden. Z.B. kann das Automobil gleichzeitig die Ziele haben, die nächste Kreuzung rechts abzubiegen, dem vorausfahrenden Fahrzeug in einem gewissen Abstand zu folgen, die zulässige Höchstgeschwindigkeit nicht zu überschreiten, und einen Sicherheitsabstand zu allen Hindernissen einzuhalten. Diese Ziele sind potentiell gegenläufig, z.B. kann das autonome Automobil den Abstand zum vorausfahrenden Fahrzeug nicht einhalten, wenn dieses die zulässige Höchstgeschwindigkeit nicht einhält.*

Das Verhaltensnetzwerk muss mit solchen nebenläufigen und gegenläufigen Zielen zurechtkommen.

Anforderung 2 (Unterstützung vielfältiger Sensoren). *Diese von Brooks identifizierte Anforderung ist für das Verhaltensnetzwerk im Speziellen nicht von Bedeutung, da es durch die ihm vorgeschalteten Wahrnehmungs- und Interpretationsschichten abstraktere Sensordaten erhält. Eine nötige Sensorfusion oder die Redundanz einzelner Sensoren wird in den eben genannten Schichten durchgeführt.*

Anforderung 3 (Daten-Fehlertoleranz). *Das Verhaltensnetzwerk sollte tolerant gegenüber fehlerhaften Eingangsdaten sein. Da, wie oben beschrieben, Daten-Fusion und -Redundanz nicht Teil des Netzwerks sind, beschränkt sich also die Fehlertoleranz hier darauf, auch bei teilweise fehlerhaften Eingangsdaten noch akzeptable Ergebnisse zu liefern.*

Anforderung 4 (Rechen-Fehlertoleranz). *Das Verhaltensnetzwerk soll auch beim Ausfall einiger seiner Komponenten noch eine sichere Verhaltensentscheidung liefern. Da die Systemarchitektur des SFB auf ein einzelnes SMP[1]-System setzt, soll hier der Totalausfall eines Rechners nicht berücksichtigt werden.*

Anforderung 5 (Anpassungsvermögen). *Das Verhaltensnetzwerk muss in der Lage sein, mit einer sich schnell verändernden Umwelt zurechtzukommen. Es muss also entsprechend oft aktualisierte Umweltdaten verarbeiten können und schnell auf diese neuen Daten reagieren können.*

Anforderung 6 (Erweiterbarkeit). *Das Verhaltensnetzwerk sollte leicht erweiterbar sein. Nach Brooks heißt dies, dass bei Hinzunahme weiterer Sensoren oder Möglichkeiten, eine Aufstockung der Rechenleistung leicht möglich sein soll.*

Diese Brooks'sche Anforderung wird im Hinblick auf die heutig verfügbare Rechenleistung, das luxuriöse Platzangebot und die gute Energieversorgung in einem Automobil als gegeben angenommen. Vielmehr soll hier die Anforderung der Erweiterbarkeit wie folgt verstanden werden: Das Verhaltensnetzwerk soll leicht um weitere Verhalten erweiterbar sein. Idealerweise beeinflusst das Hinzufügen eines weiteren Verhaltens nicht die bestehenden Verhalten.

4.1.2. Erweiterte Anforderungen

Die vorherigen sechs Anforderungen wurden *Brooks* (1986) entnommen, die folgenden sind weitere Anforderungen an ein Softwaresystem, die im Rahmen dieser Arbeit zusätzlich als wichtig erachtet wurden.

Anforderung 7 (Testbarkeit). *Das Verhaltensnetzwerk sollte ein Testframework unterstützen, um seine einzelnen Komponenten soweit wie möglich unabhängig voneinander und automatisiert testen zu können.*

Anforderung 8 (Simulationsunterstützung). *Wie im vorherigen Kapitel beschrieben, ist eine Simulationsumgebung entscheidend für das effektive Testen des Gesamtsystems, da das kognitive Automobil eine begrenzte Ressource ist und das Testen auf dieser sehr aufwendig.*

[1] Symmetric multiprocessing. Mehrere Prozessoren teilen sich gleichberechtigt Systemressourcen wie z.B. den Arbeitsspeicher.

Simulation besitzt andere Anforderungen an das Verhaltensnetzwerk in Hinsicht der Echtzeitfähigkeit als der Betrieb im Fahrzeug. Bei der Simulation sollen auch Pausen möglich sein und eine Echtzeitverarbeitung aller Sensordaten ist auf den normalen Entwicklercomputern oft nicht möglich.

Das Verhaltensnetzwerk sollte im Simulationsmodus betrieben werden können und seine Ausgaben sollten sich im Simulationsmodus mit denen im Betriebsmodus des Fahrzeugs decken.

4.2. Entwurf des Frameworks

Dieser Abschnitt erläutert die Konzepte und Entwurfsentscheidungen des Frameworks für die Entwicklung von Verhaltensnetzen. Zuerst wird ein Überblick über die Architektur gegeben, um danach die einzelnen Komponenten des Systems genauer betrachten zu können.

4.2.1. Überblick

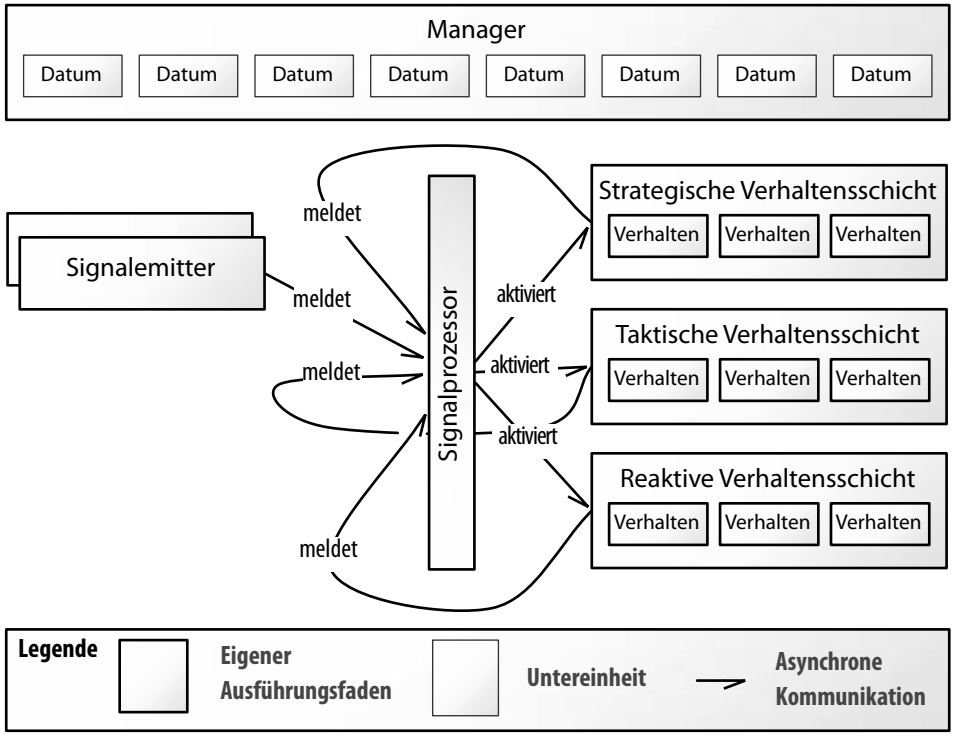


Abbildung 4.1.: Die Systemarchitektur des Verhaltensnetzwerks.

Abbildung 4.1 zeigt das grundlegende Design des hier dargelegten Verhaltensnetzwerks. Seine Hauptbestandteile sind mehrere Ausführungsfäden,[1] die im Folgenden näher erläutert werden sollen:

- Der Manager-Ausführungsfaden koordiniert und überwacht die anderen Einheiten des Systems. Außerdem ist er der Besitzer aller Ressourcen, inklusive der Repräsentationen der unterschiedlichen Eingabedaten des Netzwerks.

- Die verschiedenen Signalemitter-Ausführungsfäden warten jeweils auf ein anderes Ereignis, wie z.B. das Aktualisieren eines Datenbankobjekts oder das Ablaufen einer gewissen Zeitspanne und signalisieren dieses Ereignis dem Signalprozessor-Ausführungsfaden.

- Der Signalprozessor wiederum verarbeitet die eingehenden Signale der Signalemitter und stößt je nach Signalart und Zeit zwischen den Signalen unterschiedliche Schichten des Verhaltensnetzwerks an.

- Die Schichten des Verhaltensnetzwerks sind eigenständige Ausführungsfäden, die nebenläufig voneinander in verschiedenen Intervallen ausgeführt werden.

Um das Zusammenspiel der einzelnen Komponenten zu verdeutlichen, werden nun zwei UML-Sequenzdiagramme betrachtet, welche die Initialisierungsphase (Abbildung 4.2) und einen Aktualisierungsdurchlauf (Abbildung 4.3) verdeutlichen.

Während der Initialisierungsphase wurden die verschiedenen Ausführungsfäden noch nicht gestartet, um eine sequenzielle und wohl-definierte Initialisierung des Verhaltensnetzwerks zu ermöglichen. Der Manager ist für diese Initialisierung verantwortlich. Er initialisiert zuerst die Datenabstraktionen, danach die Verhaltensschichten, den Signalprozessor und zum Schluss die Signalemitter. Diese Reihenfolge ergibt sich durch die Abhängigkeiten der einzelnen Komponenten. Die Verhaltensschichten benötigen z.B. die Datenabstraktionen, der Signalprozessor wiederum benötigt Referenzen auf die Verhaltensschichten, um seine Signalverarbeitung zu initialisieren.

Nach erfolgreicher Initialisierung werden die Ausführungsfäden in gleicher Reihenfolge gestartet. Ab diesem Zeitpunkt werden nur noch asynchrone Nachrichten zur Koordinierung der Ausführungsfäden verwendet. Abbildung 4.3 zeigt eine beispielhafte Abfolge dieser asynchronen Nachrichten, die zur Koordination des Netzwerks verwendet werden:

- Einer der verschiedenen Signalemitter ist der Auslöser der Nachrichten-Kaskade. Wenn die Bedingung, auf die dieser Warte-Ausführungsfaden konfiguriert ist, erfüllt wurde, sendet der Signalemitter eine asynchrone Nachricht an den Signalprozessor, um danach sofort wieder auf die Bedingung zu warten.

[1] Auch im Deutschen wird häufig das englische *thread*, was für *thread of execution* steht, anstelle von *Ausführungsfaden* verwendet. In dieser Ausarbeitung soll darauf verzichtet werden.

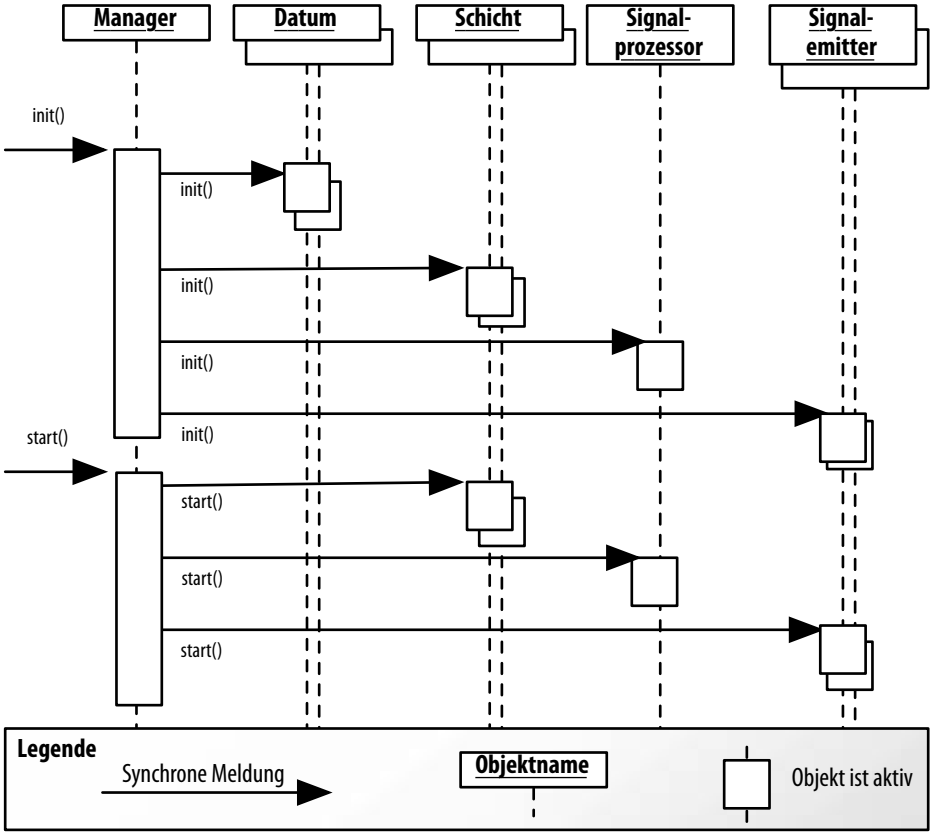


Abbildung 4.2.: UML-Sequenzdiagramm der Initialisierungsphase.

Bedingungen, auf die gewartet werden kann, sind entweder Aktualisierungen bestimmter Datenbankobjekte oder das Ablaufen eines Timers.

- Der Signalprozessor akkumuliert und filtert die Nachrichten der Signalemitter und stößt die Ausführung der verschiedenen Schichten des Verhaltensnetzwerks an. Der Signalprozessor entscheidet, welche Schichten bei welchen ankommenden Signalen ausgeführt werden müssen. So kann er z.B. entscheiden, dass die strategische Schicht ausgeführt werden muss, wenn sie schon seit 0,3 Sekunden nicht mehr ausgeführt wurde oder wenn sich die aktuelle Vorgabe der Situationsinterpretation verändert hat.

- Die Verhaltensschichten melden dem Signalprozessor, wenn ein Berechnungszyklus all ihrer Verhalten fertig ist und warten danach wieder auf ihre Aktivierung.

- Der Signalprozessor meldet dem Manager, wann ein kompletter Zyklus des Verhaltensnetzwerks durchlaufen wurde.

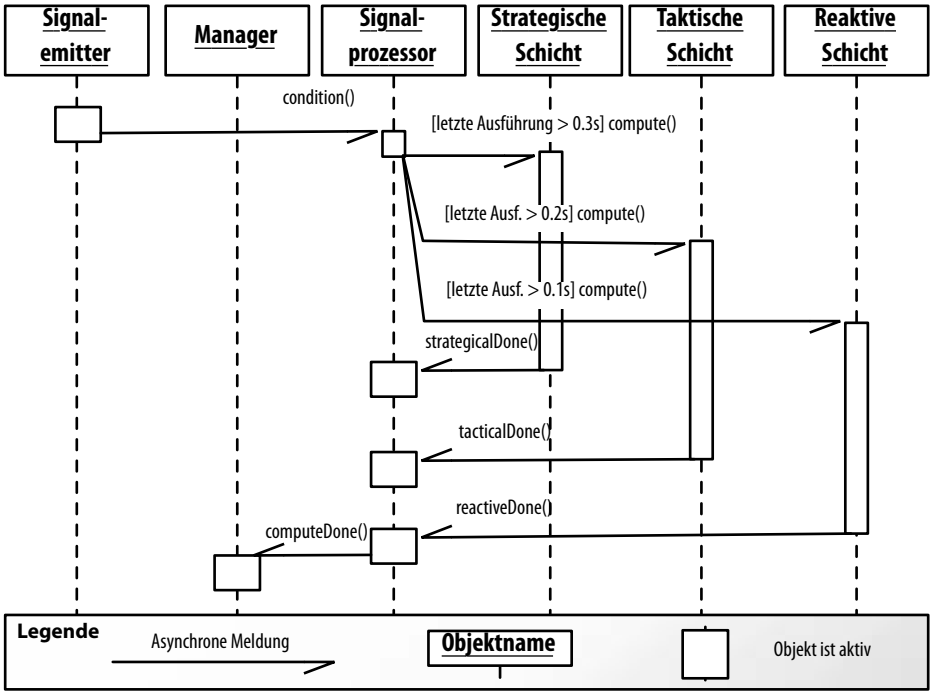


Abbildung 4.3.: UML-Sequenzdiagramm einer beispielhaften Abfolge von Nachrichten.

Nach dieser Übersicht über die Struktur des Verhaltensnetzwerks und die Interaktion seiner Ausführungsfäden soll in diesem Abschnitt auf die einzelnen Komponenten eingegangen werden und die im Vorherigen ohne Erklärung benutzten Konzepte der Datenabstraktion und des Verhaltens erläutert werden.

4.2.2. Manager

Der Manager hat ähnliche Aufgaben, wie ein Manager einer (kleinen) Firma: Er instruiert Untergebene und überwacht ihre Aktivitäten.

Der Manager verwaltet alle Ressourcen des Verhaltensnetzwerks und ist für ihre Initialisierung und ihre Destruktion genauso verantwortlich, wie für die Überwachung ihrer Funktionstüchtigkeit. Er ist der Besitzer aller Daten, der Signalemitter, des Signalprozessors und der Verhaltensschichten und vergibt entweder Referenzen auf diese Objekte oder verbindet die Objekte untereinander mit Hilfe von Signalen.

Die Kapselung aller anderen Komponenten in den Manager hat den Vorteil, dass sich dadurch die grundlegende Konfiguration des Systems an einer Stelle befindet. Um das System in einer automatisierten Testumgebung ausführen zu können, ist es z.B. nur nötig, den Manager durch einen speziellen Test-Manager auszutauschen. (Näheres hierzu siehe Abschnitt 4.2.8)

4.2.3. Datenabstraktion

Das Verhaltensnetzwerk greift nicht direkt auf die Datenobjekte mittels der Echtzeitdatenbank-C++-Schnittstelle zu, sondern verwendet eine Abstraktion dieser Schnittstelle. Diese Abstraktion stellt zum einen sicher, dass der Code des Verhaltensnetzwerks, der mit den Daten arbeitet, um die Logik der Verhaltensgenerierung durchzuführen, nicht zu eng mit der Echtzeitdatenbankschnittstelle verbunden ist. Zum anderen enthält die Abstraktion auch eine Erweiterung der Datenobjekte, die gleichzeitige Zugriffe aus unterschiedlichen Ausführungsfäden verhindert. Die Datenabstraktion kapselt also das eigentliche Datum ab und lässt nur Zugriffe über eine wohldefinierte Schnittstelle zu, durch die die Integrität des Datums und der serielle Zugriff auf es sichergestellt werden kann.

Um Verklemmungen[1] auszuschließen, existiert eine strenge Ordnung zwischen den Datenabstraktionen. Jeder Ausführungsfaden muss vor Benutzung einer Datenabstraktion diese für sich exklusiv reservieren. Eine Datenabstraktion kann nicht reserviert werden, wenn schon eine Datenabstraktion höherer Ordnung dem gleichen Ausführungsfaden zugewiesen wurde. Diese Regel, die praktisch sicherstellt, dass alle Ausführungsfäden die Datenabstraktionen jeweils in der gleichen Reihenfolge reservieren, verhindert ein zirkuläres Warten der einzelnen Ausführungsfäden aufeinander und somit ist gewährleistet, dass es nie zu einer Verklemmung kommen kann.[2]

Ein Beispiel soll dies verdeutlichen: Es gibt drei Ausführungsfäden (A, B und C) und drei Datenabstraktionen (α, β und γ). Jeder der Ausführungsfäden benötigt jede Datenabstraktion. Existiert keine starke Ordnung der Datenabstraktionen, kann jeder Ausführungsfaden die Datenabstraktionen in unterschiedlichen Reihenfolgen reservieren. Abbildung 4.4a zeigt eine Reihenfolge, die bei ungeschickter Serialisierung der Ausführungsfäden eine zirkuläre Abhängigkeit zwischen den Ausführungsfäden produzieren kann. In der Abbildung 4.4a ist der Ausführungsfaden A von B abhängig, weil B die Datenabstraktion β reserviert hat. Aber B ist auch von A abhängig, weil A die Datenabstraktion α für sich reservieren konnte: A wartet auf B und B wartet auf A; sie blockieren sich gegenseitig und diese Verklemmung behindert die Ausführung beider Ausführungsfäden.

Wenn nun aber eine starke Ordnung der Datenabstraktionen existiert und alle Ausführungsfäden die Datenabstraktionen in dieser Reihenfolge reservieren müssen, so kann es nicht zu einer solchen zirkulären Abhängigkeit zwischen den Ausführungsfäden kommen und das Problem des gegenseitigen Blockierens kann somit nicht auftreten. Abbildung 4.4b zeigt dieselbe Situation wie Abbildung 4.4a mit dem einzigen Unterschied, dass hier die Ausführungsfäden die starke Ordnung der Datenabstraktionen bei der Reservierung einhalten. Hier reserviert Ausführungsfaden A zuerst α und blockiert damit alle anderen Ausführungsfäden, die auch als erstes α benötigen. Er kann nun alle anderen Datenabstraktionen (β und γ) reservieren, seine Berechnung durchführen und alle Datenabstraktionen wieder freigeben woraufhin ein anderer Ausführungsfaden die von ihm benötigten Datenabstraktionen reservieren kann.

[1] Engl. deadlocks

[2] Andere Fehlerzustände wie z.B. das Verhungern eines Ausführungsfadens (starvation) werden durch diesen Mechanismus nicht verhindert und müssen durch Programmlogik ausgeschlossen werden.

Die Einführung einer starken Ordnung zur Verhinderung von zyklischen Abhängigkeiten ist nicht die einzige Möglichkeit, die Systemarchitektur-Lehrbücher wie (*Tanenbaum* 2002) kennen um Verklemmungen zu verhindern. Sie wurde in dieser Arbeit gewählt, weil sie eine höhere Parallelität als andere Verfahren unterstützt[1], eine feste Anzahl an Ressourcen vorhanden ist und eine (willkürliche) Ordnung leicht festgelegt und in dem abgeschlossenen System des Verhaltensnetzwerks leicht durchgesetzt werden kann.

4.2.4. Signalemitter

Signalemitter sind Ausführungsfäden, die nichts anderes tun, als auf eine bestimmte Bedingung zu warten und wenn diese eintritt, ein Signal zu emittieren und wieder auf das gleiche Ereignis zu warten. Ereignisse, auf die momentan gewartet werden kann, sind das Aktualisieren von Datenbank-Objekten mit einem bestimmten Namen und das Taktsignal eines Zeitgebers.

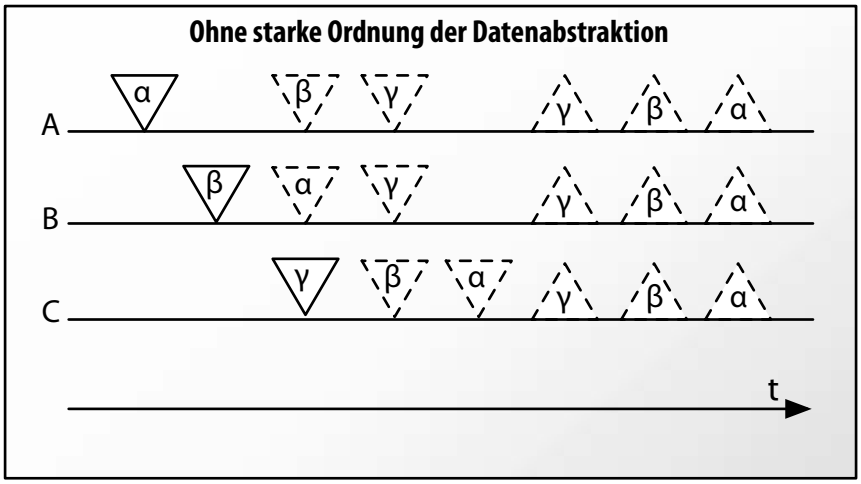
Durch Signalemitter wird ein prozessorintensives aktives Warten[2] auf die Bedingung verhindert.

4.2.5. Signalprozessor

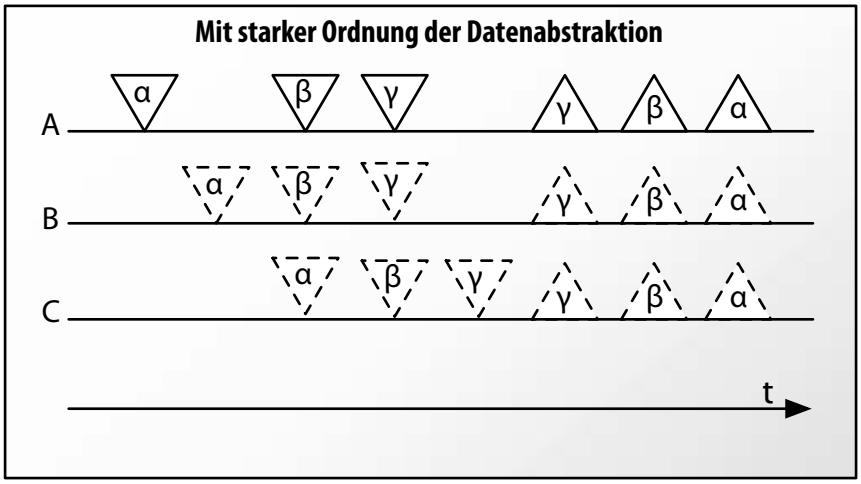
Der Signalprozessor wertet die Signale der Signalemitter aus und stösst die Ausführung der einzelnen Verhaltensschichten an, je nachdem, welche Signale in welchen Abständen eintreffen. Momentan sind die folgenden Regeln im Signalprozessor implementiert:

- Wenn die letzte Ausführung der strategischen Schicht schon mehr als 0,3 Sekunden zurückliegt, wird sie ausgeführt.
- Wenn das nicht der Fall ist und die letzte Ausführung der taktischen Schicht mehr als 0,2 Sekunden zurück liegt, wird die taktische Schicht ausgeführt.
- Wenn das nicht der Fall ist und die letzte Ausführung der reaktiven Schicht mehr als 0,1 Sekunden zurück liegt, wird die reaktive Schicht ausgeführt.
- Nach dem Ausführen der strategischen Schicht wird nochmals überprüft, ob die letzte Ausführung der taktische Schicht länger als 0,2 Sekunden in der Vergangenheit liegt; wenn dies der Fall ist, wird die taktische Schicht nach der strategischen ausgeführt.

[1] Wenn zwei Ausführungsfäden unterschiedliche Datenabstraktionen benutzen, können sie unabhängig voneinander laufen. Dies ist nicht bei allen Verfahren zur Verhinderung von zyklischen Abhängigkeiten gegeben.
[2] Engl. polling; Die Bedingung wird ständig erneut abgefragt bis sie eintritt.



(a) Eine mögliche Ausführungsreihenfolge ohne starke Ordnung der Datenabstraktionen.



(b) Eine mögliche Ausführungsreihenfolge mit starker Ordnung der Datenabstraktionen.

Abbildung 4.4.: Beispiel-Ausführungsreihenfolgen einmal ohne (a) und einmal mit (b) starker Ordnung der Datenabstraktionen.

- Auch nach dem Ausführen der taktischen Schicht wird nochmals geprüft, ob die reaktive Schicht in den letzten 0,1 Sekunden ausgeführt wurde; ist dies nicht der Fall, wird die reaktive Schicht ausgeführt.

- Nach dem Ausführen der reaktiven Schicht wird dem Manager ein Ausführen des kompletten Zyklus signalisiert.

Durch diese Regeln und einen Zeitgeber mit einem Intervall von 0,05 Sekunden wird sichergestellt, dass die strategische Schicht ca. alle 0,3 Sekunden, die taktische Schicht ca. alle 0,2 Sekunden und die reaktive Schicht ca. alle 0,1 Sekunden im Realbetrieb ausgeführt wird. Für den Simulationsmodus können diese Regeln aber einfach abgewandelt werden, so dass z.B. die einzelnen Schichten durch unterschiedliche Datenbankobjekte unabhängig voneinander ausgeführt werden können.

4.2.6. Verhaltensschichten

Die drei Schichten des Verhaltensnetzwerks sind in diesem Entwurf jeweils eigene Ausführungsfäden. Der Signalprozessor stößt den Lauf der Verhaltensschichten unabhängig voneinander an. Ein Durchlauf einer Verhaltensschicht macht dabei folgendes:

- Datenbankobjekte, die Verhalten dieser Schicht benötigen, werden aus der Datenbank geladen.

- Die virtuellen Sensoren (Aktivität, Zufriedenheit) aller Verhalten werden berechnet.

- Alle Verhalten dieser Schicht werden einmal ausgeführt.

- Datenbankobjekte, die verändert wurden, werden in die Datenbank zurückgeschrieben.

Jede Verhaltensschicht kann diesen Vorgang anpassen, so führt die strategische Schicht z.B. noch einen Punkt zwischen dem Lesen der Datenbankobjekte und dem Ausführen der Verhalten ein, in dem das strategische Verhalten motiviert wird, das von der Situationsinterpretation gerade über die Datenbank angefragt wurde.[1]

Die Verhaltensschichten verwalten die in ihnen vorhandenen Verhalten. Sie geben ihnen die Referenzen auf die benötigten Datenabstraktionen und stellen sicher, dass die Abhängigkeitsbeziehungen zwischen ihren Verhalten erfüllt sind.

[1] Die strategische Schicht besitzt die Besonderheit, dass nur jeweils eines ihrer Verhalten zu einem Zeitpunkt aktiv sein kann. Siehe *Hoffmann* (2007)

4.2.7. Verhaltensbausteine

Ein biologisch motiviertes Verhaltensnetzwerk kann aus unterschiedlichen Knoten bestehen: Den Verhalten und Fusionsknoten erster und zweiter Ordnung, wie sie in Abschnitt 2.1.6 beschrieben wurden. Auf die Knoten und ihre Unterscheidungsmerkmale soll hier nicht nochmals eingegangen werden; dieser Abschnitt befasst sich daher nur mit den Besonderheiten dieses Entwurfs in Hinblick auf die Verhaltensbausteine.

Eine Besonderheit ist, dass jedem Verhalten automatisch ein Fusionsknoten erster Ordnung zugewiesen wird. Dieser Fusionsknoten ist für andere Verhalten eine Fassade, über die sie Eingabeparameter für ein Verhalten, z.B. die Motivation eines Verhaltens, setzen können, ohne die Gefahr, dass Eingaben verloren gehen, weil sie von anderen Verhalten überschrieben werden. Zur Verdeutlichung dieser Gefahr stelle man sich zwei Verhalten A und B vor, die beide die Motivation von einem Verhalten C setzen. Verhalten A und B wissen nichts voneinander und können deshalb ihren Zugriff auf Verhalten C nicht koordinieren. Ohne eine Fusion ihrer Eingaben wird zufällig das Verhalten bevorzugt, dass später ausgeführt wird (Sei dies im Beispiel Verhalten B). Denn die Eingabe von Verhalten B überschreibt eine eventuelle Eingabe des Verhaltens A. Ein nicht vorhersagbares, inkonsistentes Systemverhalten ist die Folge.

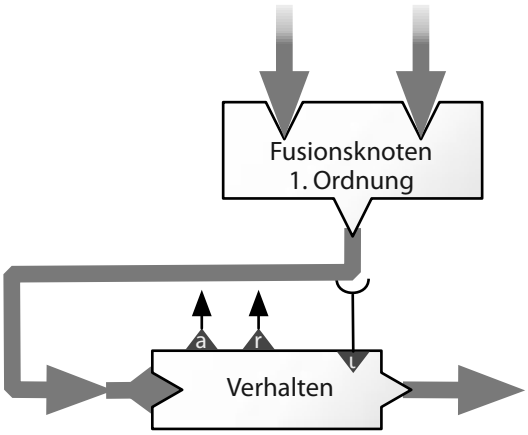


Abbildung 4.5.: Die Koppelung von Verhalten und Fusionsknoten 1. Ordnung: Jedem Verhalten ist ein Fusionsknoten 1. Ordnung vorgeschaltet. Alle anderen Verhalten können nur über diesen Fusionsknoten Eingabeparameter (wie die Motivation) an das Verhalten senden.

Da jedes Verhalten mit einem solchen Fusionsknoten 1. Ordnung gekoppelt ist, wird diese Koppelung im Folgenden als gegeben angenommen: Wenn im Folgenden also direkte Verbindungen zwischen Verhalten erwähnt oder abgebildet sind, denke sich der Leser eine Koppelung durch einen impliziten Fusionsknoten 1. Ordnung an ihrer Stelle.

Eine weitere erwähnenswerte Entwurfsentscheidung ist, die Abhängigkeiten eines Verhaltens so weit wie möglich explizit und in einer für das Verhaltensnetzwerk verständlichen Form zu spezifizieren. Ein Verhalten muss Lese- und Schreibzugriffe auf Daten und Abhängigkeiten zu anderen Verhalten bei der Initialisierung des Netzwerks angeben. Die explizite, maschinenlesbare Angabe ist aus verschiedenen Gründen von Vorteil:

- Die explizite Angabe von allen Abhängigkeiten zwingt den Programmierer, sich über alle eventuellen Abhängigkeiten des Verhaltens Gedanken zu machen.
- Zugriffe auf Daten, die nicht als Abhängigkeit für das Verhalten definiert wurden, resultieren in Laufzeitfehlern. Programmierfehler werden früher erkannt.
- Es ist möglich, die Abhängigkeiten automatisch grafisch darzustellen. Dies gewährleistet immer eine aktuelle grafische Darstellung des Verhaltensnetzwerks.
- Das Testen einzelner Verhalten wird erleichtert, da explizit bekannt ist, welche Abhängigkeiten vorliegen und so eine geeignete Testumgebung geschaffen werden kann.

4.2.8. Testumgebung

Um Anforderung 7 zu erfüllen wurde eine spezielle Testumgebung entworfen, die ein automatisiertes Testen von einzelnen Verhalten erlaubt. Durch die strikten Regeln, die Verhalten einhalten müssen um mit anderen Komponenten kommunizieren zu können (siehe Abschnitt Verhaltensbausteine, 4.2.7), kann die Testumgebung dem Verhalten eine Ausführungsumgebung zur Verfügung stellen, die sich aus Sicht des einzelnen Verhaltens nicht von der Ausführungsumgebung des Normalbetriebs unterscheidet.

Zu der Ausführungsumgebung gehören natürlich auch die von dem Verhalten benötigten Datenabstraktionen. Diese werden über sog. Fixturen zur Verfügung gestellt. Eine Fixtur ist ein Satz von Datenabstraktionen und ihrer Daten zu einem Zeitpunkt, der in einer Datei vorliegen und zu jeder Zeit von dieser wieder eingelesen werden kann. Vor jedem Test wird eine dieser Fixturen geladen, die alle Datenabstraktionen enthält, die das zu testende Verhalten benötigt. Zur Erstellung dieser Fixturen werden Hilfsmittel bereitgestellt, die z.B. das Erstellen einer Fixtur aus dem aktuellen Datenbankinhalt erlauben.

Nachdem eine Fixtur geladen wurde und die benötigten Verhalten erstellt wurden, kann der Testfall die Verhalten ausführen und ihre Ausgaben mit den erwarteten Ausgaben vergleichen. Stimmen die Ergebnisse mit der Erwartung überein, gilt ein Testfall als bestanden, ansonsten wird er als fehlerhaft markiert. Nach dem Vergleichen der Ausgabe mit den Erwartungen ist der Testfall beendet und alle benötigten Datenabstraktionen und Verhalten werden zerstört, um dem folgenden Testfall eine leere Ausführungsumgebung zur Verfügung stellen zu können.

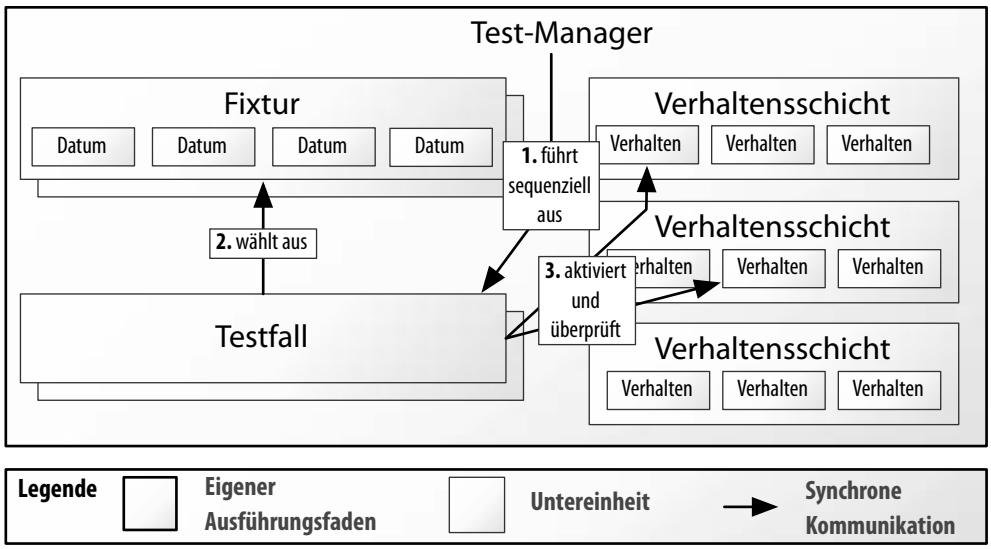


Abbildung 4.6.: Überblick über die Testumgebung.

Abbildung 4.6 zeigt einen grafischen Überblick über die Testumgebung, ihre Komponenten und die Interaktion zwischen ihnen.

Das aktuelle Design der Testumgebung begünstigt das Testen von der Ausgabe eines oder weniger Verhalten bei einer vorgegebenen und statischen Fixtur. Es wird nicht das Netzwerk als Ganzes getestet oder aber ein Verhalten über längere Zeit, sondern einzelne Aspekte eines einzelnen oder weniger Verhalten. Für Tests des Gesamtsystems und seiner Integration in die Systemarchitektur kann die im SFB/TR28 vorhandene Simulationsarchitektur verwendet werden. Die Testumgebung des Verhaltensnetzwerks soll sicherstellen, dass die einzelnen Verhalten sich ihrer Spezifikation entsprechend verhalten. Denn nur wenn sichergestellt ist, dass die einzelnen Komponenten das machen, was von ihnen erwartet wird, kann man sie zu einem Netzwerk verschalten und ein gemeinsames Verhalten mit ihnen generieren.

4.3. Entwurf des Verhaltensnetzwerks

Dieser Abschnitt geht auf das Verhaltensnetzwerk ein, das mithilfe des im vorherigen Abschnitts beschriebenen Frameworks entworfen und implementiert wurde. Er richtet den Fokus auf die Änderungen, die an dem schon bestehenden System bei der Konvertierung durchgeführt wurden. Das bestehende System wurde in *Hoffmann* (2007) ausführlich und in dieser Arbeit kurz in Abschnitt 2.1.7 vorgestellt.

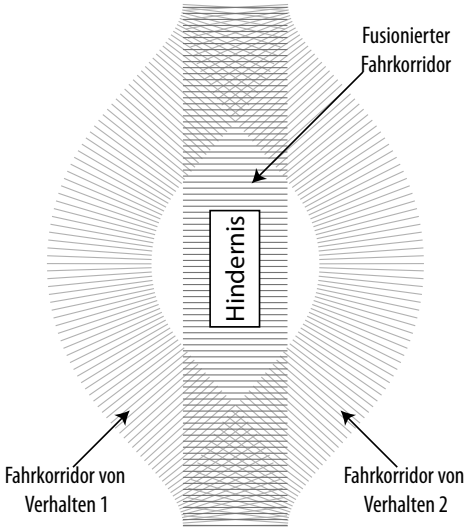


Abbildung 4.7.: Fahrkorridore zweier Verhalten und ihre Fusion. Beide Verhalten vermeiden ein Hindernis auf unterschiedliche Weise. Der fusionierte Fahrkorridor führt genau durch das Hindernis.

4.3.1. Schnittstelle zur Bahnplanung

Die Bahnplanung war in einer einfachen Version Teil des Verhaltensnetzwerks im Stand von *Hoffmann* (2007). Das Verhaltensnetzwerk hatte also eine direkte Schnittstelle zur Fahrzeugregelung. In diesem neuen Entwurf besitzt das Verhaltensnetzwerk stattdessen eine Schnittstelle zu einer gesonderten Bahnplanungskomponente, die wiederum der Fahrzeugregelung die Regelgrößen liefert.

Diese Aufteilung wurde aus folgenden Gründen vorgenommen:

- Mit der alten Schnittstelle waren Situationen denkbar, in denen das Fahrzeug in ein Hindernis gefahren wäre, anstatt es zu umfahren, weil zwei Verhalten widersprüchliche Aktionen geplant haben und die fusionierte Aktion die ursprüngliche Logik nicht mehr enthielt. Abbildung 4.7 zeigt so eine Situation.

- Die Verhaltensgenerierung arbeitet nur mit Wahrnehmungsdaten auf Objektebene. Nicht alle Objekte werden eindeutig und zuverlässig erkannt. So sind beispielsweise Bordsteinkanten nicht als Objekte vorgesehen, müssen aber trotzdem von der Bahnplanung berücksichtigt werden, um einen validen Fahrweg für das Auto zu generieren.

- Die gesonderte Bahnplanungskomponente besitzt ein fahrdynamisches Modell. Mit dessen Hilfe kann sie Trajektorien generieren, die besser als die vorherig verwendeten Fahrkorridore der realen Regelung entsprechen. Hierdurch kann auf einen Toleranzbereich

bei dieser Schnittstelle verzichtet werden und das Problem, dass mehrere Iterationen von Fahrkorridoren notwendig werden, wenn die Regelung den Toleranzbereich nicht einhalten kann, wird vermieden.

- Durch die Aufspaltung in Verhaltensgenerierungsschicht und Bahnplanungsschicht können beide Schichten unabhängig voneinander entwickelt und getestet werden.

Die Schnittstelle zur Bahnplanung besteht aus zwei Datenbankobjekten: Ein Objekt enthält Daten wie zulässige Geschwindigkeit, Fahrtrichtung etc. und das andere Datenbankobjekt enthält Karten der Umgebung des autonomen Automobils. In diesen sog. *Risikokarten* werden von dem Verhaltensnetzwerk die Bereiche markiert, die in einem bestimmten Zeitpunkt bevorzugt befahren werden sollen. Die Risikokarte vereint eine normale Befahrbarkeitskarte mit den Fahrintentionen des Verhaltensnetzwerks. Die Bahnplanung sucht innerhalb der Risikokarte eine mit den aktuellen Rahmenbedingungen fahrbare Trajektorie und liefert sie der Fahrzeugregelung zur Ausführung.

Dieser Abschnitt beschäftigt sich im Folgenden mit den Risikokarten und ihrer Erstellung innerhalb des Verhaltensnetzwerks. Die endgültige Fahrintention ergibt sich erst aus dem Zusammenspiel mehrerer Verhalten und so muss auch die Risikokarte, die ja die Fahrintention enthält, durch mehrere Verhalten erstellt werden. Jedes beteilige Verhalten erstellt eine eigene Risikokarte und die Bahnplanung erhält eine gemeinsame, fusionierte Karte.

Eine Risikokarte ist eine Graustufen-Rastergrafik, bei der jeder Pixel einen quadratischen Bereich in der Umgebung des Autos zu einem bestimmten Zeitpunkt widerspiegelt. Die 8-Bit-Farbinformation eines Pixels kodiert das Risiko, diesen Bereich zu befahren. Null ist das geringste Risiko, 255 das höchste. Pixel mit Farbwerten über 250 dürfen von der Bahnplanung auf keinen Fall in die Pfadplanung einfließen, denn sie repräsentieren Hindernisse wie Autos oder Fußgänger. Pixel mit Farbwerten von null bis 250 sind befahrbare Bereiche. Eine Risikokarte repräsentiert das Risiko zu einem bestimmten Zeitpunkt; die Verhalten erzeugen Risikokarten nicht nur für den aktuellen Zeitpunkt sondern auch für Zeitpunkte in der Zukunft, so dass die Bahnplanung Trajektorien planen, die auch vorausgesagte Ereignisse berücksichtigen.

Abbildung 4.8 erläutert den mehrstufigen Fusionsprozess, der nötig ist, um aus den einzelnen Risikokarten der verschiedenen Verhalten eine gemeinsame Darstellung zu generieren. Jedes Verhalten, das Risikokarten als Ausgabe hat, kann in eine von zwei Klassen eingeteilt werden: Zum einen Verhalten, die befahrbare Bereiche beschreiben und zum anderen Verhalten, die nicht befahrbare Bereiche beschreiben. Diese beiden Klassen müssen bei der Fusion unterschiedlich behandelt werden, um eine sinnvolle Fusion zu erreichen: Risikokarten der Verhalten der ersten Klasse müssen mittels einer Minimumfusion verschmolzen werden, die restlichen Risikokarten müssen mit der daraus resultierenden Risikokarte durch Maximumfusion verbunden werden.

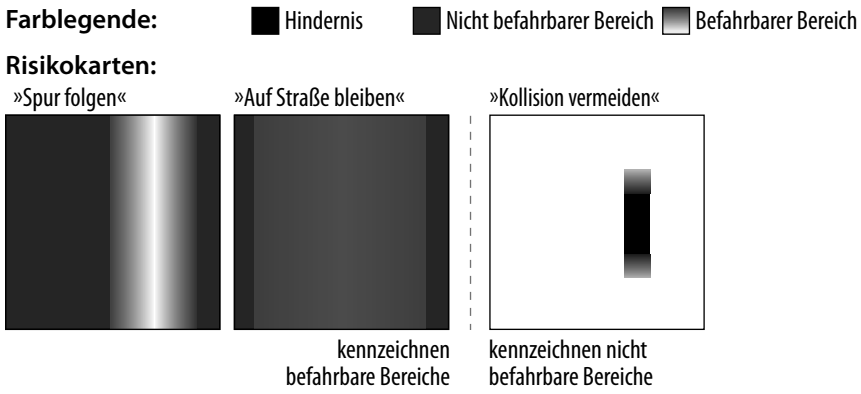


Abbildung 4.8.: Die Fusion von Risikokarten am Beispiel der Risikokarten der Verhalten *Spur halten*, *Auf Straße bleiben* und *Kollision vermeiden*. Die Fusion erfolgt in zwei Schritten: eine Minimumfusion aller Karten, die befahrbare Bereiche beschreiben und eine Maximumfusion des Ergebnisses aus Schritt 1 mit allen Karten, die nicht befahrbare Bereiche beschreiben.

4.3.2. Schnittstelle zur Situationsinterpretation

Die Situationsinterpretation ist eine gesonderte Systemkomponente, die die Weltmodellierung und Situationsanalyse übernimmt. Sie wird unter anderem als eigenständige Komponente entwickelt, weil diese Weltmodellierung nicht nur für die Verhaltensentscheidung benötigt wird, sondern z.B. auch für ein Fahrerassistenzsystem verwendet werden kann. Ein denkbares Fahrerassistenzsystem, das das Weltmodell der Situationsinterpretation verwenden könnte, ist ein Warnsystem, das den Fahrer vor gefährlichen Situationen warnen kann. Ein weiterer Grund, der dafür spricht, die Situationsinterpretation als eine eigenständige Komponente zu entwickeln, ist die gesteigerte Nachvollziehbarkeit des Gesamtsystems.

Die Situationsinterpretation liefert dem Verhaltensnetzwerk über die Echtzeitdatenbank das umgebende Straßennetz, die Verkehrsteilnehmer und ihre Verhalten und Beziehungen zueinander – z.B. welches Fahrzeug sich auf welcher Spur befindet aber auch welches Fahrzeug Vorfahrt besitzt. Außerdem liefert es die aktuell in der Mission zu fahrenden Fahrspuren. Auf diese Informationen kann aus Verhalten aller Schichten zugegriffen werden.

Die Fahrspuren werden als Menge von Punktpaaren definiert. Jedes Punktpaar stellt die rechte und linke Begrenzung der Fahrspur dar. Aufeinanderfolgende Punkte werden durch Liniensegmente miteinander verbunden. Jede Fahrspur besitzt Beziehungen zu anderen Fahrspuren. Mit diesen Beziehungen können die Nachbarspuren oder nachfolgende Spuren ermittelt werden. Außerdem besitzen Fahrspuren Beziehungen zu dynamischen Objekten. Durch diese Beziehungen kann ermittelt werden, welches Objekt sich auf welcher Fahrspur befindet. Dynamische Objekte werden als quaderförmig modelliert. Sie besitzen einen Schwerpunkt, Länge, Breite und Höhe und eine Ausrichtung in alle drei Dimensionen. Neben diesen Attributen besitzen dynamische Objekte weitere Attribute, wie z.B. ihr vorhergesagtes Verhalten, und Beziehungen zu anderen Fahrzeugen. Durch diese Beziehungen können z.B. Vorfahrtsrelationen ausgedrückt werden.

Zusätzlich zu diesen direkten Informationen der Situationsinterpretation existiert eine Komponente, die diese Schnittstelle erweitert: der Szenariomonitor. Er wurde in (*Jagszent* u. a. 2007; *Jagszent* 2007) genauer beschrieben, deshalb soll hier nur ein grober Überblick genügen: Der Szenariomonitor ist Schnittstelle zwischen Situationsinterpretation und den strategischen Verhalten. Er bündelt und abstrahiert das von den strategischen Verhalten benötigte Wissen über Verkehrsregeln. Abbildung 4.9 zeigt, wie er sich in die Systemarchitektur einfügt. Über ihn kann ein strategisches Verhalten die Bedingungen abfragen, unter denen ein bestimmtes taktisches Verhalten in der aktuellen Szene, wie sie von der Situationsinterpretation zur Verfügung gestellt wird, ausgeführt werden darf.

4.3.3. Netzwerkstruktur

Dieser Abschnitt stellt die in dieser Arbeit entwickelte Struktur des Verhaltensnetzwerks vor, das mithilfe des am Anfang dieses Kapitels vorgestellten Frameworks entwickelt wurde. Ab-

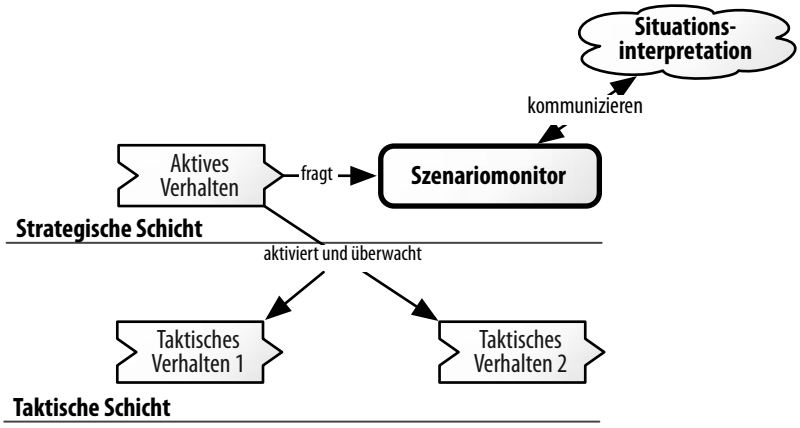


Abbildung 4.9.: Der Szenariomonitor und seine Interaktion mit anderen Systemkomponenten. (Abgewandelt aus *Jagszent* 2007, Abbildung 3.2)

bildung 4.10 gibt eine komplette Übersicht über das Netzwerk, das im Folgenden anhand der Schichten genauer betrachtet werden soll.

Strategische Schicht

Die strategische Schicht enthält die beiden Verhalten *Straße folgen* und *4-Way-Stop passieren*.[1]

Verhalten dieser Schicht schließen sich gegenseitig aus. Es kann immer nur ein Verhalten zur Zeit motiviert sein. Dies wird von der Klasse, die die strategische Schicht implementiert, sichergestellt. Sie ist auch dafür zuständig, die aktuelle Mission der Situationsinterpretation auszulesen und das für diese zuständige strategische Verhalten auszuwählen. Die einzig möglichen Missionsbefehle zur Zeit sind *normales Fahren* und *4-Way-Stop passieren*. Zu diesen beiden Missionsbefehlen sind die beiden Verhalten *Straße folgen* und *4-Way-Stop passieren* die Entsprechungen. Beide Verhalten instrumentieren die taktischen Verhalten, um den entsprechenden Missionsbefehl ausführen zu können. Diese Instrumentierung soll im Folgenden am Beispiel von *4-Way-Stop passieren* erläutert werden:

- Beim Anfahren an den 4-Way-Stop wird nur das Verhalten *Spur folgen* motiviert.
- Ist das Eigenfahrzeug nahe an der Kreuzung, wird zusätzlich noch das Verhalten *Anhalten* motiviert und instrumentiert, so dass es an der Stopplinie der Kreuzung anhält.
- Wenn die Stopplinie erreicht ist, wird das Verhalten *Stehen* noch zusätzlich motiviert, bis das Eigenfahrzeug die Vorfahrt vor allen anderen Fahrzeugen erlangt hat.

[1] Im Quelltext werden sie durch die Klassen `SB_FollowStreet` und `SB_4WayStop` repräsentiert.

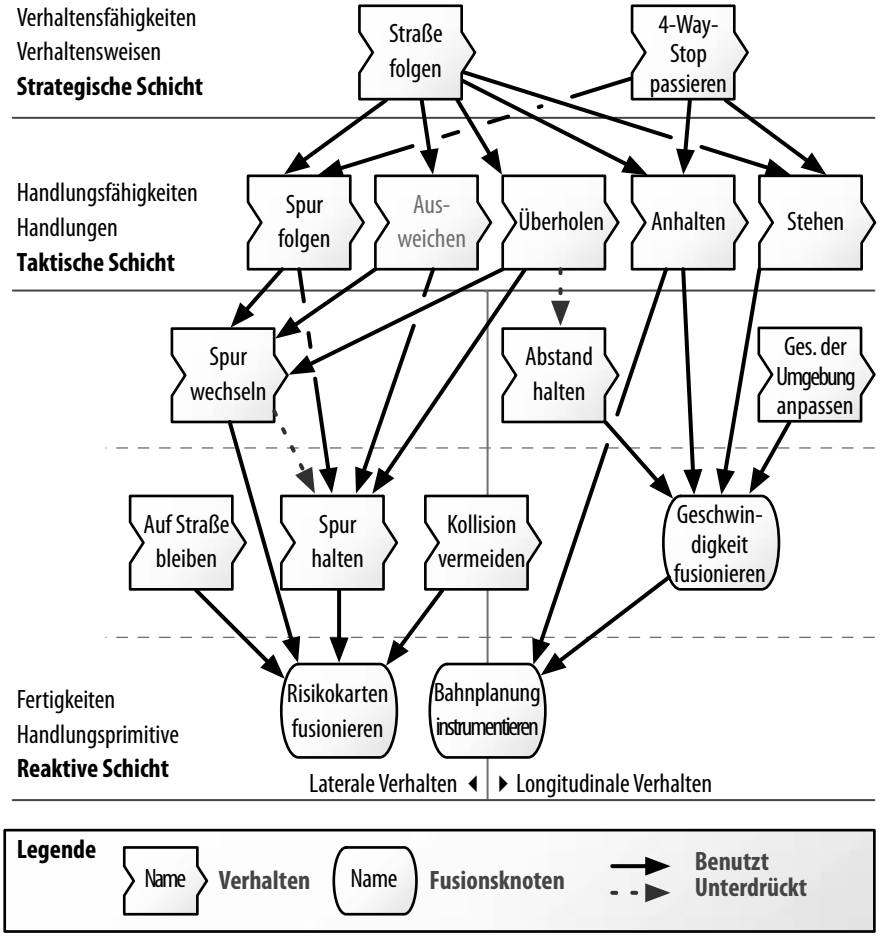


Abbildung 4.10.: In dieser Arbeit entwickelte Netzwerkstruktur des Verhaltensnetzwerks. Das Netzwerk enthält 16 Knoten und besteht aus drei Schichten, wobei die reaktive Schicht nochmals in drei Zwischenschichten unterteilt werden kann.

- Da das Eigenfahrzeug nun die Kreuzung passieren darf, werden die Verhalten *Anhalten* und *Stehen* demotiviert und das die ganze Zeit über aktive Verhalten *Spur folgen* ist wieder das einzige taktische Verhalten, das motiviert ist.

Taktische Schicht

In der taktischen Schicht sind die fünf Handlungsfähigkeiten *Anhalten*, *Stehen*, *Spur folgen*, *Ausweichen* und *Überholen* zu finden.[1]

[1] Im Quelltext werden diese Verhalten durch die Klassen `TB_Stop`, `TB_StandStill`, `TB_FollowLane`, `TB_AvoidObstacle` und `TB_Overtaking` repräsentiert.

Spur folgen versucht der von der Situationsinterpretation vorgegebenen Spur zu folgen. Wenn sich das Eigenfahrzeug aktuell auf einer anderen Spur befindet, versucht dieses Verhalten durch gezieltes instrumentieren des reaktiven Verhaltens *Spur wechseln* wieder auf die von der Situationsinterpretation vorgegebenen Spur zu fahren. Befindet sich das Eigenfahrzeug auf der gewünschten Spur, wird diese durch Motivierung des Verhaltens *Spur halten* gehalten.

Statische Hindernisse innerhalb der aktuellen Spur, wie z.B. parkende Autos oder generell in die Fahrbahn hineinragende Objekte, können durch das taktische Verhalten *Ausweichen* bewusst vermieden werden. Wenn möglich wird dies innerhalb der Fahrspur durch instrumentieren des reaktiven Verhalten *Spur halten* erreicht. Sollte es nicht möglich sein und es existiert eine Nachbarspur, wird vor dem Hindernis angehalten, auf die Nachbarspur ausgeschert, das Hindernis umfahren und wieder auf die ursprüngliche Spur gewechselt. Dieses Verhalten konnte leider noch nicht vollständig fertiggestellt werden. Zum einen, weil die Zeit hierfür nicht mehr ausreichte und zum anderen, weil im SFB/TR28 noch keine Wahrnehmungskomponente existiert, die solche statischen Hindernisse erkennen könnte.

Das Verhalten *Überholen* kann fahrende oder stehende Fahrzeuge überholen, falls die Verkehrsregeln das Überholen zulassen und die Gegenspur für den Überholvorgang frei ist. Dieses Verhalten parametrisiert das reaktiven Verhalten *Spur wechseln*, um vor dem Überholen auf die Gegenspur zu wechseln und nach dem Überholen wieder auf die Ausgangsspur zurück zu wechseln. Während des Überholens wird *Spur halten* motiviert, bis das zu überholende Fahrzeug passiert ist. Während auf die Gegenspur gewechselt wird, demotiviert *Überholen* das reaktive Verhalten *Abstand halten*, um vor dem Ausscheren möglichst dicht auf das vorausfahrende Fahrzeug auffahren zu können.

Anhalten kann so instrumentiert werden, dass es an einer bestimmten Position anhält. Dafür wird die Geschwindigkeit entsprechend der Distanz zur Stoppposition angepasst und die Bahnplanung wird über die gewünschte Stoppposition informiert, damit diese einen entsprechenden Pfad planen kann, der an dieser Position aufhört. Dieses Verhalten ist in der taktischen Schicht und nicht in der reaktiven Schicht eingeordnet, weil es eine bewusste, kurzfristig planbare Handlung ist, an einem bestimmten Punkt stehenzubleiben.

Das Verhalten *Stehen* grenzt sich von *Anhalten* derart ab, dass die Handlung, die hiermit modelliert wird, das bewusste Stillstehen ist. Es versucht nicht, einen bestimmten Punkt zu erreichen, sondern nur, das Auto anzuhalten. Dieses Verhalten kann z.B. aktiviert werden, um an einer befahrenen Kreuzung sicherzustellen, dass das Auto sich nicht mehr bewegt.

Reaktive Schicht

Die reaktive Schicht enthält sechs Handlungsprimitive und drei Fusionsknoten. Die Handlungsprimitive lassen sich in zwei Klassen unterteilen: In laterale und longitudinale Verhalten. Die lateralen Verhalten, zu denen *Spur halten*, *Spur wechseln* und *Auf Straße bleiben* gehören, regeln die laterale Position des Eigenfahrzeugs. Verhalten in der longitudinalen Klasse, *Abstand halten* und *Geschwindigkeit der Umgebung anpassen*, regeln die Richtgeschwindigkeit für

die Bahnplanung. Das Verhalten *Kollision vermeiden* kann nicht eindeutig dieser Klassifikation zugeordnet werden, es ist aber hauptsächlich in der lateralen Klasse eingeordnet.[1]

Das Verhalten *Spur halten* erzeugt eine Risikokarte für die aktuell befahrene Spur. Diese Spur muss nicht Teil der Mission sein, die von der Situationsinterpretation vorgegeben wurde. So kann z.B. beim Überholvorgang mithilfe dieses Verhaltens auch auf der Gegenspur gefahren werden, obwohl diese nicht Teil der Mission ist. Die laterale Position innerhalb der Spur kann dem Verhalten als Eingabe vorgegeben werden.

Spur wechseln erzeugt eine Risikokarte, die einen Spurwechsel von einer bestimmten Spur zu ihrer linken oder rechten Nachbarspur erlaubt. Während dieses Verhalten motiviert ist, demotiviert es automatisch das *Spur halten*-Verhalten, denn die beiden Handlungsprimitive schließen sich gegenseitig aus.

Eine Risikokarte, die den gesamten befahrbaren Bereich enthält, wird durch das Verhalten *Auf Straße bleiben* erzeugt. Dieses Verhalten ist ein Sicherheitsverhalten, das der Bahnplanung die befahrbaren Bereiche bekanntgibt. Das Risiko, das dieses Verhalten den befahrbaren Bereichen zuweist ist allerdings so hoch, dass sie nur in Ausnahmefällen befahren werden. Das Verhalten *Kollision vermeiden* erzeugt eine weitere Risikokarte, in der alle Hindernisse und Fahrzeuge als nicht befahrbare Bereiche markiert werden. Jeder nicht befahrbare Bereich erhält einen Sicherheitsbereich, der zwar befahrbar ist, aber ein ansteigendes Potential aufweist. Dieses Verhalten ist momentan das einzige, das in einem Schritt unterschiedliche Karten erzeugt. Die Fahrzeuge werden ihrer aktuellen Geschwindigkeit und Ausrichtung entsprechend in die Zukunft prädiziert und so in Karten für zukünftige Zeitpunkte eingetragen.

Jedes dieser vier Verhalten erzeugt Risikokarten, die durch den Fusionsknoten *Risikokarten fusionieren* in gemeinsame Risikokarten zusammengeführt werden. Der Algorithmus, mit dem diese Fusion durchgeführt wird um sicherzustellen, dass die Fahrintention der einzelnen Verhalten bestmöglich bestehen bleiben, wird in Abschnitt 4.3.1 erläutert.

Das logitudinale Verhalten *Abstand halten* kontrolliert die Geschwindigkeit eines vorausfahrenden Fahrzeugs und übermittelt dem Fusionsknoten *Geschwindigkeit fusionieren* eine angepasste Geschwindigkeit, damit das Eigenfahrzeug dem vorausfahrenden Fahrzeug in einem sicheren Abstand folgt. Der Sicherheitsabstand kann durch die Motivation des Verhaltens eingestellt werden. *Abstand halten* ist ein Sicherheitsverhalten, das unabhängig von den gerade aktivierten taktischen Verhalten motiviert ist. Eine Ausnahme bildet das taktische Verhalten *Überholen*, das *Abstand halten* für den kurzen Bereich des Ausscherens für das Überholen demotiviert.

Geschwindigkeit der Umgebung anpassen ist ein weiteres Sicherheitsverhalten, das dem *Geschwindigkeit-fusionieren*-Fusionsknoten eine Geschwindigkeitsbegrenzung übermittelt, die sich zum einen aus der aktuell erlaubten Höchstgeschwindigkeit und zum anderen aus dem aktuellen Straßenverlauf ergibt. Eine kurvige Straße verlangt eine langsamere Geschwindigkeit als eine

[1] Im Quelltext werden diese Verhalten durch die Klassen `RB_KeepLane`, `RB_Veer`, `RB_KeepOnStreet`, `RB_KeepDistance`, `RB_ContextSpeedGuide` und `RB_AvoidCollision` repräsentiert.

gerade. Die Krümmung der Straße ist momentan das einzige Kriterium, durch das die Höchstgeschwindigkeit angepasst wird. Denkbar ist auch, die aktuelle Wetterlage, die Beschaffenheit des Bodenbelags und der Reifen in die Berechnung mit einzubeziehen.

Der Fusionsknoten *Geschwindigkeit fusionieren* errechnet die Geschwindigkeit, die der Bahnplanung als Richtgeschwindigkeit übermittelt wird. Dies geschieht durch eine Minimumsfusion aller eingehenden Geschwindigkeiten. Die resultierende Geschwindigkeit wird dem *Bahnplanung-instrumentieren*-Fusionsknoten übergeben. Dieser Knoten sammelt alle notwendigen Parameter für die Bahnplanung, wie z.B. die Richtgeschwindigkeit oder den Planungshorizont.

4.3.4. Virtuelle Sensoren und Kooperation von Verhalten

In diesem Abschnitt werden exemplarisch die Berechnungsfunktionen für Reflexion und Aktivität von einem reaktiven und einem taktischen Verhalten vorgestellt. Außerdem wird an einem Beispiel gezeigt, wie Kooperation mithilfe dieser virtuellen Sensoren und der Motivation in dem in dieser Arbeit entwickeltem Verhaltensnetzwerk erreicht wurde.

Reflexion des Verhaltens *Spur wechseln*

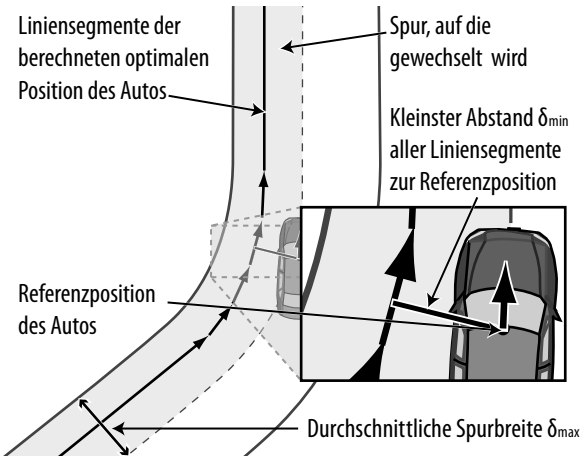


Abbildung 4.11.: Berechnung der Reflexion des Verhaltens *Spur wechseln*.

Die Reflexion eines Verhaltens kann sich aus physikalischen Grössen berechnen, die ein Maß für die Zufriedenheit dieses Verhaltens darstellen. Das Verhalten *Spur wechseln* ist hierfür ein Beispiel, das im Folgenden näher erläutert wird.

Spur wechseln ist zufrieden, wenn das Auto die Mitte der Fahrspur erreicht hat, auf die es wechseln soll. Je weiter das Fahrzeug von dieser optimalen Position entfernt ist, umso unzufriedener ist das Verhalten. Da die Reflexion auf das Intervall zwischen 0 und 1 skaliert ist, muss dieser Abstand zur optimalen Position entsprechend durch einen maximal tolerierten Abstand geteilt werden.

Der Einfachheit halber wird bei der Erstellung der Risikokarte von *Spur wechseln* die optimale Position in der Fahrspur, auf die gewechselt werden soll, anhand von Liniensegmenten ermittelt. Für die Berechnung der Reflexion wird deshalb die minimale Distanz δ_{min} des Referenzpunktes (siehe Abschnitt 3.1.2) zu diesem Linienzug ermittelt. Der maximal tolerierte Abstand δ_{max} ist die Hälfte der Breite der Fahrspur, auf die gewechselt werden soll. Die Reflexion berechnet sich demnach wie folgt:

$$r_{\text{Spur wechseln}} := \begin{cases} 0 & \text{wenn} \quad \frac{\delta_{min}}{\delta_{max}} \leq 0 \\ 1 & \text{wenn} \quad \frac{\delta_{min}}{\delta_{max}} \geq 1 \\ \frac{\delta_{min}}{\delta_{max}} & \text{wenn } 0 < \frac{\delta_{min}}{\delta_{max}} < 1 \end{cases}$$

Abbildung 4.11 zeigt die für die Berechnung herangezogenen Größen an einem Beispiel.

Reflexion des Verhaltens *Überholen*

Die Reflexion des *Überholen*-Verhaltens ist nicht durch physikalische Größen ausgedrückt, sondern spiegelt zum einem den Wunsch zu überholen wider und zum anderen – wenn ein Überholvorgang durchgeführt wird – die Phase, in der sich der Überholvorgang befindet:

$$r_{\text{Überholen}} := \begin{cases} 0 & \text{wenn nicht überholt wird und auch kein Wunsch besteht.} \\ 0,9 & \text{wenn nicht überholt wird aber ein Wunsch besteht.} \\ 0,5 & \text{wenn ein Überholvorgang abgebrochen werden muss.} \\ 0,2 & \text{wenn auf die Überholspur gewechselt wird.} \\ 0,15 & \text{wenn überholt wird.} \\ 0,125 & \text{wenn auf die ursprüngliche Spur gewechselt wird.} \end{cases}$$

Ein strategisches Verhalten, das einen Überholvorgang überwacht, ist nicht so sehr an dem speziellen Wert der Reflexion interessiert, sondern daran, dass der Wert mit der Zeit abnimmt. Befindet sich das Auto in einem Überholvorgang und die Reflexion des *Überholen*-Verhaltens bleibt längere Zeit konstant, kann das strategische Verhalten gegensteuern. Z.B. könnte es das *Überholen*-Verhalten demotivieren und das Verhalten *Spur folgen* motivieren und damit den Überholvorgang abbrechen.

Fortschrittsbestimmung des Verhaltens *Überholen*

Im vorherigen Abschnitt wurde die Reflexion des *Überholen*-Verhaltens teilweise anhand des Fortschritts definiert, den es erreicht hat. Zur Bestimmung dieses Fortschritts wird unter anderem auf die in Abschnitt 4.3.4 erläuterte Reflexion des Verhaltens *Spur wechseln* zurückgegriffen: Um zu bestimmen, ob das Fahrzeug beim Einscheren in die Überholspur die Zielspur schon erreicht hat, überprüft das *Überholen*-Verhalten ständig die Reflexion des Verhaltens *Spur wechseln*. Sinkt die Reflexion unter einen festgelegten Wert, wird der Spurwechsel als abgeschlossen angesehen und die nächste Phase kann durchgeführt werden.

Das *Überholen*-Verhalten besitzt die folgenden Phasen:

- *Kein Überholwunsch*
  In dieser Phase befindet sich das Verhalten *Überholen*, wenn es keinen Bedarf zum überholen erkennen kann oder wenn es aufgrund des Verkehrs oder der Straßensituation nicht möglich ist, zu überholen.

- *Überholwunsch*
  Diese Phase wird nach der *Kein-Bedarf*-Phase aktiviert, wenn der Wunsch zu überholen gegeben ist und wenn das Überholen möglich ist. In dieser Phase wird so lange verweilt, bis entweder das Überholen nicht mehr möglich ist – dann wird in die *Abbruch*-Phase gewechselt – oder ein strategisches Verhalten das *Überholen*-Verhalten motiviert hat und ihm somit die Erlaubnis erteilt hat, den Überholvorgang zu beginnen. In diesem Fall wird in die Phase *Spurwechsel auf Gegenspur* gewechselt.

- *Spurwechsel auf Gegenspur*
  In dieser Phase wird auf die Gegenspur gewechselt. Wurde die Gegenspur erreicht, wechselt das *Überholen*-Verhalten in die *Überholvorgang*-Phase.

- *Überholvorgang*
  Diese Phase wird wieder verlassen, wenn das zu überholende Fahrzeug vollständig überholt wurde. Dann wird in die Phase *Spurwechsel auf Ausgangsspur* gewechselt. Sollte während dieser Phase das zu überholende Fahrzeug verschwinden (Fehlererkennung) wird in die *Abbruch*-Phase gewechselt.

- *Spurwechsel auf Ausgangsspur*
  In dieser Phase wird wieder auf die Ausgangsspur gewechselt. Ist dieser Spurwechsel vollzogen, wird in die *Kein-Überholwunsch*-Phase gewechselt.

- *Abbruch*
  Diese Fehlerbehandlungsphase wird immer aktiviert, wenn ein Fehlerfall aufgetreten ist. In ihr wird die notwendige Aktion durchgeführt, um wieder auf die Ausgangsspur zu gelangen und danach wird in die Phase *Kein Überholwunsch* gewechselt.

Übersicht der Berechnungsmethoden für Reflexion und Aktivität

In den vorherigen Abschnitten wurde exemplarisch die Berechnung der Reflexionen der Verhalten *Spur wechseln* und *Überholen* vorgestellt. In diesem Abschnitt soll nun der Vollständigkeit halber die Berechnung der Reflexion und Aktivität aller Verhalten aufgelistet werden. Eine genauere Betrachtung aller Berechnungen entfällt hier aus Platzgründen.

In der nachfolgenden Tabelle werden die folgenden Notationen und Abkürzungen verwendet:

$clamp(x)$	beschränkt den Wert x auf das Intervall $[0, 1]$.
m	ist die Motivation des Verhaltens.
δ_{min}	ist der minimale Abstand zwischen der berechneten optimalen Position und der tatsächlichen Fahrzeugposition.
δ_{max}	ist die durchschnittliche Spurbreite.
δ_n	ist die Distanz bis zum nächsten dynamischen Objekt.
δ_s	ist die Distanz bis zur Stopplinie.
v_c	ist die aktuelle Geschwindigkeit des Eigenfahrzeugs.
v_d	ist die Geschwindigkeit, die das Verhalten aktuell fahren möchte.
s_s	ist der Sicherheitsabstand, der zum vorausfahrenden Fahrzeug eingehalten werden soll.
s_b	ist der Abstand, der benötigt wird, um das Eigenfahrzeug mit einer Nominalbeschleunigung von $1\ \frac{m}{s}$ auf die Geschwindigkeit des vorausfahrenden Fahrzeugs zu bringen.
s_r	ist der Abstand, der benötigt wird, um das Eigenfahrzeug mit einer Nominalbeschleunigung von $\frac{1}{2}\ \frac{m}{s}$ auf die Geschwindigkeit des vorausfahrenden Fahrzeugs zu bringen.

Reaktive Verhalten		
Verhalten	**Reflexion**	**Aktivität**
Spur halten	$clamp\left(\frac{\delta_{min}}{\delta_{max}}\right)$	$clamp\left[\frac{1}{2} + clamp\left(\frac{\delta_{min}}{\delta_{max}} \cdot \frac{1}{2}\right)\right] \cdot m$
Spur wechseln	$clamp\left(\frac{\delta_{min}}{\delta_{max}}\right)$	$clamp\left[\frac{1}{2} + clamp\left(\frac{\delta_{min}}{\delta_{max}} \cdot \frac{1}{2}\right)\right] \cdot m$
Kollision vermeiden	$1 - clamp\left(\frac{\delta_n}{10}\right)$	$clamp\left[1 - clamp\left(\frac{\delta_n}{10}\right)\right] \cdot m$
Auf Straße bleiben	$clamp\left(\frac{\delta_{min}}{\delta_{max}}\right)$	$clamp\left[\frac{1}{2} + clamp\left(\frac{\delta_{min}}{\delta_{max}} \cdot \frac{1}{2}\right)\right] \cdot m$
Abstand halten	$1 - clamp\left(\frac{\sqrt{\delta_n - s_s - s_b}}{\sqrt{s_r - s_b}}\right)$	$\left[1 - clamp\left(\frac{\sqrt{\delta_n - s_s - s_b}}{\sqrt{s_r - s_b}}\right)\right] \cdot m$
Geschwindigkeit der Umgebung anpassen	$clamp\left(\frac{v_c - v_d + \frac{1}{4}}{5\frac{1}{4}}\right)$	$clamp\left(\frac{1}{2} + \left[clamp\left(\frac{v_c - v_d + \frac{1}{4}}{5\frac{1}{4}}\right)\right] \cdot \frac{1}{2}\right) \cdot m$
Geschwindigkeit fusionieren	$clamp\left(\frac{\lvert v_c - v_d \rvert}{10}\right)$	$1 \cdot m$

Taktische Verhalten		
Verhalten	**Reflexion**	**Aktivität**
Spur folgen	$clamp\left(\frac{\delta_{min}}{\delta_{max} \cdot \frac{6}{5}}\right)$	$clamp\left[\frac{1}{2} + clamp\left(\frac{\delta_{min}}{\delta_{max} \cdot \frac{6}{5}} \cdot \frac{1}{2}\right)\right] \cdot m$
Überholen	siehe Abschnitt » Reflexion des Verhaltens *Überholen* «	
Stehen	$clamp\left(\frac{v_c}{2}\right)$	$1 \cdot m$
Anhalten	$clamp\left(\delta_s\right)$	$clamp\left(\delta_s\right) \cdot m$

5. Evaluation und Ergebnisse

In diesem Kapitel wird evaluiert inwieweit die vorliegende Arbeit ihre Aufgabenstellung erreicht hat. Das Kapitel ist hierfür in zwei Teile unterteilt. Im ersten Teil werden die praktischen Tests vorgestellt, die zur Evaluation herangezogen wurden. Der zweite und letzte Teil untersucht, inwieweit die Aufgabenstellung aus Kapitel 1 und die Anforderungen aus Abschnitt 4.1 von der vorliegenden Arbeit behandelt wurden. Der letzte Aspekt der Aufgabenstellung – der Vergleich mit der Planungskomponente des Team AnnieWAY – wird aufgrund seines Umfangs gesondert betrachtet und in Kapitel 6 behandelt.

5.1. Tests

Dieser Abschnitt stellt zuerst das grundlegende Set-up vor, das für alle Testfälle Gültigkeit besitzt. Darauf werden sechs Testfälle und ihre Ergebnisse aus der Simulation und überwiegend auch aus Testläufen mit dem Versuchsträger vorgestellt.

5.1.1. Set-up für die Testfälle

Das in Abbildung 5.1 dargestellte Straßennetz wurde in allen Testfällen verwendet. Es befindet sich auf dem Gelände der ehemaligen Mackensen-Kaserne, auf dem die Universität Karlsruhe ein asphaltiertes Testgelände besitzt. Das Straßennetz enthält unter anderem zwei im Kreis verlaufende Straßenzüge, die durch eine Kreuzung miteinander verbunden sind. Einmal auf eine Spur platziert, kann das autonome Automobil auf diesem Straßennetz ohne Unterbrechungen oder manuelles Umsetzen der Fahrzeugposition eine Serie von Fahrmanövern absolvieren.

In der Simulation wird der Regelkreislauf für die Lateralregelung durch das Fahrradmodell, in dem das Automobil als einspurig angenommen wird, geschlossen. Das gleiche Modell findet auch in der im Versuchsträger implementierten Regelung Verwendung. Die Longitudinalregelung wird in der Simulation durch einen einfachen PID-Regler durchgeführt. Hierfür besitzt der Versuchsträger einen komplizierteren Regler. Dynamische Objekte werden direkt auf Objektebene, ohne Umweg durch Wahrnehmungsdaten, simuliert und können durch Joysticks

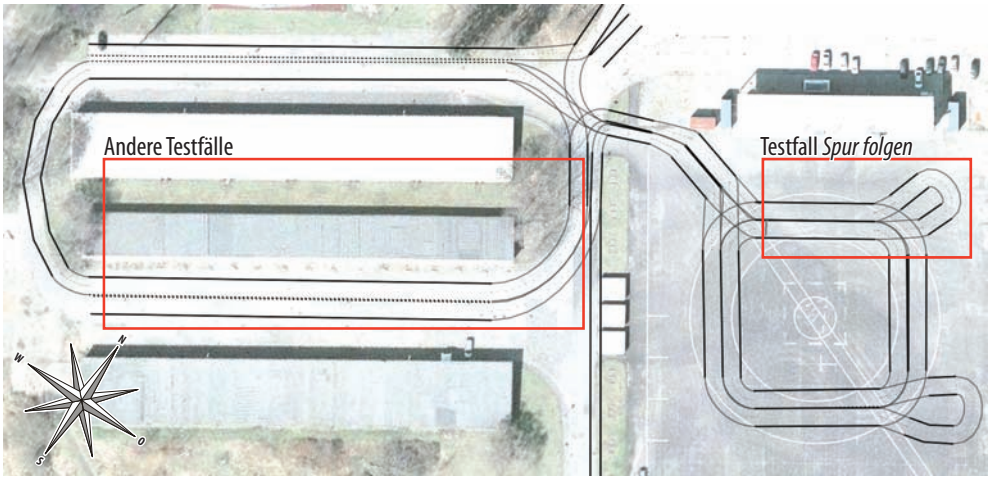


Abbildung 5.1.: Das für alle Testfälle verwendete Straßennetz. Es ist in der Abbildung über ein Luftbild des Testgeländes gelegt.

bewegt werden. Auch sie verwenden das Fahrradmodell, um die Bewegungsmuster simulierter Fahrzeuge denen von echten anzugleichen. Außerdem wird der Sensormessfehler durch Verrauschen der Position und der Geschwindigkeit der Fahrzeuge simuliert. Direkte Wahrnehmungsdaten werden nicht simuliert. (Für einige Details über die verwendeten Modelle und den Regler siehe *Kammel* u. a. 2008).

5.1.2. Testfall *Spur folgen*

Szenario

Dieser Testfall stellt das einfache Fahrmanöver dar, einer vorgegebenen Spur zu folgen. Das Szenario besteht aus der in Abbildung 5.2 dargestellten Situation: Das autonome Automobil muss dem durch die Mission vorgegebenen Straßenverlauf folgen. Dabei fährt es eine enge 180-Grad-Kurve, um seine Fahrtrichtung umzukehren. In diesem Testfall sind noch keine anderen Verkehrsteilnehmer von Bedeutung. Seine Aufgabe ist es, die grundlegende Funktionalität des Verhaltensnetzwerks zu evaluieren.

Ergebnisse

In diesem ersten Test, der den Fokus auf das Spurhalten legt, sind nur die beiden Verhalten *Spur folgen* und *Spur halten* von Bedeutung. Abbildung 5.3 zeigt ihre Motivation, Reflexion und

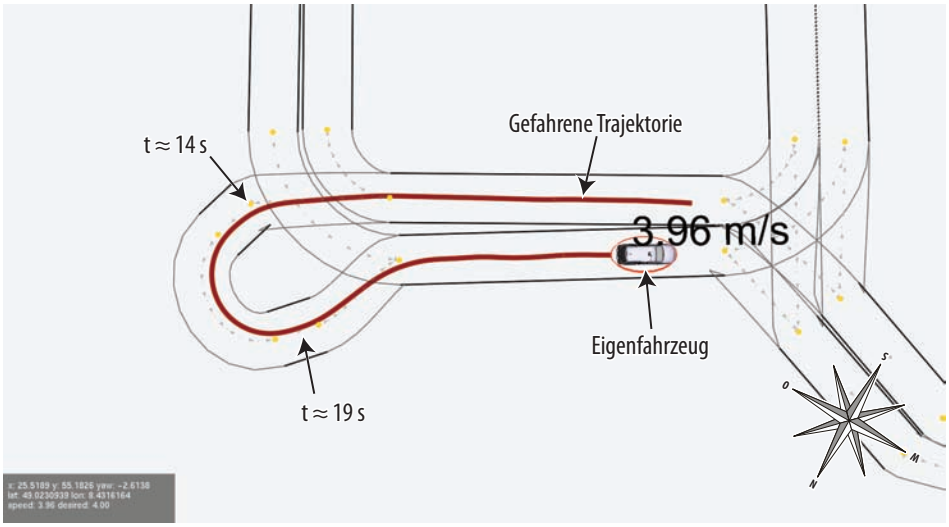
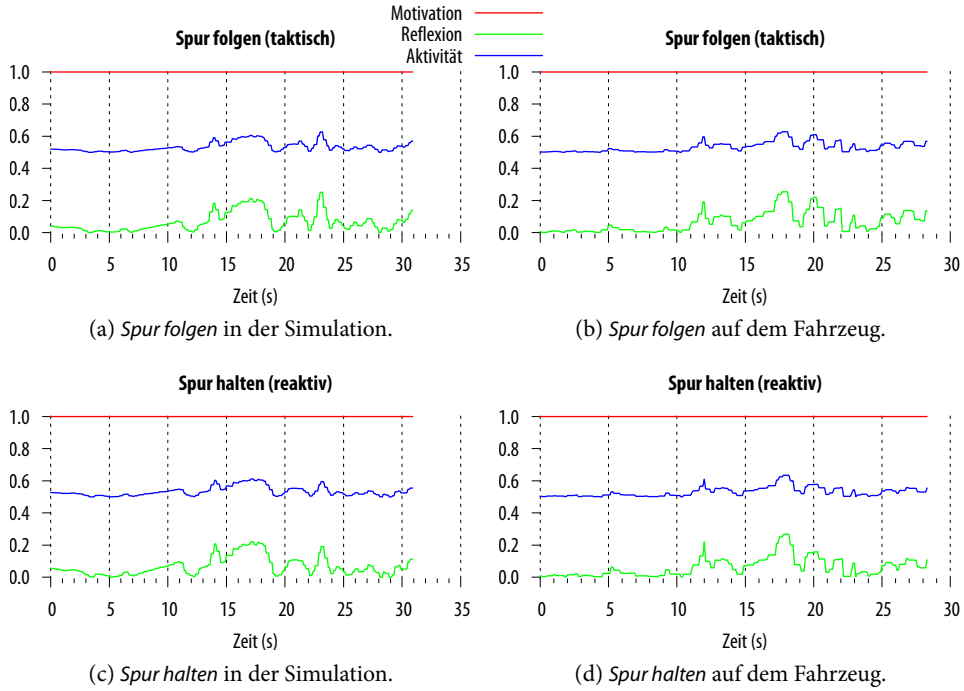


Abbildung 5.2.: Bildschirmfoto des Testfalls *Spur folgen* ausgeführt in der Simulation. Dargestellt ist das Straßennetz, die in diesem Testfall gefahrene Trajektorie und das Eigenfahrzeug.



(a) *Spur folgen* in der Simulation.

(b) *Spur folgen* auf dem Fahrzeug.

(c) *Spur halten* in der Simulation.

(d) *Spur halten* auf dem Fahrzeug.

Abbildung 5.3.: Ergebnisse des Testfalls *Spur folgen*.

Aktivität während des Testlaufs sowohl in der Simulation als auch in einem Testlauf auf dem Versuchsträger. Wie in Abschnitt 2.1.6 erläutert, sind alle drei Werte auf das Intervall $[0, 1]$ skaliert. Dort wurde auch auf die Semantik der Werte näher eingegangen.

Abbildungen 5.3a und 5.3c zeigen die Motivation und die virtuellen Sensorwerte aus der Simulation für die Verhalten *Spur folgen* und *Spur halten*. Abbildungen 5.3b und 5.3d zeigen die Werte für einen Testlauf mit dem Versuchsträger. Die Ergebnisse aus beiden Testläufen unterscheiden sich nicht grundsätzlich. Kleinere Abweichungen treten wegen geringfügig unterschiedlicher Anfangs- und Endpositionen auf. Generell kann man aus den Daten erkennen, dass sowohl in der Simulation wie auch in der Realität auf dem Versuchsträger das Fahrzeug durch das in der Arbeit entwickelte Verhaltensnetzwerk steuerbar ist und dass die Integration der SFB/TR28-Komponenten grundsätzlich funktioniert. In den Reflexionswerten von *Spur halten* und *Spur folgen* kann man erkennen, dass das Eigenfahrzeug einige Male von der Spurmitte abgewichen ist. Beispielsweise erkennt man eine Abweichung im Zeitraum zwischen 14 s und 19 s, weil die Bahnplanung hier eine von der Spurmitte abweichende Trajektorie berechnet hat, um die enge Kurve durchfahren zu können.

5.1.3. Testfall *Überholen eines stehenden Fahrzeugs*

Szenario

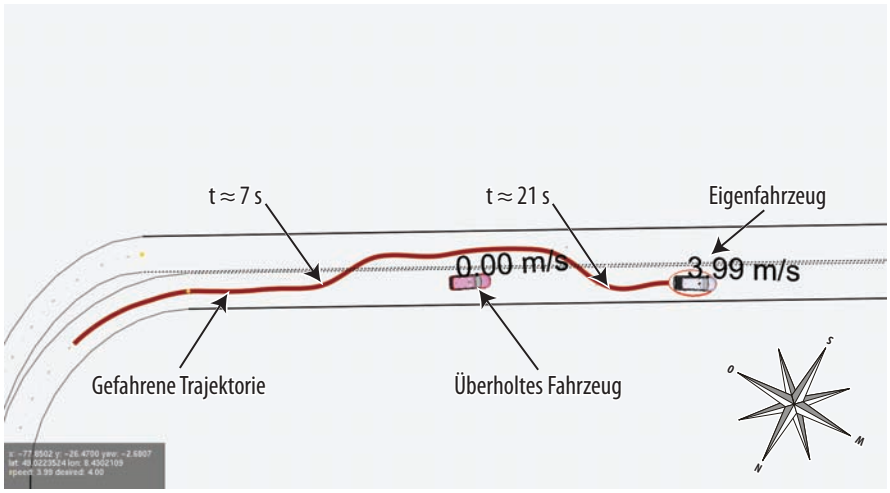


Abbildung 5.4.: Bildschirmfoto des Testfalls *Überholen eines stehenden Fahrzeugs* ausgeführt in der Simulation. Dargestellt ist das Straßennetz, die gefahrene Trajektorie, das Eigenfahrzeug und das überholte Fahrzeug nach der Ausführung des Testfalls.

Das in Abbildung 5.4 dargestellte Szenario dieses Testfalls besteht aus einem Fahrzeug, das auf einer Fahrbahn steht. Es existiert eine Gegenfahrbahn. Die beiden Fahrbahnen sind durch eine Leitlinie miteinander verbunden. Das Eigenfahrzeug darf dementsprechend die Gegenfahrbahn dazu benutzen, um das stehende Fahrzeug zu überholen.

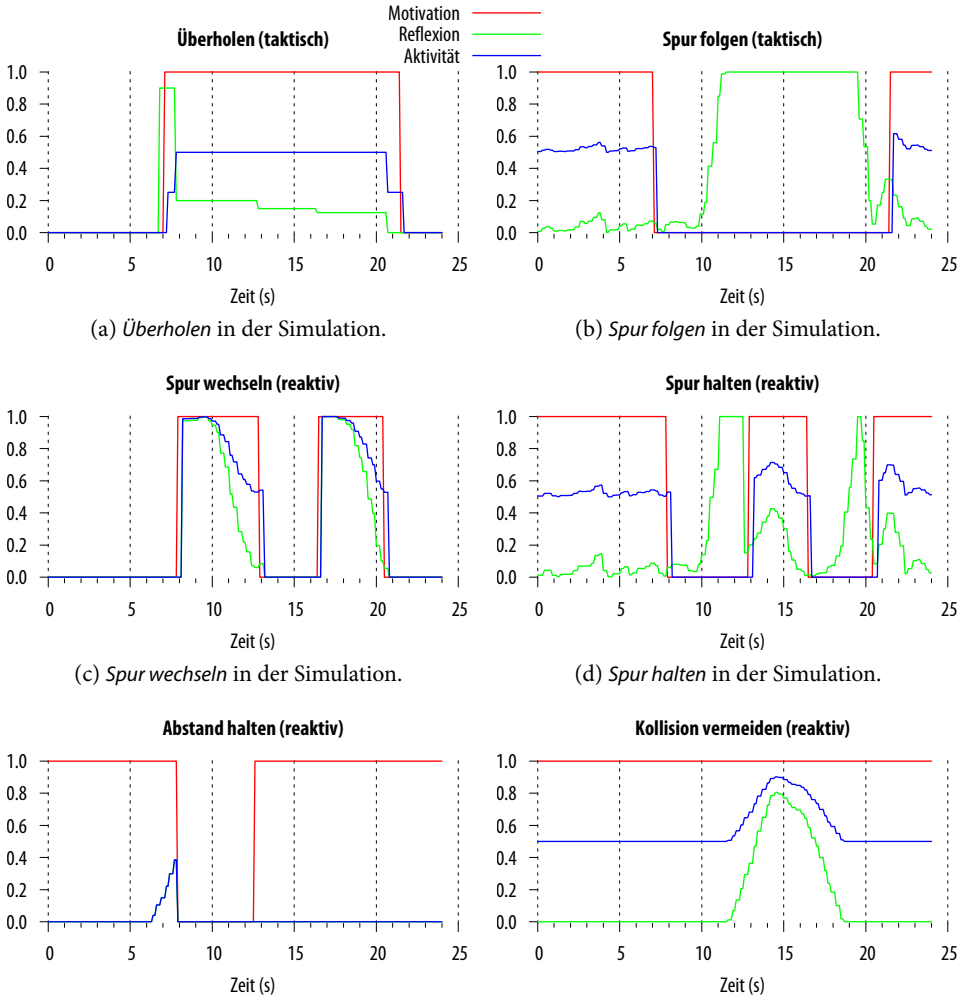
Die Aufgabe dieses Testfalls ist es, zum einen die Integration der für das Erkennen von dynamischen Hindernissen zuständigen Komponenten zu testen. Zum anderen dient er dazu, das *Überholen*-Verhalten in seiner elementarsten Form zu testen.

Ergebnisse

Abbildung 5.5 zeigt die Motivation, Reflexion und Aktivität einer Auswahl von Verhalten bei einer simulierten Ausführung des Testfalls. Abbildung 5.6 zeigt die Werte der gleichen Verhalten bei einer realen Ausführung des Testfalls auf dem Versuchsträger. Ein normaler Überholvorgang verläuft wie folgt: Das taktische Verhalten *Überholen* erkennt ein langsameres Fahrzeug voraus und hat ermittelt, dass es dieses Fahrzeug überholen kann. Den Wunsch zu überholen vermittelt es dem gerade aktiven strategischen Verhalten. Erst wenn das strategische Verhalten *Überholen* motiviert, parametrisiert *Überholen* die reaktiven Verhalten *Spur wechseln* und *Spur halten* um einen zweifachen Spurwechsel durchzuführen. Beim ersten Spurwechsel auf die Gegenfahrbahn wird zusätzlich noch *Abstand halten* demotiviert. Würde *Abstand halten* nicht demotiviert werden, würde es das Spurwechseln behindern, da hier der Sicherheitsabstand zum überholten Fahrzeug nicht unbedingt eingehalten wird. Die Reflexion von *Kollision vermeiden* signalisiert eine steigende Unzufriedenheit mit dem Abstand zum anderen Fahrzeug. An den Reflexionen der Verhalten *Spur wechseln* und *Spur halten* lässt sich die ungefähre Position innerhalb der Spuren ablesen. Die Reflexion des Sicherheitsverhaltens *Kollision vermeiden* lässt erkennen, zu welchem Zeitpunkt die beiden Fahrzeuge sich beim Überholvorgang am nächsten sind. Das taktische *Spur-folgen*-Verhalten, das über den Zeitraum des Überholens von der strategischen Schicht demotiviert wurde, ist sehr Unzufrieden während das Eigenfahrzeug auf der Gegenspur fährt. Würde es in dieser Zeit motiviert sein, würde es durch Parametrisieren von *Spur wechseln* versuchen wieder auf die von der Situationsinterpretation vorgegebene Spur zu wechseln. Deshalb hat das aktive strategische Verhalten das *Spur-folgen*-Verhalten während des kompletten Überholvorgangs demotiviert.

Vergleicht man die simulierte Ausführung des Testfalls mit dem auf dem Versuchsträger ausgeführten Testlauf, so erkennt man – abgesehen von dem geringfügig anderen Zeitablauf – nur einen signifikanten Unterschied: Das Verhalten *Überholen* hat ungefähr bei 17 s Testausführungsdauer einen Anstieg von Reflexion und Aktivität zu verzeichnen. Dieser Anstieg ist – wie in Abschnitt 4.3.4 erläutert – auf einen Abbruch des Überholvorgangs zurückzuführen. Während das andere Fahrzeug überholt wurde, hat die Wahrnehmungskomponente es für einen kurzen Moment nicht mehr erkannt und beim Wiedererkennen dem Fahrzeug eine neue Identifikation gegeben. Der Überholvorgang hat sich, wie man an den Reflexionen der reaktiven Verhalten sehen kann, durch den Abbruch in dieser späten Phase des Vorgangs allerdings nicht verändert.

Der aufmerksame Betrachter wird bemerken, dass die Aktivität bei einigen Verhalten der in Abschnitt 2.1.6 vorgestellten Korrelation widerspricht: Auch bei einer Motivation von null haben die Verhalten für kurze Zeit noch eine Aktivität, die von null abweicht. Dies ist dadurch zu erklären, dass in der aktuellen Implementierung des Frameworks die Reflexion und die Aktivität der Verhalten immer für den letzten Verarbeitungsschritt berechnet werden. Sie geben also die Zufriedenheit und Aktivität des vorherigen Schritts wider und können somit auch intern im Verhalten als Regelgrößen verwendet werden.



(a) *Überholen* in der Simulation.

(b) *Spur folgen* in der Simulation.

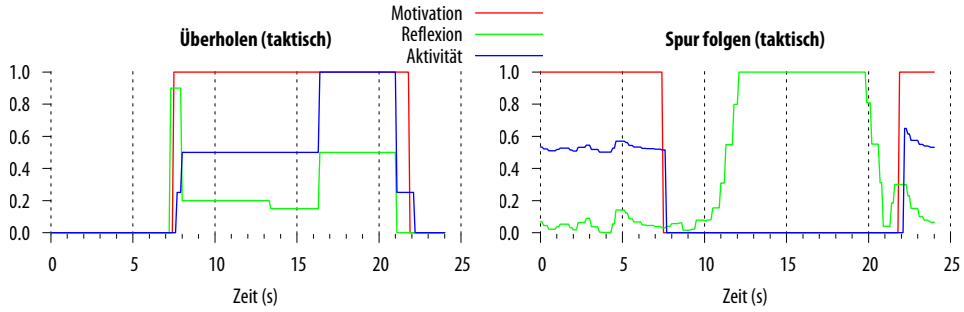
(c) *Spur wechseln* in der Simulation.

(d) *Spur halten* in der Simulation.

(e) *Abstand halten* in der Simulation.

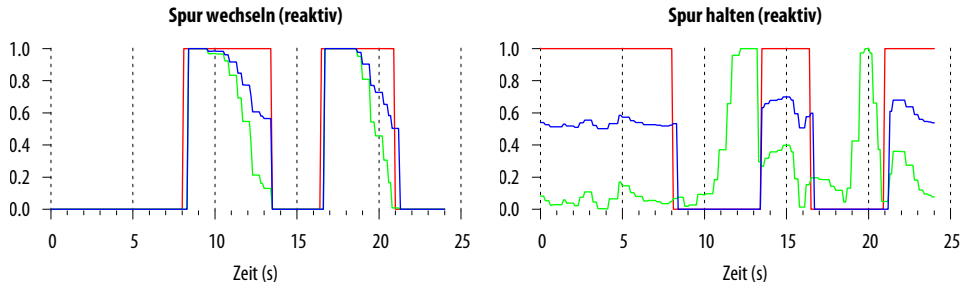
(f) *Kollision vermeiden* in der Simulation.

Abbildung 5.5.: Ergebnisse des Testfalls *Überholen eines stehenden Fahrzeugs* aus der Simulation.



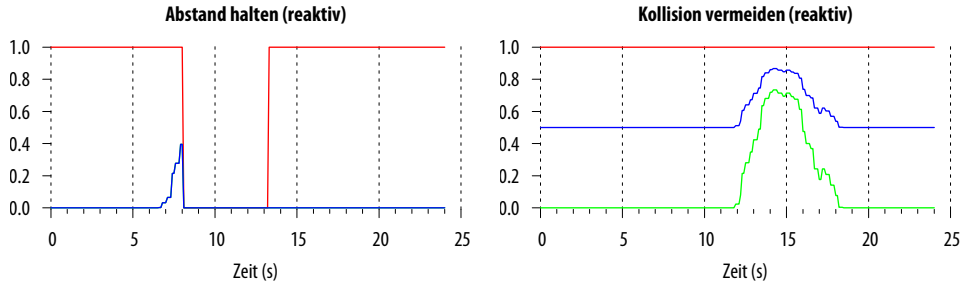
(a) *Überholen* auf dem Fahrzeug.

(b) *Spur folgen* auf dem Fahrzeug.



(c) *Spur halten* auf dem Fahrzeug.

(d) *Spur halten* auf dem Fahrzeug.



(e) *Abstand halten* auf dem Fahrzeug.

(f) *Kollision vermeiden* auf dem Fahrzeug.

Abbildung 5.6.: Ergebnisse des Testfalls *Überholen eines stehenden Fahrzeugs* ausgeführt mit dem Versuchsfahrzeug.

5.1.4. Testfall *Überholen eines fahrenden Fahrzeugs*

Szenario

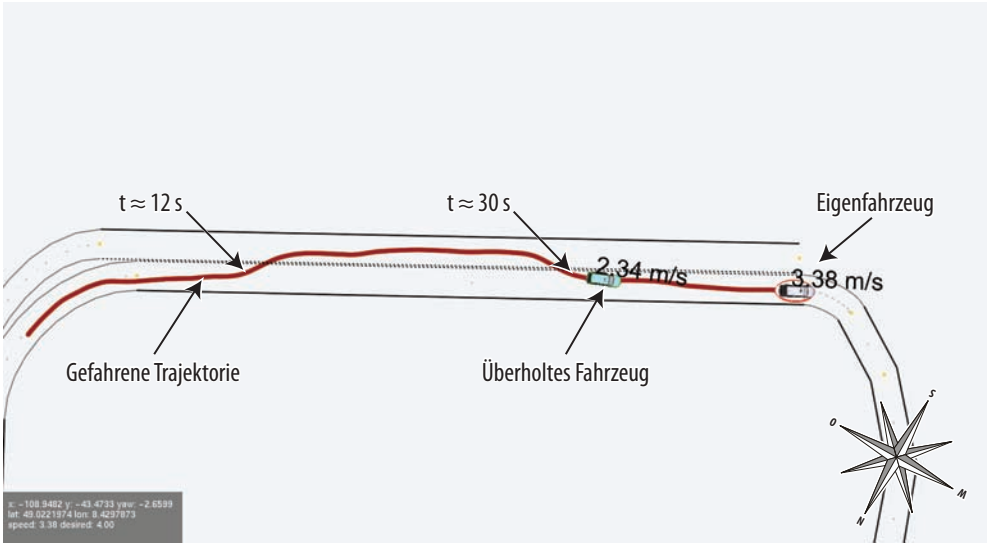


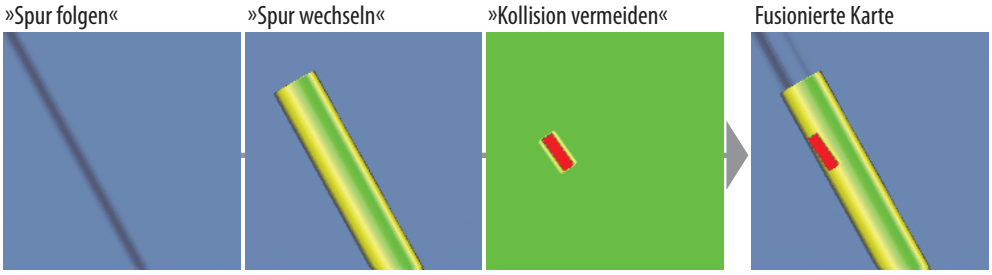
Abbildung 5.7.: Bildschirmfoto des Testfalls *Überholen eines fahrenden Fahrzeugs* ausgeführt auf dem Versuchsträger. Dargestellt ist das Straßennetz, die gefahrene Trajektorie, das Eigenfahrzeug und das überholte Fahrzeug nach der Ausführung des Testfalls.

Das in Abbildung 5.7 dargestellte Szenario dieses Testfalls besteht aus einem langsam fahrenden Fahrzeug, das sich auf einer Spur mit Gegenspur fortbewegt. Beide Spuren sind durch eine Leitlinie getrennt, dementsprechend darf die Gegenfahrbahn benutzen werden, um das langsam fahrende Fahrzeug zu überholen.

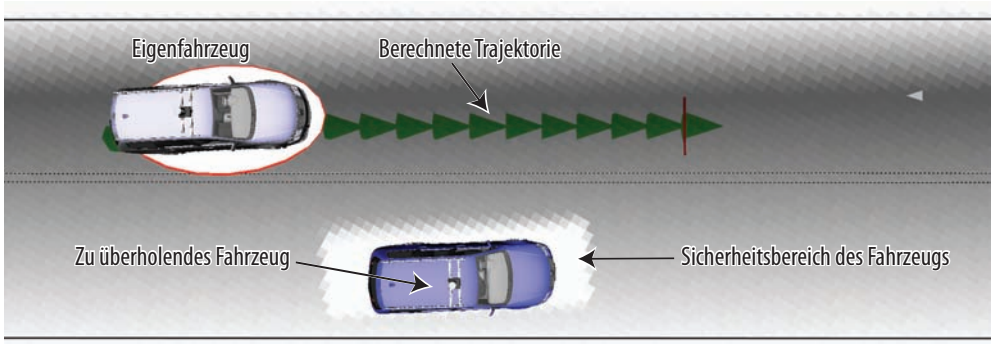
Die Aufgabe dieses Testfalls ist es zu testen, ob auch dynamischere Situationen durch das Verhaltensnetzwerk, die Situationsinterpretation, die Bahnplanung und die zwischen ihnen entwickelten Schnittstellen bewältigt werden können. Zum Anderen dient er dazu, das *Überholen*-Verhalten in einer schwierigeren Situation zu testen.

Ergebnisse

Die Motivationen, Reflexionen und Aktivitäten der Verhalten unterscheiden sich in diesem Testfall nicht wesentlich von denen im vorherigen Testfall *Überholen eines stehenden Fahrzeugs*.



(a) Die einzelnen Risikokarten und die aus ihnen erstellte, fusionierte Risikokarte. Dargestellt sind Falschfarbenabbildungen: Grün bedeutet sehr geringes Risiko, gelb mittleres, braun hohes, aber noch befahrbares Risiko. Blau bedeutet so hohes Risiko, dass hier kein Weg geplant wird. Rot bedeutet nochmals höheres Risiko: würde hier ein Pfad geplant, würde es zu einem Zusammenstoß kommen.



(b) Bildschirmfoto der Visualisierung zum Zeitpunkt, zu dem die in (a) dargestellten Risikokarten aufgenommen wurden.

Abbildung 5.8.: Zusammensetzung der fusionierten Risikokarte gezeigt am Beispiel des Spurwechsels auf die Gegenspur ungefähr zum Zeitpunkt 16 s.

Sie sind in Abbildung 5.9 und 5.10 dargestellt. Die folgenden, kleineren Unterschiede sind zu erkennen:

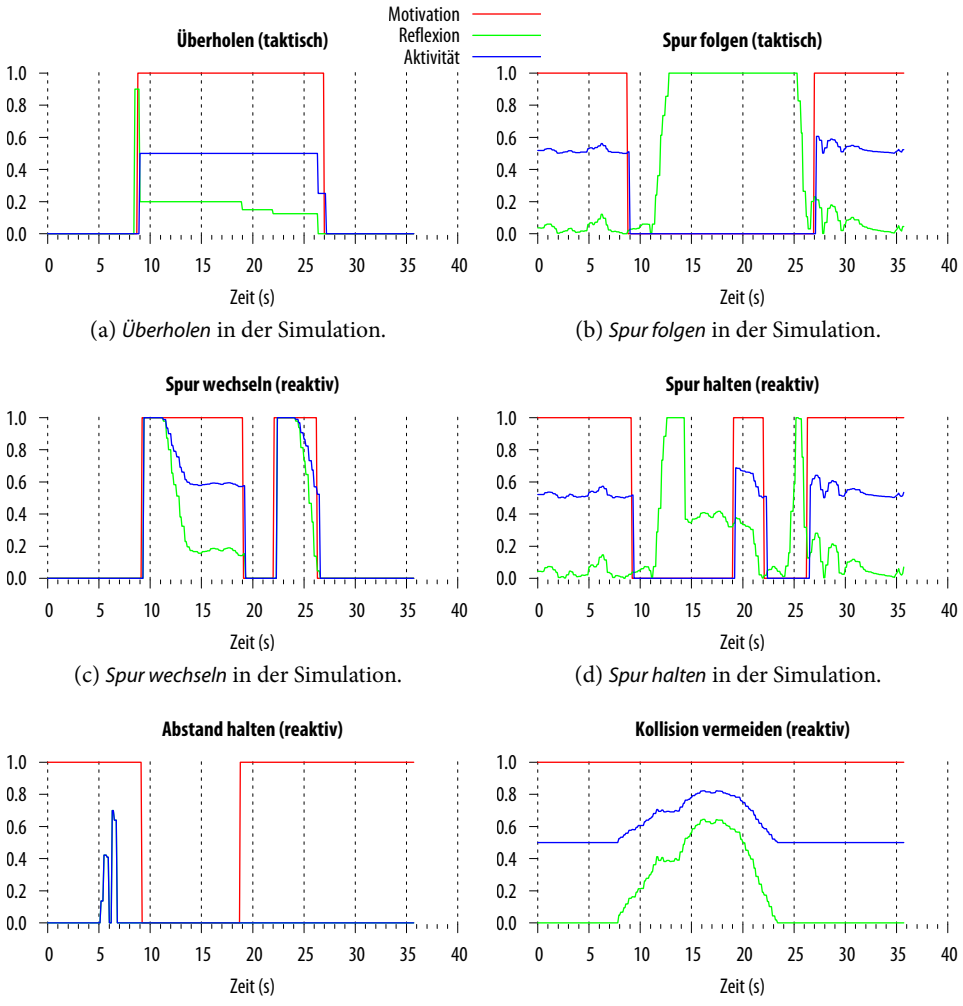
- Die Zeit, die sich das Eigenfahrzeug auf der Gegenspur befindet ist natürlich länger. Dies erkennt man z.B. aus einem Vergleich der Reflexionen des *Spur-folgen*-Verhaltens.

- Das *Abstand-halten*-Verhalten ist in diesem Testfall vor dem Überholvorgang unzufriedener. Der Abfall der Reflexion vor dem Überholvorgang lässt erkennen, dass *Abstand halten* das Eigenfahrzeug schon an die Geschwindigkeit des vorausfahrenden Fahrzeugs angepasst hat und erst dann der Wunsch zu überholen aufkam.

- Der Spurwechselvorgang auf die Gegenfahrbahn hat in diesem speziellen Testlauf der Simulation etwas länger gedauert. Die Bahnplanung hat in diesem Fall eine Trajektorie erzeugt, die das Eigenfahrzeug sanfter auf die Gegenfahrbahn lenkt.

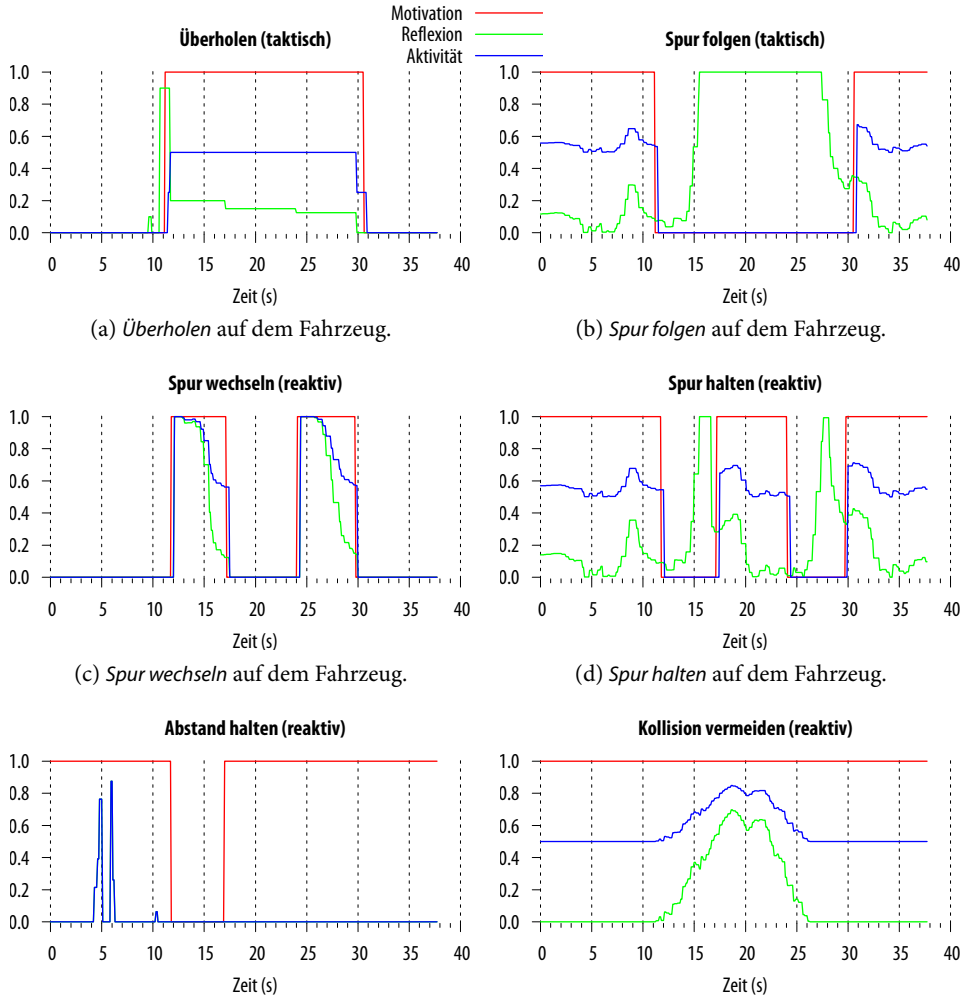
- Beim Testlauf auf dem Versuchsträger hat das *Überholen*-Verhalten den Überholvorgang nicht aufgrund von falschen Wahrnehmungsdaten abgebrochen. Die momentan implementierte Wahrnehmungskomponente erkennt fahrende Fahrzeuge robuster als stehende und so ist es in diesem Testfall unwahrscheinlicher, dass eine solche Fehlmessung wie im Testlauf des *Überholen-eines-stehenden-Fahrzeugs*-Testfalls auftritt.

Abbildung 5.8 zeigt exemplarisch für einen Zeitpunkt, wie sich die Risikokarte zusammensetzt, die der Bahnplanung als Pfadplanungsgrundlage dient.



(a) *Überholen* in der Simulation.
(b) *Spur folgen* in der Simulation.
(c) *Spur wechseln* in der Simulation.
(d) *Spur halten* in der Simulation.
(e) *Abstand halten* in der Simulation.
(f) *Kollision vermeiden* in der Simulation.

Abbildung 5.9.: Ergebnisse des Testfalls *Überholen eines fahrenden Fahrzeugs* aus der Simulation.



(a) *Überholen* auf dem Fahrzeug.

(b) *Spur folgen* auf dem Fahrzeug.

(c) *Spur wechseln* auf dem Fahrzeug.

(d) *Spur halten* auf dem Fahrzeug.

(e) *Abstand halten* auf dem Fahrzeug.

(f) *Kollision vermeiden* auf dem Fahrzeug.

Abbildung 5.10.: Ergebnisse des Testfalls *Überholen eines fahrenden Fahrzeugs* ausgeführt mit dem Versuchsfahrzeug.

5.1.5. Testfall *Folgefahren*

Szenario

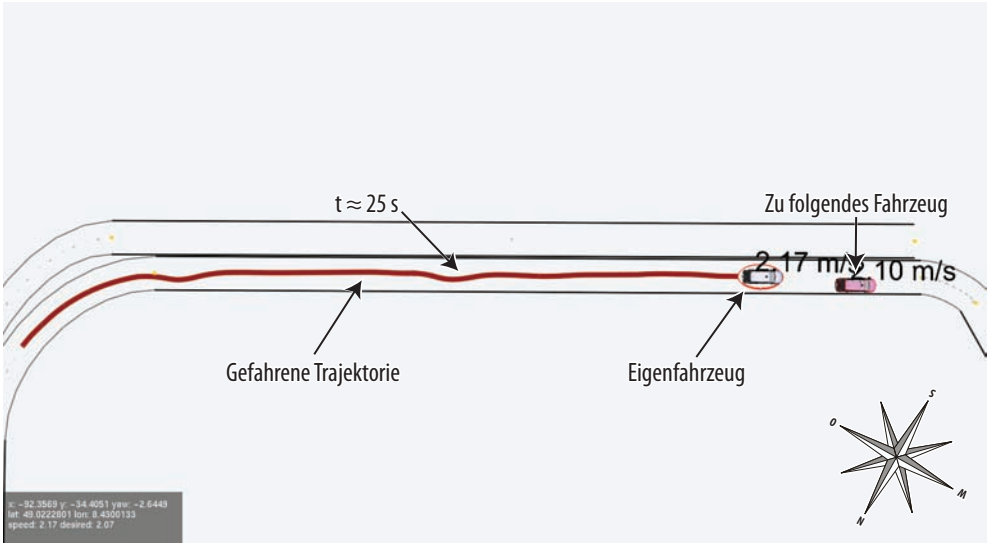


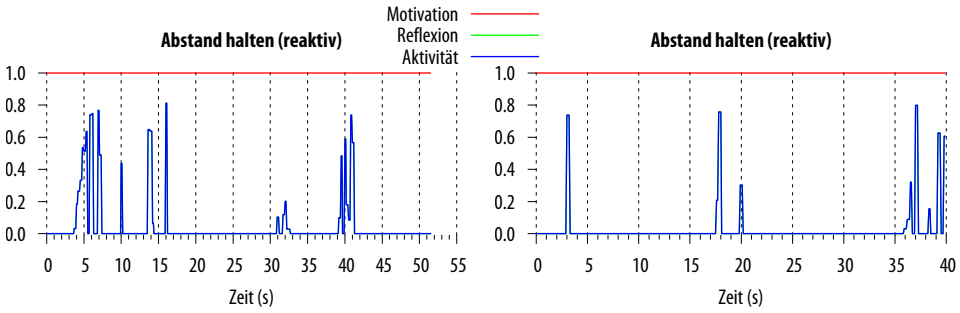
Abbildung 5.11.: Bildschirmfoto des Testfalls *Folgefahren* ausgeführt in der Simulation. Dargestellt sind das Straßennetz, die gefahrene Trajektorie, das Eigenfahrzeug und das Fahrzeug, dem gefolgt wird.

Das in Abbildung 5.11 dargestellte Szenario für diesen Testfall besteht aus einem langsam fahrenden Fahrzeug, das sich aufgrund der Straßenverhältnisse nicht überholen lässt. Das Fahrzeug führt mehrere Abbrems- und Beschleunigungsvorgänge aus. Ungefähr zur Hälfte des Testfalls bleibt das Fahrzeug ganz stehen und verweilt so einige Sekunden.

Die Aufgabe dieses Testfalls ist es zum einen, die Schnittstelle zur Bahnplanung und die Bahnplanung selbst zu testen. Für korrektes Folgefahren muss das Erstellen von prädizierten Risikokarten und die korrekte Wahl dieser Karten funktionieren. Zum anderen soll mit diesem Testfall das reaktive Verhalten *Abstand halten* getestet werden.

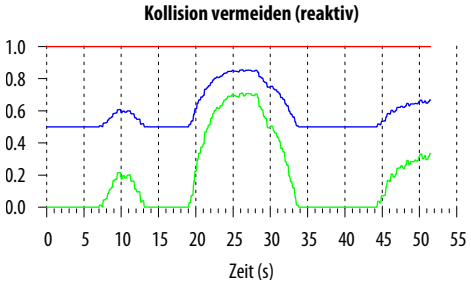
Ergebnisse

Die Ergebnisse aus der Simulation und aus einem Testlauf auf dem Versuchsträger, die in Abbildung 5.12 dargestellt sind, lassen sich leider nicht so direkt vergleichen wie bei den anderen Testfällen. Ein Grund ist, dass der Testfall sehr schwierig genau gleich wiederholt werden kann. Dazu ist die notwendige Koordination mit dem Fahrer des vorausfahrenden Fahrzeugs zu auf-

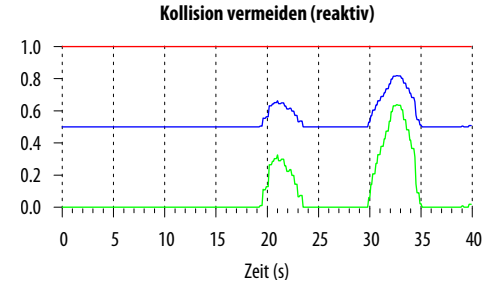


(a) *Abstand halten* in der Simulation.

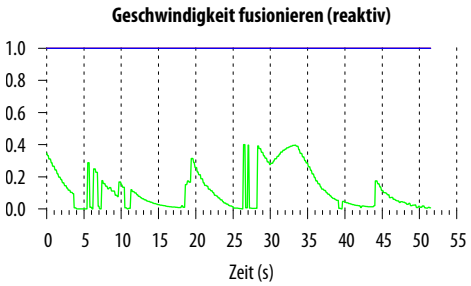
(b) *Abstand halten* auf dem Fahrzeug.



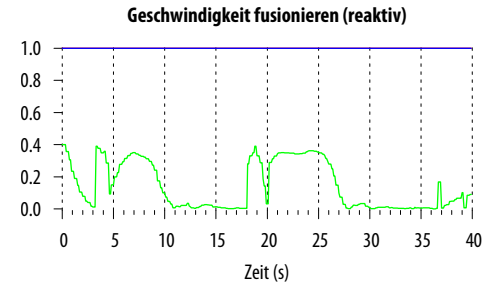
(c) *Kollision vermeiden* in der Simulation.



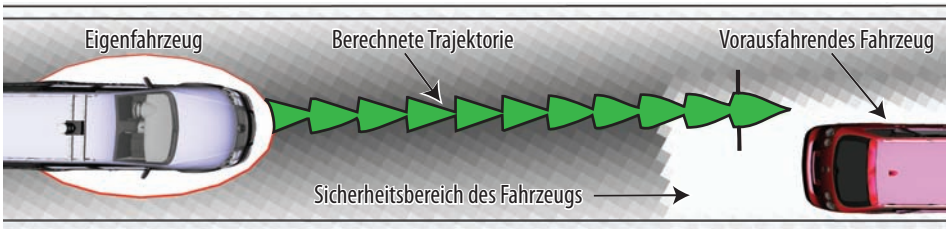
(d) *Kollision vermeiden* auf dem Fahrzeug.



(e) *Geschwindigkeit fusionieren* in der Simulation.



(f) *Geschwindigkeit fusionieren* auf dem Fahrzeug.



(g) Detailaufnahme aus der Visualisierung. Dargestellt ist das Eigenfahrzeug, das vorausfahrende Fahrzeug und die von der Bahnplanung ermittelte Trajektorie. Die aktuelle Risikokarte ist der Szene hinterlegt. Schwarz bedeutet geringes, weiß bedeutet hohes Risiko.

Abbildung 5.12.: Ergebnisse des Testfalls *Folgefahren*

wendig. Ein anderer Grund, warum sich die Ergebnisse nicht direkt vergleichen lassen, ist in der unterschiedlichen Longitudinalregelung der Simulation und des Versuchsfahrzeugs zu finden. Die Reflexion des *Geschwindigkeit-fusionieren*-Verhaltens verdeutlicht diesen Unterschied. In Abbildungen 5.12e und 5.12f sieht man die unterschiedlich ausgebildeten Steigungsformen der Regler in den Zufriedenheitswerten des *Geschwindigkeit-fusionieren*-Fusionsknotens.

Abbildung 5.12g zeigt eine Detailaufnahme des Simulationslaufs zum Zeitpunkt 40 s. Durch diese Abbildung wird ersichtlich, dass die Bahnplanung Trajektorien berechnet, die durch das vorausfahrende Fahrzeug führen würden, wenn es selbst keine Eigengeschwindigkeit hätte. Dies kann nur der Fall sein, wenn die Prädiktion der Bewegung des Fahrzeugs und das Eintragen in unterschiedliche Risikokarten im *Abstand-halten*-Verhalten funktioniert und wenn die Bahnplanung diese unterschiedliche Risikokarten richtig auswertet.

Die Einknicke, die in der Trajektorie in Abbildung 5.11 an den beiden Stellen zu sehen sind, an denen das vorausfahrenden Fahrzeug stand, sind zu erklären durch das Abbruchkriterium, das die Bahnplanung in diesem Planungsmodus verwendet: Es wird ein Pfad gesucht, der eine bestimmte Zeit in die Zukunft reicht. Da der direkte, optimale Pfad durch das stehende Fahrzeug blockiert wird, sucht die Bahnplanung einen möglichst langen, noch befahrbaren Pfad und findet einen mit hohem Risiko aber noch befahrbaren Pfad, der an den Rand der Spur führt.

5.1.6. Testfall *Ausfall der taktischen und strategischen Schicht*

Szenario

In diesem Testfall werden die taktische und strategische Schicht des Verhaltensnetzwerks komplett deaktiviert. Das reaktive *Spur-halten*-Verhalten ist motiviert und alle Sicherheitsverhalten der reaktiven Schicht sind ebenfalls motiviert. Sonst ist das Szenario das gleiche, wie im Testfall *Überholen eines stehenden Fahrzeugs*.

Aufgabe dieses Testfalls ist es, die Robustheit des Verhaltensnetzwerks gegenüber Fehlern zu demonstrieren und zu zeigen, dass selbst in einem solchen Fehlerfall das Fahrzeug noch sicher geführt werden kann.

Ergebnisse

Dieser Testfall wurde der Einfachheit halber nur in der Simulation ausgeführt. Einer Ausführung auf dem Versuchsträger steht nichts im Wege, sie würde aber der Aufgabe dieses Testfalls

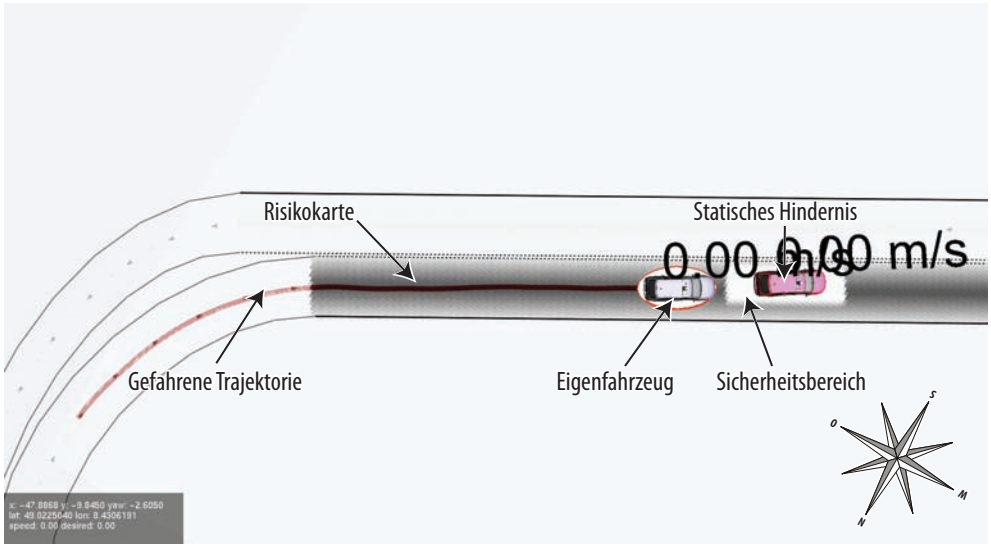
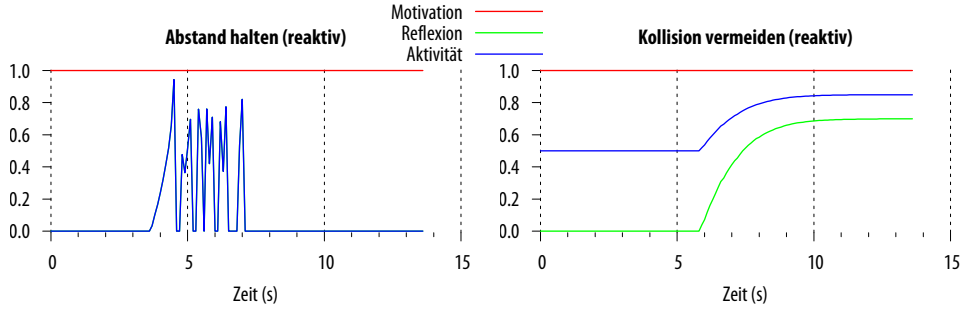
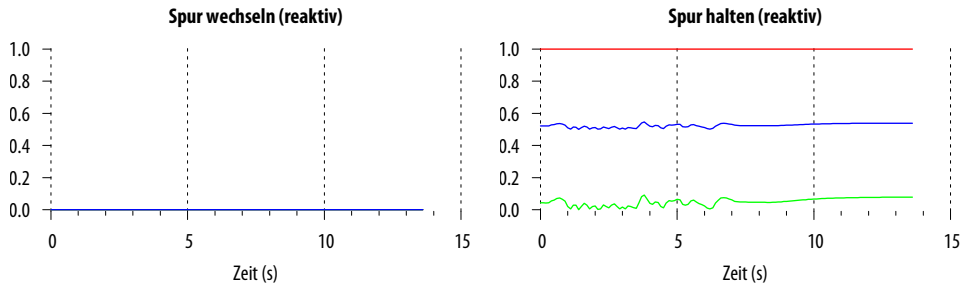


Abbildung 5.13.: Szenario des Testfalls *Ausfall der taktischen und strategischen Schicht*.



(a) *Abstand halten* in der Simulation.

(b) *Kollision vermeiden* in der Simulation.



(c) *Spur wechseln* in der Simulation.

(d) *Spur halten* in der Simulation.

Abbildung 5.14.: Ergebnisse des Testfalls *Ausfall der taktischen und strategischen Schicht*.

nicht von Nutzen sein. Die Simulation zeigt, dass auch bei einem solchen fehlerhaften System das autonome Automobil noch ein sicheres Verhalten hat. Da das *Spur-wechseln*-Verhalten nicht von einer taktischen Schicht aktiviert werden kann, hält das Eigenfahrzeug vor dem stehenden Fahrzeug in sicherer Distanz an. Dies wird, wie in Abbildung 5.14 zu sehen, durch das *Abstand-halten*-Verhalten erreicht. Das Verhalten *Kollision vermeiden* ist wegen dem nahen Hindernis unzufrieden. Die von diesem Verhalten generierte Risikokarte ist in diesem Testfall aber nicht vonnöten, um einen Zusammenstoß zu vermeiden. Abbildung 5.14 zeigt auch die Motivationen von den Verhalten *Spur wechseln* und *Spur halten*. Es ist zu sehen, dass *Spur wechseln* in diesem Testfall nicht motiviert wurde und dementsprechend nur *Spur halten* motiviert war.

5.1.7. Testfall *Ausfall eines reaktiven Verhaltens*

Szenario

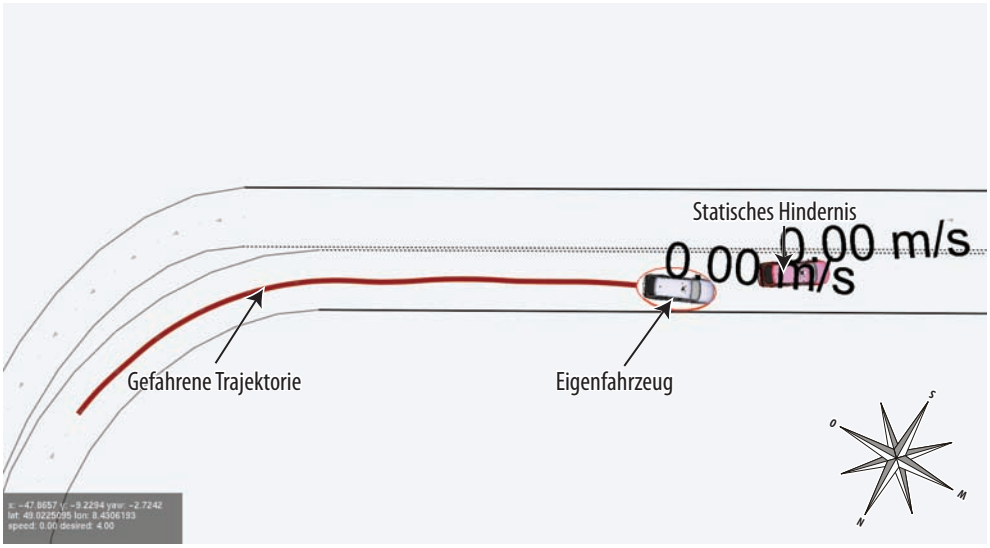
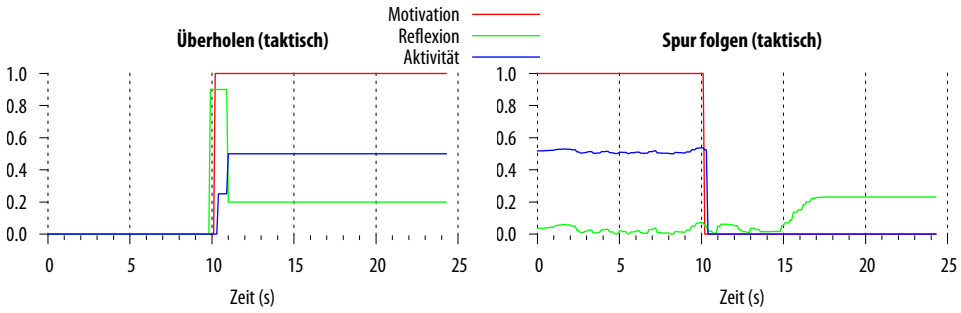


Abbildung 5.15.: Szenario des Testfalls *Ausfall eines reaktiven Verhaltens*.

Auch dieses Szenario verwendet der Einfachheit halber das Szenario des Testfalls *Überholen eines stehenden Fahrzeugs*. In diesem Testfall arbeiten die taktische und strategische Schicht wie gewohnt, allerdings ist das *Spur-wechseln*-Verhalten so modifiziert worden, dass es keine Ausgabe mehr erzeugt. Das taktische Verhalten *Überholen* kann so zwar das *Spur-wechseln*-Verhalten motivieren, allerdings ohne die gewünschte Wirkung.

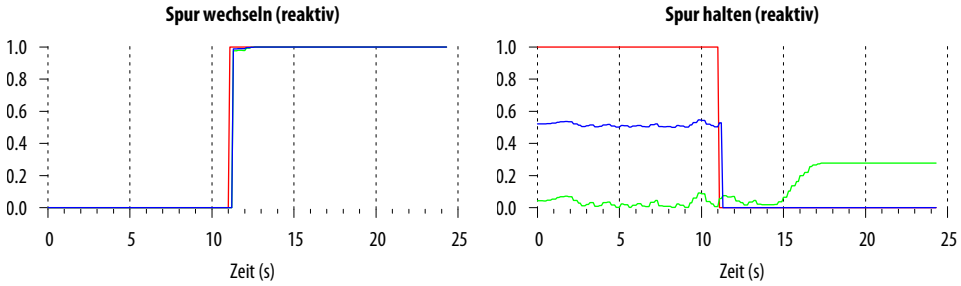
Dieser Testfall hat die Aufgabe zu zeigen, dass auch bei einer teilweisen fehlerhaften reaktiven Schicht noch ein sicheres Verhalten vom Verhaltensnetzwerk generiert werden kann.

Ergebnisse



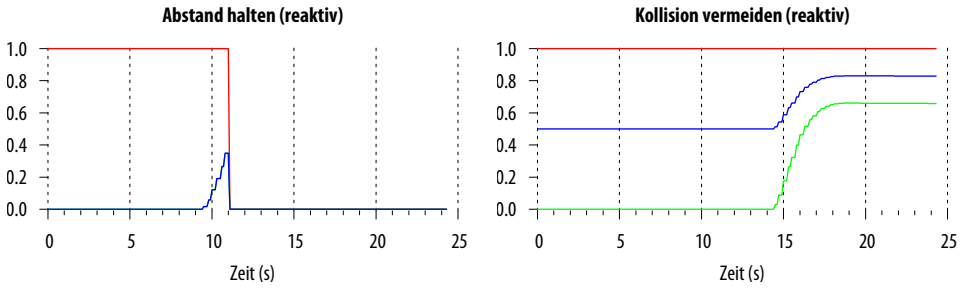
(a) *Überholen* in der Simulation.

(b) *Spur folgen* in der Simulation.



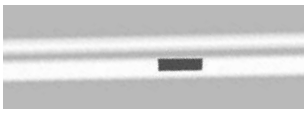
(c) *Spur wechseln* in der Simulation.

(d) *Spur halten* in der Simulation.



(e) *Abstand halten* in der Simulation.

(f) *Kollision vermeiden* in der Simulation.



(g) Fusionierte Risikokarte am Ende des Testlaufs. Zur besseren Darstellung weicht die Farbkodierung vom Standard ab: schwarz bedeutet sehr hohes, weiß mittleres Risiko.

Abbildung 5.16.: Ergebnisse des Testfalls *Ausfall eines reaktiven Verhaltens*.

Der Testfall wurde nur in der Simulation ausgeführt, Abbildung 5.16 zeigt deshalb nur die Motivation, Reflexion und Aktivität einiger ausgewählter Verhalten aus einem Simulationslauf.

Für den Überholvorgang motiviert das taktische Verhalten *Überholen* das reaktive Verhalten *Spur wechseln* und demotiviert die reaktiven Verhalten *Spur halten* und *Abstand halten*. Da aber das *Spur-wechseln*-Verhalten keine Risikokarte erzeugt, wird auch die Spur nicht gewechselt und da das *Abstand-halten*-Verhalten demotiviert wurde, kann dieses Verhalten nicht – wie im vorherigen Testfall – das Eigenfahrzeug zum kontrollierten Stillstand bringen. In dieser Situation tritt das reaktive Sicherheitsverhalten *Kollision vermeiden* in Aktion, denn die von diesem Verhalten erzeugten Risikokarten verhindern, dass die beiden Fahrzeuge zusammenstoßen.

Abbildung 5.16g zeigt die fusionierte Risikokarte, wie sie sich am Ende des Testlaufs darstellt. Die Position der Fahrzeuge ist also wie in Abbildung 5.15 abgebildet. Die weniger hohen Risikowerte werden durch das Sicherheitsverhalten *Auf Straße bleiben* erzeugt. Das Rechteck mit hohem Risiko wird durch das *Kollision-vermeiden*-Verhalten erzeugt. Die Risikokarte des *Auf-Straße-bleiben*-Verhaltens verhindert ein Ausscheren aus der Spur, die Risikokarte des *Kollision-vermeiden*-Verhaltens verhindert das Zusammenstoßen. Anzumerken ist hier, dass bei der vom Standard abweichenden Farbkodierung in Abbildung 5.16g nur die weißen Bereiche des Bildes befahrbar sind. Graue und schwarze Bereiche sind nicht befahrbarere Bereiche. Diese Farbkodierung wurde hier gewählt, da im Druck sonst kein Unterschied zwischen den Risikowerten ersichtlich wäre.

In diesem Testfall wurde das *Spur-wechseln*-Verhalten bewusst als fehlerhaft ausgewählt, da es in dieser Situation das Verhalten ist, das am meisten zum gewünschten Fahrverhalten beiträgt. Der gleichzeitige Ausfall mehrerer reaktiver Verhalten oder der Ausfall eines Fusionsknotens oder Sicherheitsverhaltens kann in der aktuellen Netzwerkstruktur nicht kompensiert werden.

5.2. Evaluation

In diesem Abschnitt werden die Aufgabenstellungen aus Kapitel 1, die das Verhaltensnetzwerk oder das Framework zur Erstellung von Verhaltensnetzwerken betreffen, nochmals aufgegriffen, um hier zu zeigen, inwieweit die vorliegende Arbeit diese Aufgaben erfüllt hat. Der Vergleich mit der Planungskomponente des Team AnnieWAY, der auch Teil der Aufgabenstellung war, findet sich im Kapitel 6. Nach den Aufgabenstellungen aus Kapitel 1 wird in diesem Abschnitt noch auf die in Abschnitt 4.1 erarbeiteten Anforderungen eingegangen.

5.2.1. Aufgabenstellung

> »Das Framework sollte die im SFB/TR28 entwickelte Softwarearchitektur nutzen und die vorherrschende Hardwareplattform unterstützen. Speziell ist hier die gute Anbindung an die im SFB/TR28 entwickelte Echtzeitdatenbank zu nennen.«

Durch die Implementierung eines Verhaltensnetzwerks mit dem erarbeiteten Framework und die Durchführung von Tests auf dem Versuchsträger mit dieser Implementierung konnte die Integration in die Software- und Hardwarearchitektur demonstriert werden.

> »Das Testen der Funktionstüchtigkeit einzelner Verhalten soll leichter als bisher möglich sein. Insbesondere sollen auch automatisierte Tests unterstützt werden.«

In Abschnitt 4.2.8 wurde eine spezielle Testumgebung vorgestellt, die automatisierte Tests von einzelnen Verhalten ermöglicht. Eine experimentelle Implementierung dieser Testumgebung existiert. Sie zeigt die Machbarkeit des Entwurfs. Aus Zeitgründen wurden aber über diese experimentelle Implementierung hinaus keine weiteren, richtigen automatisierte Testfälle geschrieben.

> »Das Verhaltensnetzwerk soll auf eine im SFB/TR28 noch zu entwickelnde einheitliche Simulationskomponente vorbereitet werden. Zur Unterstützung dieser Simulationskomponente muss das Framework die Möglichkeit bieten, die Ausführung unterschiedlicher Teile des Verhaltensnetzwerks durch externe Ereignisse auslösen zu lassen. Außerdem sollte das Framework auch eine Simulation unterstützen, die nicht in Echtzeit ausgeführt wird. Somit können auch aufwendigere Algorithmen auf weniger leistungsfähiger Hardware simuliert werden.«

Aufgrund dieser Aufgabenstellung wurde die Ausführung der Schichten in eine spezielle Komponente, den *Signalprozessor* (siehe Abschnitt 4.2.5), verlagert. Er stößt die Ausführung der unterschiedlichen Verhaltensschichten aufgrund von Ereignissen der *Signalemitter* (siehe Abschnitt 4.2.4) und einer einfachen Vorschrift an. Während der Simulation kann anstatt des Taktsignals eines Zeitgebers die externe Simulationskomponente das Taktsignal liefern. Eine Ausführung langsamer als in Echtzeit kann so leicht realisiert werden. Spezielle Ereignisse, die direkt einzelne Schichten des Verhaltensnetzwerks anstoßen, sind durch diese Architektur leicht zu realisieren.

> »Das Framework soll eine explizite Dokumentation der Netzwerkstruktur überflüssig machen, indem es die implizit im Quelltext des Verhaltensnetzwerks definierten Abhängigkeiten der Verhalten für eine automatische Dokumentation der Netzwerkstruktur ausnutzt.«

In einem speziellen Modus kann das Verhaltensnetzwerk die aktuelle Netzwerkstruktur in eine XML-Datei ausgeben. Diese XML-Datei kann mithilfe einer Visualisierung (siehe Abschnitt B.3) dargestellt werden und in PDF-Dateien exportiert werden.

> »Die enge Koppelung von Bahnplanung/Trajektorienplanung und der Verhaltensentscheidung soll gelöst werden. Dazu muss eine Schnittstelle zu einer neu entwickelten Bahnplanungskomponente geschaffen werden und die reaktive Schicht des Verhaltensnetzwerks muss für diese Schnittstelle überarbeitet werden.«

Beide Unteraufgaben wurden erfolgreich durchgeführt. Die in diesem Kapitel vorgestellten Tests belegen dies. Die Verhalten der reaktiven Schicht erstellen sog. *Risikokarten* (siehe Abschnitt 4.3.1), die fusioniert werden und der Bahnplanung als Grundlage für die Pfadsuche dienen. Aufgrund spezieller SIMD[1]-Optimierungen ist es dem Verhaltensnetzwerk möglich, zu jedem Ausführungsschritt der reaktiven Schicht 40 Ausgangsrisikokarten zu generieren und diese Ausgangskarten zu 10 Risikokarten zu fusionieren. Diese 10 Risikokarten werden über die Echtzeitdatenbank an die Bahnplanung übermittelt. Die Bahnplanung besitzt mit ihrer Hilfe auch Wissen über die Zukunft und kann so auch in dynamischen Umgebungen valide Pfade planen.

> »Die Verhalten sollten, sofern es sinnvoll ist, die von der Situationsinterpretation aufbereiteten Modelle und nicht mehr direkt die Daten der Wahrnehmungskomponente verwenden. Insbesondere gilt dies für die Repräsentation der Fahrbahnen; die Situationsinterpretation bietet für sie ein fusioniertes Modell.«

Diese Aufgabenstellung wurde umgesetzt, in dieser Ausarbeitung aber nicht weiter dokumentiert, weil auf konkrete Implementierungsdetails des Verhaltensnetzwerks hier aus Platzgründen nicht eingegangen werden konnte.

5.2.2. Anforderungen

In Abschnitt 4.1 wurden 8 Anforderungen an das Verhaltensnetzwerk gestellt, deren Einhaltung hier näher betrachtet werden soll.

Anforderung 1: Unterstützung vielfältiger Ziele

Das Konzept des biologisch motivierten Verhaltensnetzwerks sieht durch seine Aufteilung in Schichten mit aufsteigender Reaktivität und die in diesen Schichten unabhängig voneinander operierenden Verhaltensbausteine schon eine Unterstützung vielfältiger Ziele vor. Das in der Anforderung erwähnte Ziel des Rechts-abbiegens an einer Kreuzung wird z.B. durch einen Verhaltensbaustein in der strategischen Schicht koordiniert. Die Einhaltung der Höchstgeschwindigkeit und das Halten eines Sicherheitsabstands werden jeweils von Verhaltensbausteinen der reaktiven Schicht durchgesetzt. Mehrere Ziele werden also gleichzeitig verfolgt. Die Schwierigkeit besteht allerdings darin, diese zum Teil gegenläufigen Fahrintentionen zu einer einzigen ausgeführten Handlung zu fusionieren. Wie Abschnitt 4.3.1 zeigt, kann eine Fusion auf zu stark abstrahierten Daten in eine Fahrhandlung münden, die keiner der beteiligten Verhaltensbausteine so geplant hat. Das neu eingeführte Konzept der *Risikokarte* (siehe Abschnitt 4.3.1) unterstützt diese Anforderung. Eine Fusion, die alle Fahrintentionen unverfälscht erhält, ist durch dieses Konzept effizient möglich, wie die Tests in diesem Kapitel und ihre Ausführung unter realen Bedingungen auf dem Versuchsträger gezeigt haben. Für dynamischere Fahrmanöver,

[1] Engl. Abkürzung für *Single Instruction Multiple Data*. Ein Befehl wird parallel auf vielen Daten ausgeführt.

in denen es entscheidend ist, dass sich das Eigenfahrzeug zu einem bestimmten Zeitpunkt an einer bestimmten Position befindet, ist die aktuelle Geschwindigkeitsfusion noch nicht ausgelegt. Momentan wird sie durch eine Minimumfusion über die gewünschten Geschwindigkeiten aller beteiligter Verhalten erreicht. Es sind aber Situationen denkbar, in denen es notwendig ist, nicht die minimale Geschwindigkeit, sondern eine höhere Geschwindigkeit zu fahren, um eine gefährliche Situation vermeiden zu können. Beispielsweise ist das aktive Ausweichen auf den Standstreifen einer Autobahn um einen Auffahrunfall zu vermeiden eine solche Situation. Das *Abstand-halten*-Verhalten würde in diesem Fall Geschwindigkeit null fahren wollen. Ein hypothetisches aktives Sicherheitsausweichen-Verhalten verlangt aber eine höhere Geschwindigkeit, bis der Standstreifen erreicht ist.

Anforderung 2: Unterstützung vielfältiger Sensoren

Wie in Abschnitt 4.1 erläutert, ist diese Brooks'sche Anforderung für das Verhaltensnetzwerk nicht von Bedeutung, da höhere und niedrigere Schichten der SFB/TR28-Architektur diese Anforderung durch Fusion der Sensordaten durchsetzen.

Anforderung 3: Daten-Fehlertoleranz

Die Anforderung der Daten-Fehlertoleranz wurde in einigen Verhalten individuell gelöst. Fehlertoleranz gegenüber falschen Wahrnehmungsdaten wurde nur dort eingefügt, wo bei Tests ersichtlich war, dass dies notwendig ist. So wurde z.B. im *Kollision-vermeiden*-Verhalten Fehlertoleranz gegenüber der Ausrichtung der dynamischen Objekte in Bezug auf die Spur, auf der sie sich befinden, hinzugefügt. Tests haben gezeigt, dass besonders die Ausrichtung von stehenden Objekten von der aktuell implementierten Wahrnehmungskomponente nicht zuverlässig genug erkannt wird. Mithilfe der an einigen Stellen hinzugefügten Daten-Fehlertoleranz war es möglich, die in diesem Kapitel beschriebenen Testfälle auch auf dem Versuchsträger unter realen Bedingungen auszuführen. Das Framework bietet keine Unterstützung in Hinblick auf die Daten-Fehlertoleranz.

Anforderung 4: Rechen-Fehlertoleranz

In den Testfällen *Ausfall der taktischen und strategischen Schicht* und *Ausfall eines reaktiven Verhaltens* wurde die Rechen-Fehlertoleranz an Beispielen gezeigt. Wenn, wie im vorliegenden Fall, das Verhaltensnetzwerk mit einer gewissen Redundanz entwickelt wird, kann ein robustes Netzwerk erstellt werden. Einzelne Verhalten oder Schichten höherer Reaktivität können in einem solchen Netzwerk ausfallen oder fehlerhaft sein und das Verhaltensnetzwerk als ganzes kann noch ein sicheres Führen des Automobils gewährleisten. Sollten mehrere reaktive Verhalten oder ein Fusionsknoten der reaktiven Schicht ausfallen, kann dieser Ansatz keinen sicheren Betrieb mehr gewährleisten. In einem solchen Fall, oder wenn das ganze Verhaltensnetzwerk

ausfallen sollte, wird als letzte Notfallschicht in der SFB/TR28-Architektur das autonome Automobil durch eine Vollbremsung zum Stehen gebracht.

Anforderung 5: Anpassungsvermögen

Spezielle Testfälle, die das Anpassungsvermögen des in dieser Arbeit entwickelten Verhaltensnetzwerks belegten, wurden aus Zeitgründen nicht durchgeführt. Die Anpassungsfähigkeit an sich ändernde Umwelteinflüsse war aber eine Entwurfsgrundlage des Verhaltensnetzwerks, wie sich z.B. am Testlauf des Testfalls *Überholen eines stehenden Hindernisses* auf dem Versuchsträger zeigen lässt. In diesem Testlauf wurden, durch fehlerhafte Wahrnehmungsdaten ausgelöst, der Überholvorgang abgebrochen. Hier hat sich das Verhaltensnetzwerks also an eine neue (und in diesem Fall fehlerhaft erkannte) Umwelt angepasst.

Anforderung 6: Erweiterbarkeit

Ob die Anforderung der Erweiterbarkeit zur Zufriedenheit erfüllt wurde, kann nur eine aufbauende Arbeit bewerten. Da die Abhängigkeiten der Verhalten untereinander und zu den Datenabstraktionen explizit angegeben werden und diese Abhängigkeiten auf das Nötigste beschränkt wurden, sollten Erweiterungen der Netzwerkstruktur nur lokale Änderungen des Netzwerks benötigen.

Anforderung 7: Testbarkeit

Die in Abschnitt 4.2.8 vorgestellte Testumgebung ermöglicht es, automatisierte Tests einzelner Verhalten auszuführen. Eine experimentelle Implementierung zeigt die Machbarkeit dieses Entwurfs. Das Framework unterstützt die Anforderung der Testbarkeit in dem in Abschnitt 4.2.8 erläuterten Umfang. Automatisierte Tests die das komplette Netzwerk als solches Testen sind mit diesem Entwurf nicht möglich.

Anforderung 8: Simulationsunterstützung

Aufgrund dieser Anforderung wurde die Ausführung der Schichten in eine spezielle Komponente, den *Signalprozessor* (siehe Abschnitt 4.2.5), verlagert. Er stösst die Ausführung der unterschiedlichen Verhaltensschichten aufgrund von Ereignissen der *Signalemitter* (siehe Abschnitt 4.2.4) und einer einfachen Vorschrift an. Während der Simulation kann anstatt des Taktsignals eines Zeitgebers die externe Simulationskomponente das Taktsignal liefern. Eine Ausführung langsamer als in Echtzeit kann so leicht realisiert werden. Spezielle Ereignisse, die direkt einzelne Schichten des Verhaltensnetzwerks anstoßen, sind durch diese Architektur leicht zu realisieren.

6. Vergleich mit der Planungskomponente des Team AnnieWAY

Dieses Kapitel stellt den hierarchischen Zustandsautomaten vor, der beim Team AnnieWAY während der DARPA Urban Challenge 2007 als Planungskomponente zum Einsatz kam. Eine ausführlichere, englische Darstellung findet sich in *Gindele* u. a. (2008). Nach dieser Einführung wird der hierarchische Automat mit der im Rest der Arbeit vorgestellten Architektur in einigen Punkten verglichen.

6.1. Eingliederung und Weltmodellierung

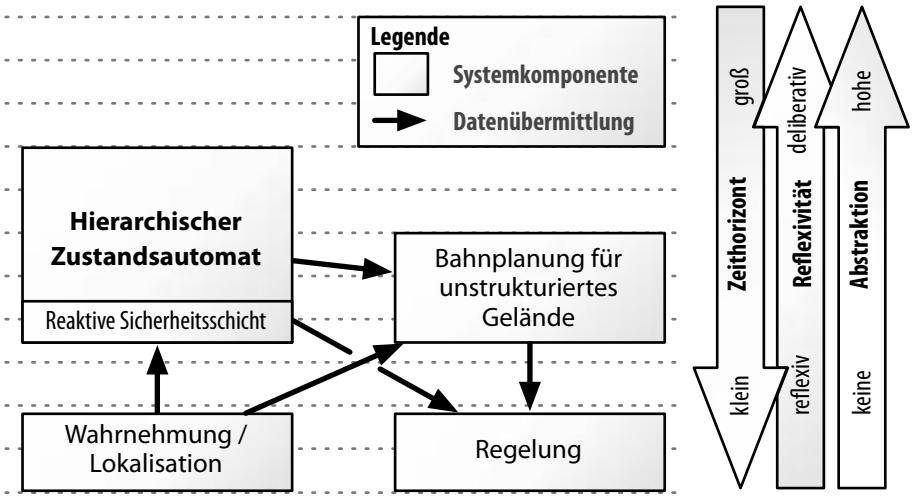


Abbildung 6.1.: Systemarchitektur des Team AnnieWAY.

Der im Team AnnieWAY für die DARPA Urban Challenge 2007 entwickelte hierarchische Zustandsautomat ersetzt die Situationsinterpretation, die Verhaltensentscheidung und die Bahnplanung in der in Kapitel 3 vorgestellten Architektur. Wie in Abbildung 6.1 zu sehen, erhält der Automat direkt die Ergebnisse der Wahrnehmungskomponenten und liefert der Regelung die

zu fahrende Trajektorie. Eng gekoppelt mit dem Zustandsautomaten ist eine reaktive Sicherheitsschicht, die bei drohenden Kollisionen die vom Zustandsautomaten generierte Trajektorie abändern kann.

Zusätzlich zu dem Zustandsautomaten existiert noch eine zweite planerische Komponente, die speziell für unstrukturiertes Gelände entwickelt wurde und temporär vom Zustandsautomaten die Kontrolle erhält. Nähres zu dieser speziellen Bahnplanungskomponente und allgemein zu der Systemarchitektur des Team AnnieWAY findet der interessierte Leser in (*Ziegler* u. a. 2008; *Schröder* u. a. 2008; *Kammel* u. a. 2008).

Der Automat verwendet ein internes Weltmodell um seine Aktionen zu planen. Dieses Modell wird durch die Wahrnehmungsdaten ständig aktualisiert und erweitert. Ein Hauptteil des Weltmodells ist eine virtuelle Karte, die Informationen über das gesamte zu befahrende Straßennetz enthält. Hinzu kommt das Wissen über dynamische und statische Objekte die sich im Umfeld befinden, wie andere Verkehrsteilnehmer oder Straßenabsperrungen.

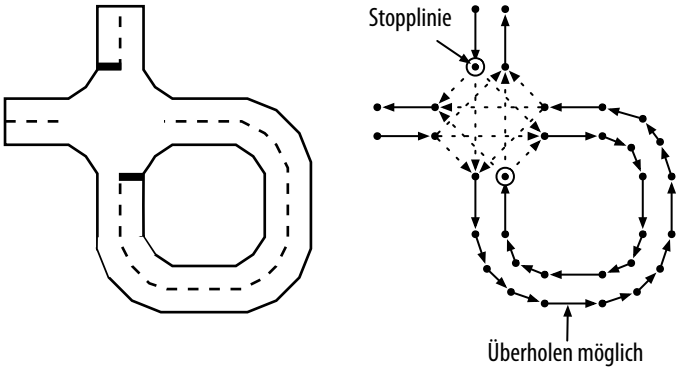


Abbildung 6.2.: Modellierung des Straßenverlaufs als gerichteter Graph, in dem jede Spur getrennt modelliert ist. Knoten und Kanten des Graphen können Attribute erhalten.

Die Karte wird als gerichteter Graph repräsentiert. Dessen Kanten stellen Fahrspurabschnitte dar. Abbildung 6.2 zeigt einen Vergleich zwischen einem exemplarischen Straßennetz und der internen Graphdarstellung dieses Beispiels. Die Knoten stellen Punkte in der realen Welt dar, die über globale Koordinaten verknüpft sind. Sie stellen die Anfangs- und Endpunkte der Fahrspurabschnitte dar und liegen in der Fahrspurmitte. Die Kanten verfügen über weitere Attribute welche z.B. die Länge, Breite oder Art der Fahrbahnmarkierungen definieren. Knoten können ebenfalls weitere Attribute besitzen um z.B. zu kennzeichnen, dass an ihnen angehalten werden muss.

6.2. Aufbau des Automaten

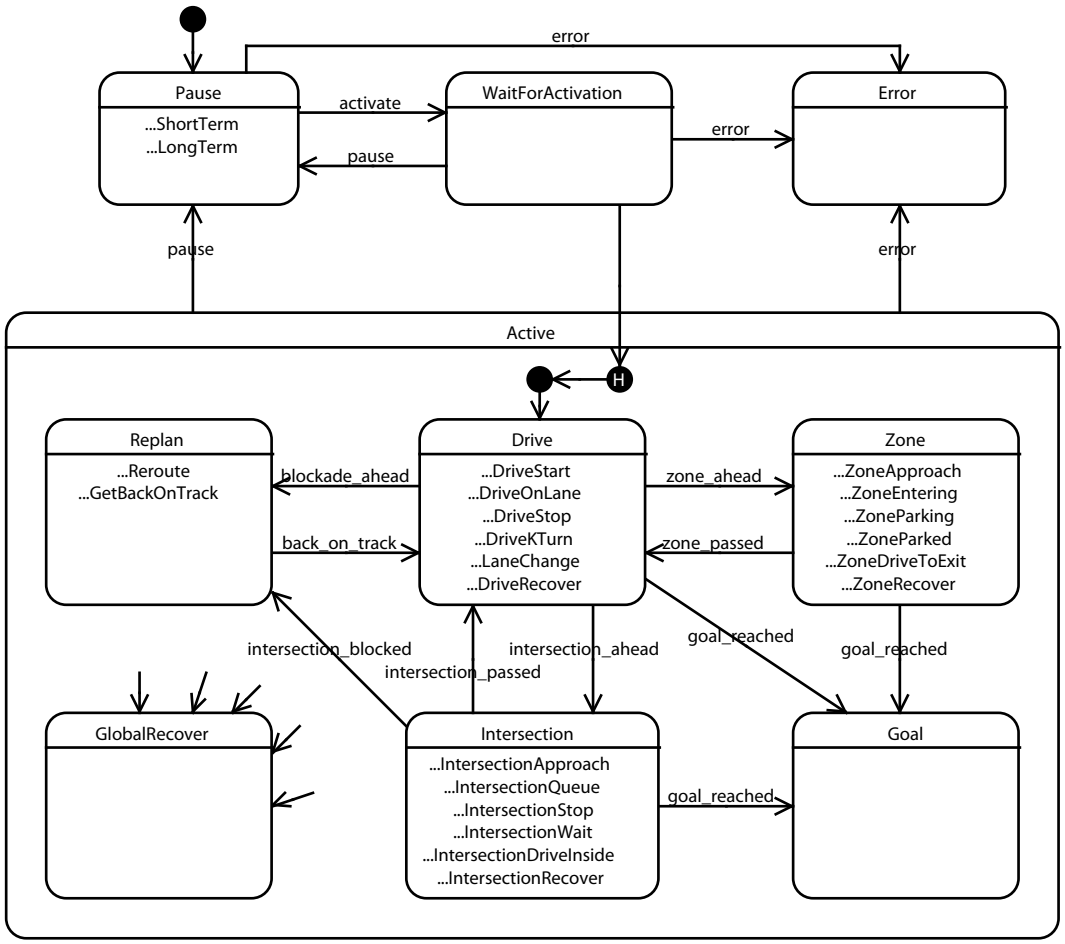


Abbildung 6.3.: UML-Diagramm des Zustandsautomaten. Einige Zustandsübergänge sind aus Gründen der Übersichtlichkeit nur angedeutet. (aus *Gindele* u. a. 2008)

Die Grundidee bei einem hierarchischen Zustandsautomat ist es – ähnlich dem Vererbungsprinzip der objektorientierten Programmierung – die Zustände so zu entwerfen und anzuordnen, dass jeder Unterzustand eine Spezialisierung des jeweiligen Oberzustandes darstellt. In den Unterzuständen werden nur noch die jeweiligen Unterschiede modelliert. Dadurch wird die funktionale Redundanz innerhalb der Zustände und die Anzahl der zu spezifizierenden Zustandsübergänge verringert. Dies erleichtert das Entwerfen und Erweitern von komplexeren Zustandsautomaten.

Im Zustandsautomaten des Team AnnieWAY repräsentiert mit Ausnahme einiger Sonderzustände jeder Zustand ein Verhalten. Der Zustandsautomat wird mit einer festen Taktrate betrieben. Dieser feste Takt ermöglicht es, auf einfache Weise Aussagen über das Laufzeitverhalten

zu machen und somit Echtzeitbedingungen zu garantieren. Da ein Zustandsautomat ereignisgesteuert arbeitet, besitzt der Automat ein spezielles Ereignis, das mit einer festen Taktrate erzeugt wird. Jeder Zustand ist so definiert, dass er auf dieses Ereignis reagiert. Innerhalb der jeweiligen Reaktionsroutine werden dann die für das Verhalten relevanten Aspekte der aktuellen Situation analysiert und bei Bedarf wird in einen neuen Zustand übergegangen.

Wie aus Abbildung 6.3 ersichtlich, hat der Automat vier Hauptzustände: Er kann aktiv (Zustand *Active*) sein, pausiert (Zustand *Pause*) sein, sich in einem Fehlerzustand (Zustand *Error*) befinden, oder auf seine Aktivierung warten (Zustand *WaitForActivation*).

Der Automat startet pausiert. Wenn er das Aktivierungsereignis erhält, wechselt der Zustandsautomat in den *WaitForActivation*-Zustand, aus dem er nach fünf Sekunden automatisch in den *Active*-Zustand wechselt.[1] Wenn der Automat sich im *Active*-Zustand befindet und das Pausierungsereignis empfangen wird, wechselt der Automat sofort in den *Pause*-Zustand. Wenn in einem dieser Zustände ein nicht behebbarer Fehler auftritt, wird in den *Error*-Zustand gewechselt.

Befindet sich der Automat in einem anderen Zustand außer dem *Active*-Zustand, wird das autonome Automobil von ihm angehalten. Die überwiegende Zeit verbringt der Zustandsautomat in dem *Active*-Zustand; dieser Zustand und seine Unterzustände sind für das Verhalten des autonomen Fahrzeugs verantwortlich. Deshalb beschränkt sich die folgende detailliertere Beschreibung des Aufbaus auf diesen Zustand.

6.2.1. Fahren (Zustand *Drive*)

Dieser Zustand ist der Standardzustand, in dem sich der Automat befindet, wenn das Fahrzeug auf einer normalen Straße abseits von Kreuzungen fährt. Seine in Abbildung 6.4 dargestellten Unterzustände ermöglichen es dem autonomen Fahrzeug, die folgenden Manöver durchzuführen:

- Fahren auf einer freien Fahrspur.
- Verfolgen eines anderen Fahrzeugs.
- Wechseln der Spur, inklusive das temporäre Wechseln auf die Gegenfahrbahn, um ein stehendes Fahrzeug zu überholen.
- Anhalten an einer Stopplinie, die nicht Teil einer Kreuzung ist.
- Umdrehen und Zurückfahren am Ende einer Sackgasse.

[1] Diese fünf Sekunden Verzögerung zwischen Aktivierungssignal und tatsächlicher Aktivierung waren eine Vorgabe der DARPA Urban Challenge 2007.

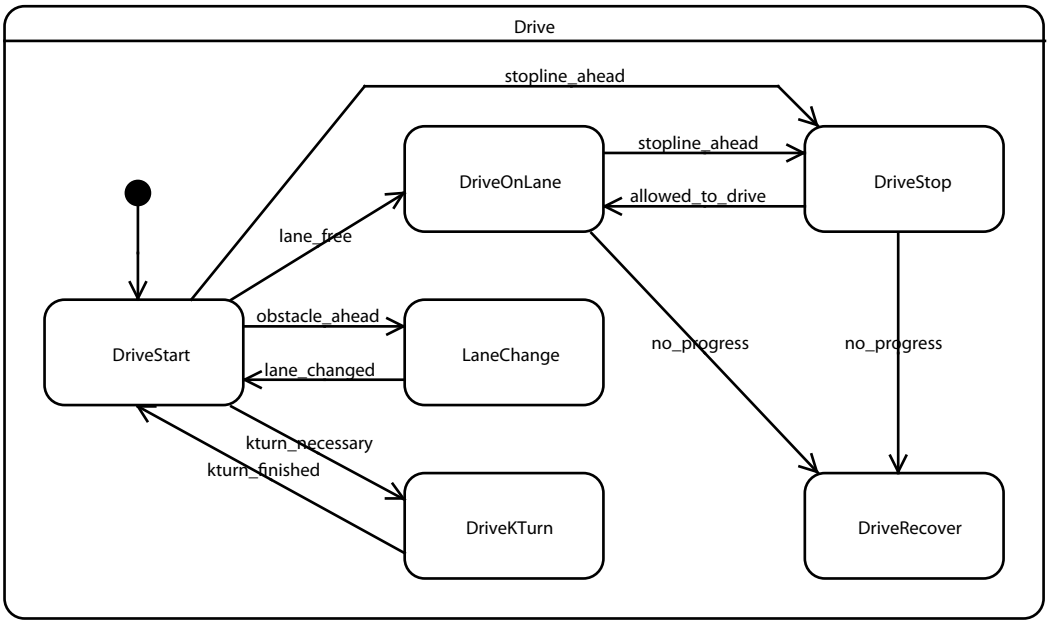


Abbildung 6.4.: Der *Drive*-Zustand und seine Unterzustände (aus *Gindele* u. a. 2008).

Diese Manöver sind alle so gestaltet, dass sie die kalifornische[1] Straßenverkehrsordnung einhalten. Sollte das autonome Fahrzeug allerdings keinen Fortschritt mehr erzielen, besitzen dieser Zustand und seine Unterzustände mehrere Strategien, die nicht unbedingt die Verkehrsregeln einhalten, um das Fahrzeug wieder in eine Situation zu bringen, in der es wieder Fortschritt erzielen kann.

6.2.2. Kreuzung queren (Zustand *Intersection*)

Nähert sich das Auto einer Kreuzung, wechselt es in den *Intersection*[2]-Zustand. Dieser Zustand und seine Unterzustände (siehe Abbildung 6.5) können die folgenden Manöver behandeln:

- An die Kreuzung fahren, mit oder ohne vorausfahrenden Fahrzeugen.
- Anhalten an der Stopplinie, falls erforderlich.
- Wenn nötig, warten, bis das Eigenfahrzeug die Vorfahrt hat.
- Die Kreuzung überqueren.

[1] Die UC07 fand in Kalifornien, USA statt, deshalb galten für sie die kalifornischen Verkehrsregeln.
[2] Engl. Kreuzung

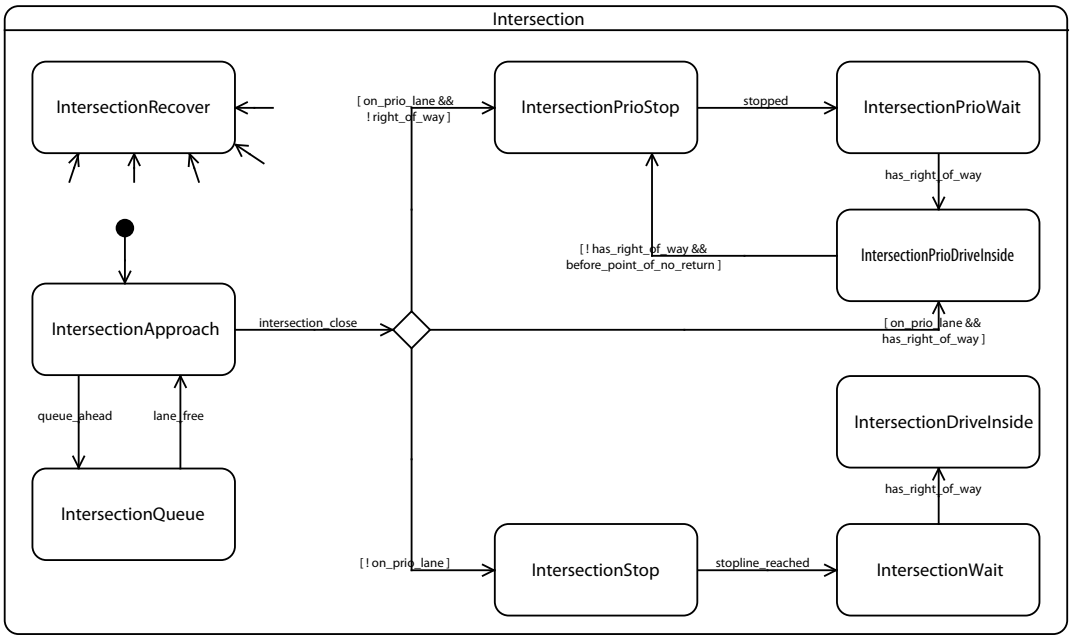


Abbildung 6.5.: *Intersection*-Zustand und seine Unterzustände (aus *Gindele* u. a. 2008).

Um den *Intersection*-Zustand robust zu gestalten, besitzt er mehrere Strategien, um blockierte Kreuzungen zu umfahren oder, wenn eine Verklemmung vorliegt, die Kreuzung vorsichtig zu überqueren.

6.2.3. Off-Road Fahren (Zustand *Zone*)

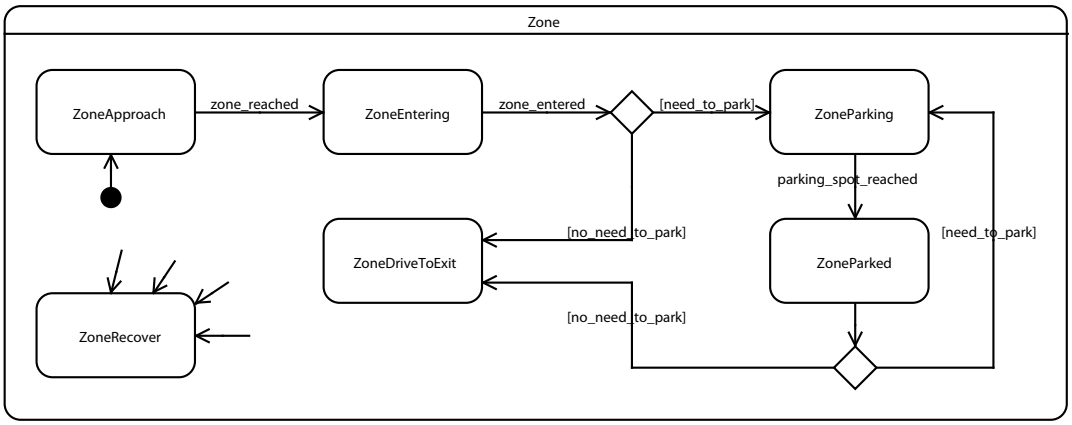


Abbildung 6.6.: Zustand *Zone* und seine Unterzustände (aus *Gindele* u. a. 2008).

Bei der DARPA Urban Challenge 2007 waren Bereiche vorgesehen (sog. *zones*[1]), in denen keine markierten Straßen existierten. Diese Bereiche muss das autonome Fahrzeug durchqueren können und auf speziell ausgewiesenen Parkplätzen einparken können. Im Zustandsautomaten sind der Zustand *Zone* und seine Unterzustände (siehe Abbildung 6.6) für die Behandlung dieser Bereiche verantwortlich. Dieser Zustand nimmt eine Sonderstellung ein, denn er instrumentiert eine externe Bahnplanungskomponente für unstrukturiertes Gelände anstatt die Regelung direkt anzusprechen, wie es im Standardfall geschieht.

6.2.4. Umplanungen (Zustand *Replan*)

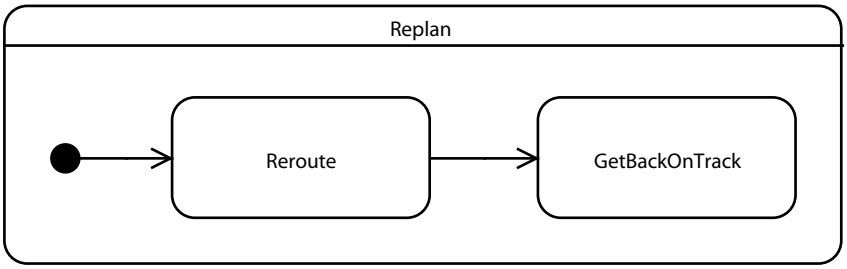


Abbildung 6.7.: Zustand *Replan* und seine Unterzustände (aus *Gindele* u. a. 2008).

Sollte in einem der anderen Zustände erkannt werden, dass die aktuelle Fahrbahn blockiert ist, wird in diesen Zustand gewechselt. Er und seine Unterzustände (siehe Abbildung 6.7) annotieren die Straßenkarte und berechnen eine neue Route durch das Straßennetz. Danach wird der nächstmögliche Zugangspunkt zu dieser neuen Route angefahren. Wenn sich das Fahrzeug wieder auf der Route befindet, wird in den normalen *Drive*-Modus gewechselt.

6.3. Ergebnisse

Mit dem in diesem Kapitel vorgestellten System erreichte das Team AnnieWAY (siehe Abbildung 6.8) als eines von 11 Teams das Finale der *DARPA Urban Challenge 2007* (siehe Abschnitt 2.3.2). Im Finale und den ihm vorgelagerten Ausscheidungsrunden zeigte der im vorherigen Abschnitt vorgestellte Zustandsautomat, dass er in der Lage ist, die folgenden Verhalten kollisionsfrei und den kalifornischen Verkehrsregeln entsprechend durchzuführen:

- Normales Fahren im Straßenverkehr.
- Abbiegen mit entgegenkommendem Verkehr.

[1] Engl. Zonen

Abbildung 6.8.: Ein Großteil des *Team AnnieWAY*, sein Fahrzeug und zwei DARPA-Offizielle bei der *DARPA Urban Challenge 2007*.

- Wechseln der Spur.
- Folgefahren und Überholen.
- Behandlung der Vorfahrtsregeln an einem 4-Way Stop.
- Einfädeln in fließenden Verkehr.

Ein Softwarefehler verhinderte beim Eintritt in einen Parkplatzbereich die Umschaltung auf den für unstrukturierte Bereiche zuständigen Bahnplaner. Nachdem das autonome Automobil mehrere Minuten stillstand, ohne Fortschritt zu erzielen, wurde es vom Veranstalter disqualifiziert. Bis zu diesem Zeitpunkt hat das autonome Automobil unfallfrei und den kalifornischen Verkehrsregeln entsprechend mehrere Fahrverhalten demonstrieren können.

6.4. Vergleich mit Verhaltensnetzwerk

In diesem Abschnitt soll der in diesem Kapitel vorgestellte Ansatz des Team AnnieWAY mit dem in der vorliegenden Arbeit entworfenen Verhaltensnetzwerk und der Systemarchitektur des SFB/TR28 verglichen werden. Da die SFB/TR28-Komponenten und speziell das Verhaltensnetzwerk noch nicht im Funktionsumfang an das System des Team AnnieWAY heranreichen, soll sich hier auf einen ersten, theoretischen Vergleich der beiden Architekturen beschränkt werden. Ein praktischer und quantitativer Vergleich beider Systeme, wenn sie einen vergleichbaren Funktionsumfang haben, ist sicher auch von Interesse.

Im Folgenden werden die Anforderungen aus Abschnitt 4.1 aufgegriffen und anhand von ihnen die beiden Systeme verglichen.

Unterstützung vielfältiger Ziele

Beide Systeme unterstützen vielfältige, parallele, nicht orthogonale Ziele. Im hierarchischen Zustandsautomaten des Team AnnieWAY ist die Unterstützung von Unterzielen, wie z.B. das Folgen eines Fahrzeugs während das Eigenfahrzeug einer Mission folgt, explizit durch Regeln in jedem Zustand des Automaten gelöst. Die Koordination und Auswahl des aktuell wichtigsten Ziels erfolgt auch durch feste Regeln. Das Verhaltensnetzwerk unterstützt das Verfolgen vielfältiger Ziele direkter: Jedes Verhalten kann ein eigenes, unabhängiges Ziel verfolgen. So verfolgt z.B. das *Abstand-halten*-Verhalten das Ziel, einem vorausfahrenden Fahrzeug zu folgen und das *Spur-folgen*-Verhalten verfolgt das Ziel, dem Missionsverlauf zu folgen. Die Verhalten können gleichzeitig aktiv sein und spezielle Fusionsknoten fusionieren ihre Ausgaben, um eine gemeinsame Verhaltensentscheidung zu treffen.

Im Ansatz vom Team AnnieWAY besteht die Schwierigkeit darin, die Komplexität zu beherrschen, die dieser Ansatz mit sich bringt. Ein Verfolgen von n gleichzeitigen, nicht orthogonalen Zielen bei m Zuständen erfordert mindestens das Einfügen von $n \cdot m$ Regeln im Zustandsautomaten, um die Ziele koordinieren zu können.

Die Schwierigkeit beim Ansatz des biologisch motivierten Verhaltensnetzwerks besteht darin, geeignete Datenstrukturen zu finden, die sich leicht fusionieren lassen und die Fahrintention der Verhalten so gut es geht widerspiegeln.

Unterstützung vielfältiger Sensoren

Diese Anforderung ist bei beiden Systemen nicht relevant. Die für diese Anforderung erforderliche Sensorfusion wird gegebenenfalls in der Wahrnehmungskomponente realisiert.

Daten-Fehlertoleranz

Da die Daten der Wahrnehmungsschicht immer mit einem Fehler behaftet sein werden, ist es notwendig Toleranzen bei der Verhaltensentscheidung einzufügen. Der Ansatz des Team AnnieWAY sieht hierfür einige Modellvereinfachungen, Fehlertoleranzbereiche und Fehlerbehandlungen vor. Ein Beispiel einer Modellvereinfachung ist, dass erkannte Fahrzeuge immer auf die Mitte der nächstgelegenen Spur positioniert werden. Ein Beispiel eines Fehlertoleranzbereichs ist die Erkennung von Fahrzeugen an Stopplinien: Hier wird auch noch ein Fahrzeug, das sich ein Meter von der Stopplinie entfernt befindet als an der Stopplinie stehend angesehen. Beispiele für Fehlerbehandlungen sind die *...Recover*-Zustände, die im Abschnitt 6.2 erwähnt werden. Das verwendete Konzept – der hierarchische Zustandsautomat – bietet für keine dieser Maßnahmen Hilfen.

Auch das in der vorliegenden Arbeit entwickelte Verhaltensnetzwerk verwendet Fehlertoleranzbereiche und Fehlerbehandlungen, um mit fehlerhaften Daten umgehen zu können. Beispielsweise wird die Ausrichtung von langsam fahrenden Fahrzeugen der Ausrichtung der Straße angepasst. Beispiel einer Fehlerbehandlung ist die Abbruchmöglichkeit beim Überholvorgang. Aber auch beim Verhaltensnetzwerk bietet das verwendete Konzept keine direkten Hilfen, um mit fehlerhaften Daten umgehen zu können.

Rechen-Fehlertoleranz

Auch beim Ausfall einiger Komponenten oder bei fehlerhaften Komponenten sollte eine, wenigstens eingeschränkte, Verhaltensentscheidung möglich sein. Der hierarchische Zustandsautomat des Team AnnieWAY unterscheidet nicht zwischen unterschiedlichen Reaktionsstufen. Strategische, taktische und reaktive Verhaltenskomponenten sind in seinen Zuständen vermengt. Sollte ein Zustand ausfallen oder fehlerhaft sein, beeinträchtigt dies also Verhaltensentscheidungen aller Reaktivitätsstufen.

Die explizite Aufteilung in Reaktivitätsstufen beim Ansatz des biologisch motivierten Verhaltensnetzwerks ist hier robuster. Wie Abschnitt 5.1.6 zeigt, kann dieser Ansatz auch bei fehlerhaften höheren Reaktivitätsschichten eine eingeschränkte Verhaltensentscheidung aufrecht erhalten. Selbst Fehler in der reaktiven Schicht können kompensiert werden, wie Abschnitt 5.1.7 zeigt.

Anpassungsvermögen

Die Verhaltensentscheidung muss auch mit einer sich schnell ändernden Umwelt zurechtkommen. Die Planungskomponente des Team AnnieWAY sieht deshalb eine hohe Taktrate für das Taktereignis des Zustandsautomaten vor. Alle 10 ms wird ein neues Taktereignis generiert und somit die Umwelt neu modelliert und aufgrund dieses erneuerten Modells der Umwelt eine neue Verhaltensentscheidung getroffen.

Der SFB/TR28-Ansatz sieht eine getrennte Weltmodellierung und Verhaltensentscheidung vor. Die Situationsinterpretation stellt ein Modell der Umgebung über die Echtzeitdatenbank der Verhaltensentscheidung zur Verfügung. Dieses Modell wird mit einem unterschiedlichen Takt aktualisiert als das Verhaltensnetzwerk Verhaltensentscheidungen trifft. Das Verhaltensnetzwerk führt seine unterschiedlichen Schichten mit unterschiedlichen Taktraten aus. Die strategische Schicht wird in einem langsameren Takt ausgeführt als die taktische, diese wiederum als die reaktive Schicht. Durch diesen asynchronen Prozess kann das Verhaltensnetzwerk auf sich schnell ändernde Umwelteinflüsse genauso reagieren, wie es rechenintensive, deliberative Verhaltensentscheidungen treffen kann.

Erweiterbarkeit

Das Hinzufügen neuer Verhaltensweisen sollte leicht möglich sein. Beim hierarchischen Zustandsautomaten des Team AnnieWAY ist das Hinzufügen einer neuen Verhaltensweise im Idealfall dadurch erreicht, dass unterhalb des *Active*-Zustands ein neuer Zustand hinzugefügt wird, der diese Verhaltensweise beherrscht. In wenigstens einigen der bestehenden Zuständen muss dieser neue Zustand durch Zustandsübergänge berücksichtigt werden. Beim Hinzufügen einer neuen Verhaltensweise müssen immer mehrere Zustände verändert werden. Das Prozedere dieses Idealfalls kann nur befolgt werden, wenn die neue Verhaltensweise keine Abhängigkeiten mit bereits bestehenden Zuständen hat.

Das Hinzufügen einer neuen Verhaltensweise beim biologisch motivierten Verhaltensnetzwerk kann durch Hinzufügen eines oder mehrerer neuer Verhaltensknoten geschehen. Im Idealfall kann hier auf vorhandene Verhaltensbausteine niederer Reaktivitätsstufen zurückgegriffen werden. Die Integration in die Verhaltensentscheidung wird durch das Anpassen oder Hinzufügen von Fusionsknoten erreicht.

Der Autor hatte die Gelegenheit, bei beiden Systemen selbst Erweiterungen vornehmen zu können und nach seiner Einschätzung lässt sich das Konzept des biologisch motivierten Verhaltensnetzwerks besser inkremental skalieren als das Konzept des hierarchischen Zustandsautomaten. Eine Validierung dieser Einschätzung wird die Praxis zeigen müssen.

Testbarkeit

Das automatisierte Testen einzelner Zustände ist im Planer des Team AnnieWAY nicht vorgesehen. Getestet wird hier nur das Gesamtsystem mithilfe der Simulation. Das in der vorliegenden Arbeit entwickelte Framework für biologisch motivierte Verhaltensnetzwerke unterstützt das automatisierte Testen einzelner Verhalten mit statischen Fixturen (siehe Abschnitt 4.2.8). Hierfür existiert eine prototypische Implementierung, die die Machbarkeit des Ansatzes demonstriert.

Simulierbarkeit

Die Planungskomponente des Team AnnieWAY besitzt einen festen Takt und ist somit nicht auf Simulation vorbereitet, die langsamer als in Echtzeit ausgeführt wird. Die Modifikation des Systems, so dass es nur einen Takt generiert, wenn ein neuer Simulationsschritt berechnet wird, sollte aber leicht möglich sein.

Das in dieser Arbeit entwickelte System bietet mit Signalemittern (siehe Abschnitt 4.2.4) und dem Signalprozessor (siehe Abschnitt 4.2.5) direkt die Möglichkeit, den Takt der verschiedenen Reaktivitätsstufen zu variieren und von externen Ereignissen abhängig zu machen.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein bestehendes biologisch motiviertes Verhaltensnetzwerk (*Albiez* 2006; *Hoffmann* 2007) auf ein in dieser Arbeit neu entwickeltes Framework migriert. Die Migration wurde notwendig, da erweiterte Anforderungen an das Verhaltensnetzwerk mit dem alten System nicht praktikabel umsetzbar waren. Es wurde ein Framework mit den folgenden Merkmalen entworfen und implementiert:

- Es wurde die gleiche Terminologie und Semantik, wie sie *Albiez* (2006) entwickelt hat, im Framework verwendet. Dies unterstützt die Entwicklung von biologisch motivierten Verhaltensnetzwerken, da diese so direkter in Quelltext umgesetzt werden können.

- Das Framework wurde in die Hard- und Softwarearchitektur des SFB/TR28 eingefügt. Dies gilt auf Softwareseite für die Integration der Echtzeitdatenbank und anderer, im SFB/TR28 entwickelter Hilfsmittel aber auch die Hardware der Architektur wurde beim Entwurf berücksichtigt. Aufgrund der SMP-Architektur der gewählten Rechnerplattform, wurde eine massiv parallel arbeitende Softwarearchitektur für das Verhaltensnetzwerk entwickelt. Die Verhaltensschichten und auch die einzelne Verhalten werden jeweils in einem eigenen Ausführungsfaden betrieben, um die Rechnerplattform bestmöglich ausnutzen zu können. Damit diese massive Parallelität auch tatsächlich ausgenutzt werden kann, wurde eine spezielle Datenabstraktion entworfen und implementiert. Dieses Konzept abstrahiert zum einen die Anbindung an die Echtzeitdatenbank aber zum anderen, wichtigeren Teil ermöglicht es den Zugriff auf gemeinsam genutzte Daten in einer effizienten und – da Verklemmungen ausgeschlossen sind – sicheren Weise.
  In Tests auf dem Versuchsträger und in der Simulation konnte gezeigt werden, dass diese Integration erfolgreich war. Auch rechenaufwendige Aufgaben, wie das Erstellen und Fusionieren der Risikokarten, sind mit der hier entwickelten Architektur parallel zur Ausführung des Netzwerks in Echtzeit möglich.

- Es wurde eine spezielle Testumgebung entwickelt und prototypisch implementiert, mit der automatisierte Tests einzelner Verhaltensbausteine möglich sind. Im Laufe der Entwicklung dieser Arbeit – aber auch schon bei Arbeiten am Vorgängersystem – wurde deutlich, dass bei der Vernetzung der Einzelkomponenten zu einem komplexen System

wie dem Verhaltensnetzwerk sichergestellt sein muss, dass die Einzelkomponenten, also hier die Verhaltensbausteine, gemäß ihrer Spezifikation funktionieren. Ist diese Voraussetzung nicht gegeben, so ist eine Vernetzung der Komponenten praktisch nicht möglich. Automatisierte Tests sind ein gutes Mittel, um die Einhaltung der Spezifikation eines Verhaltens auch bei inkrementellen Änderungen zu belegen.

- Zur Unterstützung der Simulierbarkeit des Verhaltensnetzwerks wurde die Ausführung der einzelnen Verhaltensschichten voneinander entkoppelt. Es wurde ein System entwickelt und implementiert, das es externen (Datenbank-)Ereignissen erlaubt, die unterschiedlichen Schichten unabhängig voneinander auszuführen. Eine koordinierende Komponente nimmt diese Ereignisse entgegen und kann eventuell notwendige Abhängigkeiten durchsetzen.[1]

- Das Framework wurde so konzipiert und implementiert, dass die Abhängigkeiten, die ein Verhalten zu anderen Verhalten und Datenabstraktionen besitzt, explizit und symbolisch angegeben werden müssen. Aus diesen expliziten Abhängigkeiten wurde unter anderem eine Übersicht über die Netzwerkstruktur des Verhaltensnetzwerks automatisch generiert. Eine Visualisierung der Netzwerkstruktur wurde implementiert, die auch als Visualisierung der Motivationen und virtuellen Sensorwerte der Verhalten eines laufenden Verhaltensnetzwerks dient und so eine Hilfestellung bei der Fehlersuche ist.

Mithilfe dieses Frameworks wurde das bestehende Verhaltensnetzwerk migriert, erweitert und an geänderte Schnittstellen zu anderen Systemkomponenten angepasst. Tests in der Simulation und auf dem Versuchsträger haben gezeigt, dass das in der vorliegenden Arbeit entwickelte Verhaltensnetzwerk die folgenden Fahrverhalten durchführen kann:

- Das Folgen eines Straßenverlaufs, ohne dass andere dynamische Hindernisse involviert sind.

- Das Folgen eines anderen Fahrzeugs mit Berücksichtigung der aktuell erlaubten Höchstgeschwindigkeit und der Geschwindigkeit des vorausfahrenden Fahrzeugs.

- Das Überholen eines stehenden Fahrzeugs bei Beachtung des Verkehrs und des Straßenverlaufs.

- Das Überholen eines fahrenden Fahrzeugs bei Beachtung des Verkehrs und des Straßenverlaufs.

Außerdem wurde die Robustheit des Verhaltensnetzwerks gegenüber Fehlern demonstriert. Die Verhaltensbausteine, die notwendig sind, um 4-Way-Stop-Kreuzungen zu passieren, wurden implementiert. Sie konnte allerdings nicht demonstriert werden, da die hierfür notwen-

[1] Eine denkbare Abhängigkeit wäre z.B., dass die reaktive Schicht immer automatisch ausgeführt wird, wenn eine der oberen Schichten ausgeführt wurde.

digen Informationen der Situationsinterpretation bis zur Vollendung dieser Arbeit nicht zur Verfügung standen. Die enge Kopplung von Verhaltensentscheidung und Bahnplanung wurde aufgelöst. Risikokarten wurden in der reaktiven Schicht als einheitliche Beschreibung der Fahrintention der Verhalten eingeführt.

Das in der vorliegenden Arbeit entwickelte System und die zugehörigen SFB/TR28-Komponenten wurden mit dem hierarchischen Zustandsautomaten verglichen, der im Team AnnieWAY während der DARPA Urban Challenge 2007 für die Verhaltensentscheidung verantwortlich war. Bei diesem theoretischen Vergleich wurde sich auf qualitative Aussagen beschränkt, weil ein quantitativer Vergleich aufgrund des unterschiedlichen Funktionsumfangs nicht praktikabel ist.

7.2. Ausblick

Als einer der nächsten Schritte sollte die Anzahl der vom Verhaltensnetzwerk beherrschten Fahrverhalten vergrößert werden. Für die Erweiterung des Netzwerks können die vorhandenen Verhaltensbausteine wiederverwendet werden. Eine Anpassung von ihnen oder auch das Hinzufügen neuer reaktiver Verhalten wird voraussichtlich notwendig sein. Insbesondere die aktuelle Behandlung der Geschwindigkeit des Eigenfahrzeugs im Verhaltensnetzwerk als zeitlich konstant und ohne nähere Definition der gewünschten Beschleunigung wird für dynamischere Fahrmanöver nicht ausreichend sein. Hierfür müssen sowohl die Schnittstelle zur Bahnplanung als auch der Geschwindigkeitsfusionsknoten der reaktiven Schicht angepasst werden. Die Schnittstelle zur Bahnplanung muss außerdem für kompliziertere Fahrverhalten, wie das Einparken, angepasst werden.

Bei der Weiterentwicklung des Verhaltensnetzwerks sollte auch die prototypisch implementierte automatisierte Testumgebung ausgebaut werden. Um Verhalten auch in dynamischen Szenen automatisch testen zu können, sollte die eingeführten Fixturen erweitert werden, so dass sie zeitlich veränderbar sind.

Wenn das Verhaltensnetzwerk, die Bahnplanung und die Situationsinterpretation den Funktionsumfang der Planungskomponente des Team AnnieWAY erreicht haben, wäre ein erneuter, umfassenderer und auch praktischer Vergleich der beiden Systeme von Vorteil, um hierdurch Schwächen des gewählten Ansatzes zu identifizieren und beheben zu können.

Um auch in mehrdeutigen Situationen und unter schlechten Wahrnehmungsbedingungen robustere Verhaltensentscheidungen treffen zu können, sollte das Verhaltensnetzwerk längerfristig mit Wahrscheinlichkeiten versehene Daten der Situationsinterpretation und der Wahrnehmung auswerten können.

A. Anhang: Implementierungsdetails

In diesem Anhang werden einige Implementierungsdetails vorgestellt, die im wissenschaftlichen Teil dieser Ausarbeitung keinen Platz fanden, aber dennoch als erwähnenswert angesehen werden. Dieses Kapitel verlangt einiges Vorwissen über die Entwicklungs- und Ausführungsumgebung des SFB/TR28, da es primär als Einstieg in die Weiterentwicklung des Verhaltensnetzwerks gedacht ist und dieses Vorwissen bei einer solchen als gegeben angenommen werden kann.

A.1. Ein neues Verhalten hinzufügen

Dieser Abschnitt beschreibt, was notwendig ist, um ein neues Verhalten dem Verhaltensnetzwerk hinzuzufügen. Es wird davon ausgegangen, dass das Verhaltensnetzwerk an sich kompilierbar ist. Dass also das Verhaltensnetzwerk aus dem SFB/TR28-SVN ausgecheckt ist und die RACE-Umgebung aufgesetzt wurde.

- Das neue Verhalten wird durch eine neue Unterklasse von `Behaviour` realisiert. Die neue Klasse sollte entsprechend der Schicht, in das das Verhalten eingefügt werden soll, benannt werden. Ein neues reaktives Verhalten erhält den Präfix `RB_`, ein taktisches Verhalten den Präfix `TB_` und ein neues strategisches Verhalten den Präfix `SB_`. Außerdem sollten die Headerdatei und die .cpp-Datei in dem der Schicht entsprechenden Verzeichnis eingeordnet sein. Unter `template/behaviour.{h|cpp}` findet sich eine Vorlage für ein neues Verhalten.

- In dem **`enum`** `ID` der Klasse `Behaviour` (`src/utils/behaviour.h`) muss für das neue Verhalten ein neuer Eintrag hinzugefügt werden. Dabei sollte der Eintrag genau so heißen wie die Klasse.

- Die Abhängigkeiten zu anderen Verhalten und den Datenabstraktionen müssen in den beiden Präprozessor-Makros `DATA_LIST` und `BEHAVIOUR_CONNECTIONS` definiert werden. Ein Beispiel und eine kurze Beschreibung der Syntax findet sich in der oben erwähnten Vorlage.

- Damit das Verhalten in seiner Schicht erzeugt wird, muss es in der `createBehaviours`-Methode durch den folgenden Quelltext erzeugt werden:

```
#include "Verhalten.h"
...
void createBehaviours(){
...
BEHAVIOUR(Verhalten)
}
```

- Ein »`make qmake`« in dem Hauptverzeichnis erzeugt/verändert die notwendigen Makefiles, um das Verhaltensnetzwerk mit dem neuen Verhalten zu übersetzen.

A.2. Eine neue Datenabstraktion hinzufügen

Neben dem Hinzufügen eines neuen Verhaltens wird eine eventuelle Modifikation des Verhaltensnetzwerks auch aus dem Hinzufügen einer neuen Datenabstraktion bestehen. Z.B. wenn ein neues Datenbankobjekt in dem Verhaltensnetzwerk zur Verfügung stehen soll, ist es notwendig, eine neue Datenabstraktion für dieses Datenbankobjekt zu erstellen. Im Folgenden sollen deshalb die notwendigen Schritte anhand des folgenden, fiktiven Datenbankobjekts `kogmo_rtdb_obj_b2_fakeobj_t` erklärt werden:

```
typedef struct {
        int a;
        double b;
} kogmo_rtdb_subobj_b2_fakeobj_t;

typedef struct {
        kogmo_rtdb_subobj_base_t base;
        kogmo_rtdb_subobj_b2_fakeobj_t fake;
} kogmo_rtdb_obj_b2_fakeobj_t;
```

Da es sich um ein Objekt aus dem Teilbereich B2 handelt, sollte die Datenabstraktion in der Datei `src/utils/data/B2_data.h` definiert werden:

```
typedef DBData<KOGMO_RTDB_OBJTYPE_B2_FAKEOBJ_V1, kogmo_rtdb_subobj_b2_fakeobj_t>
    DBData_B2_FakeObj;
```

Dieses Typedef erstellt eine von `Data` abgeleitete Klasse `DBData_B2_FakeObj`. Um diese Unterklasse im System bekannt zu machen, sind noch zwei Schritte zu tun: Für die neue Datenabstraktion muss ein neuer Eintrag in der Aufzählung `Data::ID` hinzugefügt werden und es muss eine Verbindung zwischen diesem neuen Eintrag in der Aufzählung und der neuen Klasse erstellt werden.

Die Aufzählung `Data::ID` befindet sich in der Header-Datei der `Data`-Klasse: `data.h`. Sie muss um einen neuen Punkt erweitert werden:

```
...
class Data {
...
        enum ID {
                ...
                B2_FakeObj
                ...
        }
...
};
```

Die Position des neuen Aufzählungspunktes *ist* von Bedeutung, denn die Position innerhalb dieser Aufzählung bestimmt die starke Ordnung in der die Datenabstraktionen reserviert werden müssen (siehe Abschnitt 4.2.3).

A.3. Grafisches Testen eines Verhaltens

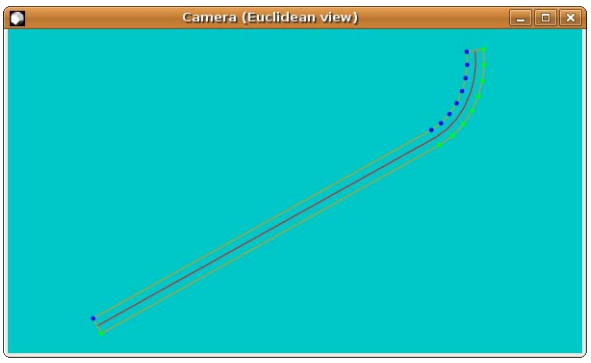


Abbildung A.1.: Visualisierung einer Spur durch Geomview.

Bei der Implementierung des Verhaltensnetzwerks hat es sich als sehr wertvoll erwiesen, die notwendigen geometrischen Berechnungen schnell und unkompliziert visualisieren zu können. Hierfür wurde die Geomview-Anbindung von CGAL (*Fabri* und Pion 2007) verwendet. In der Header-Datei `utils/geometric/geomview_debug.h` finden sich Hilfsroutinen und Makros, die das ohnehin schon einfache Handling dieser Stream-Schnittstelle vereinfachen. Eine Verwendung dieses grafischen Testhilfsmittels findet sich beispielsweise in dem *Spur-halten*-Verhalten (Klasse `RB_KeepLane`). Von diesem Verhalten stammt auch der in Abbildung A.1 dargestellte Bildschirmausschnitt. Er zeigt das Polygon, aus dem das Verhalten die für die Erstellung der Risikokarte notwendige Datenstruktur erstellt.

A.4. Log-Bibliothek

In diesem Abschnitt wird ein Framework vorgestellt, das in einer Anwendung mit mehreren Ausführungsfäden einen konsistenten Log-Mechanismus zur Verfügung stellt, der eine eindeutige Zuordnung der Nachrichten zu den Ausführungsfäden sicherstellt.

Es wurde darauf geachtet, die Schnittstelle zu der Log-Bibliothek soweit es geht der normalen C++-Stream-Schnittstelle anzupassen und die erweiterten Funktionen in diese Schnittstelle zu integrieren.

Nach dem Einbinden der Header-Datei mit `#include <mtlog.h>` können vier neue Streams benutzt werden:

- `mtLog`
- `mtWarn`
- `mtError`
- `mtDebug`

Diese vier Variablen können genau so benutzt werden, wie `std::cout` oder `std::cerr`. Es kann also geschrieben werden:

```
1 mtLog << "Some␣String";
```

Und natürlich sind alle auf `std::ostream` definierten `<<`-Operatoren auch mit den Log-Streams benutzbar:

```
2 int i = 0;
3 mtLog << "i␣equals␣" << i;
```

Hier soll nicht weiter auf die normale Stream-Schnittstelle eingegangen werden; es wird angenommen, dass der Leser mit dieser, in C++ essentiellen, Schnittstelle vertraut ist. Deshalb wird hier nur auf die folgenden Erweiterungen und Abweichungen eingegangen:

- Die Streams sind nicht Teil eines C++-Namensraumes. Sie können überall einfach mit `mt{Log|Warn|Error|Debug}` erreicht werden.[1]
- Jeder Ausführungsfaden kann/muss einen Namen erhalten, der jeder Ausgabe, die in diesem Ausführungsfaden getätigt wird, vorangestellt wird. Den Namen eines Ausfüh-

[1] Dies ist ein Nachteil der aktuellen Implementierung, da die Streams durch C-Präprozessor-Makros definiert werden.

rungsfadens setzt man, indem man einmalig innerhalb dieses Ausführungsfadens die Methode `mt_log::set_thread_name(const char* name)` aufruft. Also z.B.

```
#include <pthread.h>
#include <mtlog.h>

void *PrintHello(void *threadid)
{
   mt_log::set_thread_name("HelloPrinter");
   mtLog << "Hello␣World!␣It's␣me";
   pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
   pthread_t hello_printer;
   mt_log::set_thread_name("in␣main");
   mtLog << "creating␣thread";
   pthread_create(&hello_printer, NULL, PrintHello,0);
   pthread_exit(NULL);
}
```

Sollte `set_thread_name` nicht aufgerufen werden, wird eine generischere, numerische Zeichenkette erzeugt, die den Ausführungsfaden eindeutig identifiziert.

- Es wird automatisch ein Zeilenumbruch nach jeder Ausgabe eingefügt.

```
mtLog << "logline\n";
```

oder

```
mtLog << "logline" << std::endl;
```

ist deshalb nicht nötig. Es kann aber für mehrzeilige Nachrichten benutzt werden:

```
   mtLog << "logline1\nlogline2" << std::endl
         << "logline3";
```

- Die Daten werden *nicht* sofort auf die Konsole geschrieben, sondern durch einen gesonderten Ausführungsfaden in einem festen Intervall ausgegeben. Dies stellt zum einen sicher, dass die Nachrichten in einer geordneten Reihenfolge seriell ausgegeben werden und zum anderen beugt es eine Überlastung der Konsole vor, wenn sehr viele Nachrichten in einer kurzen Zeit ausgegeben werden sollen. Der Hauptgrund, die Konsolenausgabe in einen gesonderten Ausführungsfaden zu verlagern ist aber, dass somit die Algorithmen, die solche Lognachrichten ausgeben, nicht durch die relativ langsamen I/O-Operationen ausgebremst werden.

B. Anhang: Hinweise zur Bedienung

Dieser Anhang möchte dem Leser Hilfestellung geben beim Einrichten und Verwenden von einem Teil der im Rahmen dieser Arbeit entwickelten Software. Auch in diesem Anhang wird, wie im Anhang A, Vorwissen über die Entwicklungs- und Ausführungsumgebung vorausgesetzt.

B.1. Einrichtung

Das Verhaltensnetzwerk setzt eine eingerichtete RACE-Umgebung voraus. D.h. RACE muss aus dem Subversion-Repository heruntergeladen worden sein, die für RACE notwendigen Bibliotheken wurden installiert (siehe `README.txt`) und RACE wurde konfiguriert und übersetzt. Das Verhaltensnetzwerk braucht eine Umgebugsvariable `RACE_HOME`, die auf das Hauptverzeichnis[1] von RACE zeigt. Sind diese Vorbedingungen erfüllt, kann im Hauptverzeichnis des Verhaltensnetzwerks der folgende Befehl in der Konsole das Verhaltensnetzwerk, die automatisierten Tests und die beiden im Folgenden beschriebenen Visualisierungen kompilieren:

```
make all
```

Auf den Rechnern des IAIM sollte man allerdings dem Befehl noch `pump` voranstellen, um von der `distcc`-Installation profitieren zu können.

```
pump make all
```

Die Programme und Bibliotheken werden in das `bin`-Unterverzeichnis des Verhaltensnetzwerk-Hauptverzeichnis kopiert.

[1] Also das Verzeichnis, das `src`, `data` etc. als Unterverzeichnisse besitzt.

B.2. Das Programm behaviour_network

Das behaviour_network-Programm beinhaltet das in dieser Arbeit implementierte Verhaltensnetzwerk. Es besitzt die folgenden Komandozeilenoptionen:

```
~/workspace/b2_behaviour_ng/$ bin/behaviour_network --help
Allowed options:
  -h [ --help ]                         output command line help and quit
  -w [ --write-network-definition ] arg writes a network definition XML file
                                        with the current network definition of
                                        the behaviour network. Do that once if
                                        you have altered the network structure.
  -d [ --database ] arg                 connect to the database with name
                                        databasename. If not specified, use the
                                        standard database in
                                        $KOGMO_RTDB_DBHOST.
  -p [ --ini-file ] arg                 currently this option is ignored
```

Es setzt – wie jede SFB/TR28-Komponente – eine laufende Echtzeitdatenbank voraus. Außerdem benötigt es eine Position des Eigenfahrzeugs in der Datenbank (»*UNIBW Ins-Data*«) und die Daten, die von der Situationsinterpretation zur Verfügung gestellt werden.

Sind diese Vorbedingungen erfüllt, kann das Programm ohne Angabe von Parametern ausgeführt werden.

B.3. Das Programm gui_behaviour

gui_behaviour ist die Visualisierung der Netzwerkstruktur des Verhaltensnetzwerks. Mit dem folgenden Befehl ausgeführt im Hauptverzeichnis des Verhaltensnetzwerks kann eine XML-Datei erstellt werden, die die aktuelle Struktur des Verhaltensnetzwerks enthält:

```
~/workspace/b2_behaviour_ng/$ bin/behaviour_network -w network.xml
```

Diese Datei kann nun mit der GUI gui_behaviour geöffnet werden. In der GUI können die Positionen der Verhalten durch Drag&Drop noch von Hand angepasst werden. Solche angepassten Netzwerkstrukturen können wieder in die XML-Datei gesichert werden, damit sie für die nächste Programmausführung zur Verfügung stehen. Abbildung B.1 zeigt gui_behaviour nachdem es eine aktuelle XML-Datei geladen hat. Auf dem Bildschirmausschnitt kann man auch die zweite Funktion des Programms sehen: Es kann sich mit einer Echtzeitdatenbank verbinden und die aktuellen Motivations-, Aktivitäts- und Reflexionswerte der Verhalten eines

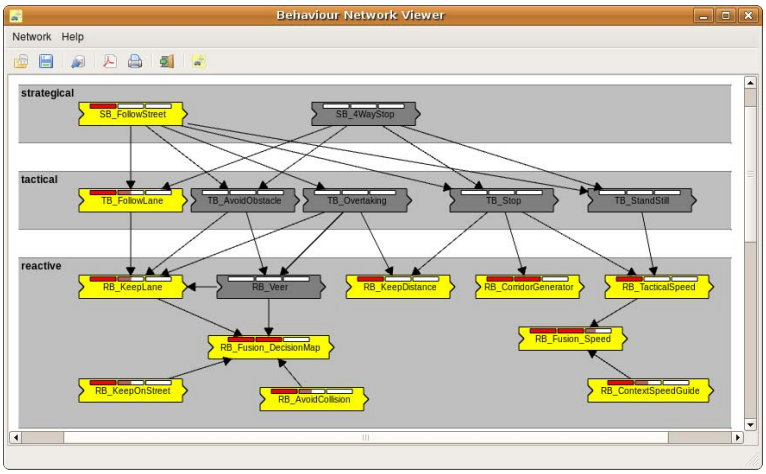
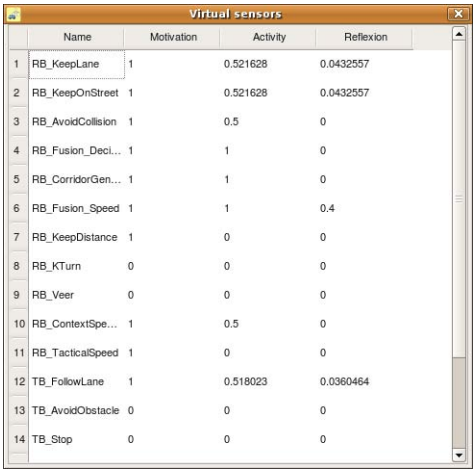


Abbildung B.1.: Die Netzwerkvisualisierung gui_behaviour. Dargestellt ist das aktuelle Verhaltensnetzwerk und die Motivation, Aktivität und Reflexion der Verhalten.



	Name	Motivation	Activity	Reflexion
1	RB_KeepLane	1	0.521628	0.0432557
2	RB_KeepOnStreet	1	0.521628	0.0432557
3	RB_AvoidCollision	1	0.5	0
4	RB_Fusion_Deci...	1	1	0
5	RB_CorridorGen...	1	1	0
6	RB_Fusion_Speed	1	1	0.4
7	RB_KeepDistance	1	0	0
8	RB_KTurn	0	0	0
9	RB_Veer	0	0	0
10	RB_ContextSpe...	1	0.5	0
11	RB_TacticalSpeed	1	0	0
12	TB_FollowLane	1	0.518023	0.0360464
13	TB_AvoidObstacle	0	0	0
14	TB_Stop	0	0	0

Abbildung B.2.: Zusätzliche tabellarische Darstellung der Motivation, Aktivität und Reflexion aller Verhalten.

laufenden Verhaltensnetzwerks anzeigen. Zum einen erfolgt dies grafisch direkt in der Netzwerkstruktur, zum anderen wird zusätzlich eine tabellarische Übersicht (siehe Abbildung B.2) der Werte gegeben.

B.4. Das Programm `gui_decisionmap`

Abbildung B.3.: Die Visualisierung der Risikokarten `gui_decisionmap`. Die Karten der Verhalten *Spur halten* und *Kollision vermeiden* und die fusionierte Risikokarte sind zu sehen. Eine optionale Falschfarbendarstellung der Risikokarten ist in diesen Bildschirmfoto nicht aktiviert.

B.5. Ein typischer Ablauf einer Simulationssitzung

In diesem Abschnitt sollen die Schritte, die für das Starten einer Simulation notwendig sind, kurz erläutert werden. Es wird vorausgesetzt, dass alle Programme eingerichtet sind. Um die speziellen Visualisierungen von `kogmo_view` verwenden zu können, muss RACE (oder auch nur `kogmo_view`) mit dem nachfolgenden Befehl übersetzt werden:

```
make B12_EXTRAS=1
```

RACEs `annieway.ini` muss entsprechend dem Test angepasst werden – hier ist besonders auf den Koordinatenoffset zu achten. Das verwendete RNDF sollte mit dem Skript `set_rndf`, das sich im RACE-`bin`-Verzeichnis befindet, gesetzt werden. Sind diese Vorbereitungen getroffen, setzen die folgenden Befehle die Simulationsumgebung auf:

1. Zuerst sollten einige Konsolen geöffnet werden, damit jedes Programm seine eigene Konsole besitzt. Die folgenden Befehle öffnen 7 Konsolen im RACE-`bin`-Verzeichnis:

   ```
   cd $RACE_HOME/bin
   ./popup 7
   ```

2. In der ersten Konsole sollte man die Echtzeitdatenbank starten:

   ```
   ./kogmo_rtdb_man -nd
   ```

3. Die zweite Konsole kann `kogmo_view` beherbergen:

   ```
   ./kogmo_view
   ```

4. `fake_controller`, die Fahrzeugsimulation, startet man mit dem folgenden Befehl:

   ```
   ./fake_controller
   ```

   Ohne Argumente werden die Startposition und die Ausrichtung aus der `annieway.ini`-Datei übernommen.

5. Die Situationsinterpretation startet man auch ohne Parameter:

   ```
   ./interpretation
   ```

6. Als nächstes startet man das Verhaltensnetzwerk:

   ```
   cd PFAD_ZUM_VERHALTENSNETZWERK
   bin/behaviour_network
   ```

7. Nun ist das System so weit, dass der Simulationskreislauf geschlossen werden kann. Die einzig fehlende Komponente ist die Bahnplanung:

   ```
   cd PFAD_ZUR_BAHNPLANUNG
   bin/uppm
   ```

8. Das simulierte Auto wird jetzt schon fahren. Um noch andere Verkehrsteilnehmer zur Szene hinzuzufügen gibt es zwei Möglichkeiten: `vehicle_write` für statische oder semi-statische Objekte und `otdm_rtdb_server` für dynamische, mit einem Joystick gesteuerte Objekte. Hier soll nur die zweite Möglichkeit näher erklärt werden, da `vehicle_write` ohne Erklärung zu bedienen sein sollte. In einer von den per `popup` geöffneten Konsolen erzeugt der folgende Befehl ein dynamisches Objekt, das mit dem ersten am System angeschlossenen Joystick gesteuert werden kann:

```
./otdm_rtdb_server -j /dev/input/js0 \
  -c ../param/joystick0.calib \
  -o $KOGMO_RTDB_DBHOST -x 0 -y 0
```

Der Joystick-Server besitzt noch eine Fülle andere Komandozeilenoptionen, die er mit der Option `--help` preisgibt.

Wenn die Simulationssitzung beendet werden soll, reicht ein Aufruf des Skripts `kogmo_rtdb_kill`, um alle Programme zu beenden.

Abbildungsverzeichnis

Literaturverzeichnis

Die Literaturangaben sind alphabetisch nach den Namen der Autoren und dem Erscheinungsjahr der Publikation sortiert. Bei mehreren Autoren wird nach dem ersten Autor sortiert. Die Zahlen in eckigen Klammern am Ende eines jeden Eintrags geben die Abschnitte in der Ausarbeitung an, in denen auf die Literaturangabe verwiesen wird.

***Albiez* 2006** ALBIEZ, Jan C.: *Verhaltensnetzwerke zur adaptiven Steuerung biologisch motivierter Laufmaschinen*, Universität Karlsruhe, Dissertation, 2006 [1.1, 2.1.6, 2, 2.1.7, 3.4, 7.1]

***Arkin* 1987** ARKIN, Ronald C.: Motor schema based navigation for a mobile robot: An approach to programming by behavior. In: *Robotics and Automation. Proceedings. 1987 IEEE International Conference on* 4 (1987), Mar, S. 264–271 [2.1.3]

***Arkin* 1989** ARKIN, Ronald C.: Motor Schema – Based Mobile Robot Navigation. In: *The International Journal of Robotics Research* 8 (1989), Nr. 4, S. 92–112. – URL http://ijr.sagepub.com/cgi/content/abstract/8/4/92. – Zugriffsdatum: 2008-12-13 [2.1.3]

***Arkin* 1998** ARKIN, Ronald C.: *Behavior-Based Robotics*. MIT Press, 1998. – ISBN 0-262-01165-4 [2.1.1, 2.1, 2.1.3, B.5]

***Booch* u. a. 1997a** BOOCH, G. ; JACOBSON, I. ; RUMBAUGH, J.: UML Notation Guide, Version 1.1. In: *UML Partners Consortium* (1997) [2.2]

***Booch* u. a. 1997b** BOOCH, G. ; JACOBSON, I. ; RUMBAUGH, J.: UML Semantics, Version 1. In: *Rational Software Corporation* (1997) [2.2]

Boost *The Boost Library*. – URL http://www.boost.org/. – Zugriffsdatum 2008-12-13. [2.4.3]

Braess und Reichart 1997a Braess, H.-H. ; Reichart, G.: Prometheus: Vision des 'intelligenten Automobils' auf 'intelligenter Straße'? Versuch einer kritischen Würdigung – Teil 1. In: *ATZ Automobiltechnische Zeitschrift* (1997), Nr. 4, S. 200–205 [3]

Braess und Reichart 1997b Braess, H.-H. ; Reichart, G.: Prometheus: Vision des 'intelligenten Automobils' auf 'intelligenter Straße'? Versuch einer kritischen Würdigung – Teil 2. In: *ATZ Automobiltechnische Zeitschrift* (1997), Nr. 6, S. 330–343 [3]

Braitenberg 1984 Braitenberg, Valentino: *Vehicles, Experiments in Synthetic Psychology*. MIT Press, 1984. – ISBN 0-262-02208-7 [2.1.1, 2.1.2]

Broggi u. a. 2001 Broggi, Alberto ; Bertozi, Massimo ; Conte, Gianni ; Fasciolo, Alessandra: ARGO prototype vehicle. In: Vlacic, Ljubo (Hrsg.): *Intelligent vehicle technologies*. Butterworth-Heinemann, 2001, Kap. 14, S. 445–493. – ISBN 0-7506-5093-1 [2.3.1]

Brooks 1986 Brooks, Rodney A.: A robust layered control system for a mobile robot. In: *IEEE Journal of Robotics and Automation* RA-2 (1986), April, Nr. 1, S. 14–23 [2.2, 2.1.2, 2.3, 4.1, 4.1.2]

CGAL *Cgal, Computational Geometry Algorithms Library*. – URL http://www.cgal.org. – Zugriffsdatum: 2008-12-13. [2.4.2]

CGAL Editorial Board 2007 CGAL Editorial Board: *CGAL User and Reference Manual*. 3.3, 2007. – URL http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html. – Zugriffsdatum: 2008-12-13 [2.4.2]

Craig 2005 Craig, John J.: *Introduction to robotics*. 3. internat. Auflage. Pearson/Prentice Hall, 2005. – ISBN 0-13-123629-6 [2]

DARPA 2007 *DARPA Urban Challenge 2007*. 2007. – URL http://www.darpa.mil/grandchallenge/. – Zugriffsdatum 2008-12-13. [1, 2.3.2]

DARVIN 2008 DARVIN: *Projektbeschreibung*. Online-Quelle. 2008. – URL http://i21www.ira.uka.de/darvin/projektbeschreibung.html. – Zugriffsdatum: 2008-05-29 [2.1.5]

Destatis 2006 Statistisches Bundesamt Deutschland: *Im Blickpunkt: Verkehr in Deutschland 2006*. Statistisches Bundesamt Deutschland, 2006 [1]

***Dickmanns* 2002** Dickmanns, E.D.: The development of machine vision for road vehicles in the last decade. In: *Intelligent Vehicle Symposium, 2002. IEEE* 1 (2002), 17-21 June, S. 268–281 vol.1 [2.3.1]

***Dowling* u. a. 1987** Dowling, Kevin ; Guzikowski, R. ; Ladd, J. ; Pangels, Henning ; Singh, Sanjiv ; Whittaker, William Red L.: NAVLAB: A Autonomous Navigation Testbed / Robotics Institute, Carnegie Mellon University. Pittsburgh, PA, November 1987 (CMU-RI-TR-87-24). – Forschungsbericht [2.3.1, 2.13]

***Evans* 1985** Evans, Leonard: Human behaviour feedback and traffic safety. In: *Human Factors* 27 (1985), Nr. 5, S. 555–576 [1]

***Evans* 2004** Evans, Leonard: *Traffic Safety*. Bloomfield Hills, Mich. : Science Serving Society, 2004. – ISBN 0-9754871-0-8 [1]

***Fabri* und Pion 2007** Fabri, Andreas ; Pion, Sylvain: Geomview. In: Board, CGAL E. (Hrsg.): *CGAL User and Reference Manual*. 3.3. 2007 [A.3]

***Gerber* 2000** Gerber, R.: *Natürlichsprachliche Beschreibung von Straßenverkehrsszenen durch Bildfolgenauswertung*. 2000, Fakultät für Informatik der Universität Karlsruhe (TH), Diplomarbeit, Januar 2000 [2.1.5]

***Gindele* 2007** Gindele, Tobias: *Wissensrepräsentation von Verkehrssituationen und Analyse mittels fallbasiertem Schließen für kognitive Automobile*, Institut für Technische Informatik, Universität Karlsruhe, Diplomarbeit, 2007 [3.3]

***Gindele* u. a. 2008** Gindele, Tobias ; Jagszent, Daniel ; Pitzer, Benjamin ; Dillmann, Rüdiger: Design of the planner of Team AnnieWAY's autonomous vehicle used in the DARPA Urban Challenge 2007. In: *Proc. IEEE Intelligent Vehicles Symposium (IV'08) in Eindhoven, The Netherlands*, IEEE Press, June 2008, S. 1131–1136 [6, 6.3, 6.4, 6.5, 6.6, 6.7]

***Goebl* und Färber 2007** Goebl, Mathias ; Färber, Georg: A Real-Time-capable Hard- and Software Architecture for Joint Image and Knowledge Processing in Cognitive Automobiles. In: *Proceedings of the IEEE Intelligent Vehicles Symposium* (2007), S. 734–740 [3.1.1]

***Goto* und Stentz 1987** Goto, Y. ; Stentz, A.: The CMU system for mobile robot navigation. In: *Robotics and Automation. Proceedings. 1987 IEEE International Conference on* 4 (1987), Mar, S. 99–105 [2.3.1]

Harel 1987 Harel, D.: Statecharts: A visual formalism for complex systems. In: *Science of Computer Programming* 8 (1987), Nr. 3, S. 231–274 [2.2]

Hart u. a. 1968 Hart, PE ; Nilsson, NJ ; Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *Systems Science and Cybernetics, IEEE Transactions on* 4 (1968), Nr. 2, S. 100–107 [2.3.1]

Hoffmann 2007 Hoffmann, Markus: *Biologisch motiviertes Verhaltensnetzwerk zur Steuerung eines kognitiven Automobils*, Institut für Technische Informatik, Universität Karlsruhe, Diplomarbeit, 2007 [1.1, 2.1.7, 2.9, 3.4, 1, 4.3, 4.3.1, 7.1, B.5]

Howard u. a. 2006 Howard, Thomas ; Knepper, Ross A. ; Kelly, Alonzo: Constrained Optimization Path Following of Wheeled Robots in Natural Terrain. In: *Proceedings of the 10th International Symposium on Experimental Robotics 2006 (ISER '06)*, July 2006 [2.3.2]

Jagszent 2007 Jagszent, Daniel: *Betrachtungen zur Verhaltensauswahl am Beispiel eines Abbiegevorganges*, Institut für Technische Informatik, Universität Karlsruhe, Studienarbeit, 2007 [4.3.2, 4.9]

Jagszent u. a. 2007 Jagszent, Daniel ; Schröder, Joachim ; Zöllner, Marius ; Dillmann, Rüdiger: An Approach for Behaviour Selection in an autonomous Vehicle. In: *6th IFAC Symposium on Intelligent Autonomous Vehicles* (2007), September [4.3.2]

Kammel u. a. 2008 Kammel, Sören ; Ziegler, Julius ; Pitzer, Benjamin ; Werling, Moritz ; Gindele, Tobias ; Jagszent, Daniel ; Schröder, Joachim ; Thuy, Michael ; Goebl, Matthias ; Hundelshausen, Felix v. ; Pink, Oliver ; Frese, Christian ; Stiller, Christoph: Team AnnieWAY's Autonomous System for the DARPA Urban Challenge 2007. In: *Journal of Field Robotics* 25 (2008), Sep, Nr. 9, S. 615–639 [3.5, 3.7, 5.1.1, 6.1]

LaValle 1998 LaValle, S. M.: Rapidly-exploring random trees: A new tool for path planning / Computer Science Dept., Iowa State University. Oct 1998 (TR 98-11). – Forschungsbericht [2.3.2]

Likhachev u. a. 2005 Likhachev, Maxim ; Ferguson, Dave ; Gordon, Geoff ; Stentz, Anthony ; Thrun, Sebastian: Anytime Dynamic A*: An Anytime, Replanning Algorithm. In: *International Conference on Automated Planning and Scheduling (ICAPS)* (2005) [2.3.2]

Lowrie u. a. 1985 Lowrie, J. M. ; Thomas, M. ; Gremban, K. ; Turk, M.: The autonomous land vehicle (ALV) preliminary raod-following demonstration. In: *Intelligent Robots and Computer Vision (Proc. SPIE 579)* (1985), Sep, S. 336–350 [2.3.1]

Meissner 1982 Meissner, H. G.: *Steuerung dynamischer Systeme aufgrund bildhafter Informationen*, Universität der Bundeswehr München, Dissertation, 1982 [2.3.1]

Michon 1985 Michon, J.A.: A critical view of driver behaviour models: What do we know, what should we do? In: Evans, L. (Hrsg.) ; Schwing, R.C. (Hrsg.): *Human Behaviour and Traffic Safety*. Plenum, 1985 [2.1.7]

Mitschke und Wallentowitz 2004 Mitschke, Manfred ; Wallentowitz, Henning: *Dynamik der Kraftfahrzeuge*. 4. Auflage. Berlin / Heidelberg : Springer, 2004. – ISBN 3-540-42011-8 [3.1.4]

Moravec 1999 Moravec, Hans: *Robot: Mere Machine to Transcendent Mind*. Oxford University Press, 1999. – 240 S. – ISBN 0195136306 [2.3.1]

Nagel und Arens 2003 Nagel, Hans-Hellmut ; Arens, Michael: Zur unscharf-metrisch-temporallogischen Spezifikation von Verkehrssituationen bei der Autobahnfahrt. In: *Beitrag zum Fahrerassistenzworkshop*, 2003 [2.1.5]

Nagel und Arens 2005 Nagel, Hans-Hellmut ; Arens, Michael: *Fahrerassistenzsysteme mit maschineller Wahrnehmung*. Kap. Innervation des Automobils und Formale Logik, S. 89–116, Springer Verlag Berlin, 2005 [2.1.5]

Nilsson 1969 Nilsson, N.J.: A Mobile Automaton: An Application of Artificial Intelligence Techniques / AI Center, SRI International. 333 Ravenswood Ave, Menlo Park, CA 94025, Mar 1969 (40). – Forschungsbericht. SRI Project 7494 IJCAI 1969 [2.3.1]

OpenCV *Intel, Open Source Computer Vision Library*. – URL http://www.intel.com/technology/computing/opencv/index.htm. – Zugriffsdatum: 2008-12-13. [2.4.4]

Pellkofer 2003 Pellkofer, Martin: *Verhaltensentscheidung für autonome Fahrzeuge mit Blickrichtungssteuerung*, Universität der Bundeswehr München, Fakultät für Luft- und Raumfahrttechnik, Dissertation, 2003 [2.1.4, 2.5, 2.3.1]

Pomerleau 1995 POMERLEAU, D.: RALPH: rapidly adapting lateral position handler. In: *Intelligent Vehicles' 95 Symposium., Proceedings of the* (1995), S. 506–511 [2.3.1]

Qt *QT, Trolltech Qt Framework*. – http://trolltech.com/products/qt. – Zugriffsdatum: 2008-12-13. [2.4.1]

Reinholtz u. a. 2007 REINHOLTZ, Charles ; ALBERI, Thomas u. a.: DARPA Urban Challenge Technical Paper: VictorTango / Virginia Tech. 2007. – DARPA Urban Challange 2007 Semifinalist Technical Report [2.3.2]

Russell und Norvig 2004 RUSSELL, Stuart J. ; NORVIG, Peter: *Künstliche Intelligenz*. 2. Auflage. Pearson Studium, 2004. – ISBN 978-3-8273-7089-1, 3-8273-7089-2 [2]

Schmidhuber SCHMIDHUBER: *Robot Cars*. – URL http://www.idsia.ch/~juergen/robotcars.html. – Zugriffsdatum 2008-12-13. [2.3.1]

Schröder u. a. 2008 SCHRÖDER, Joachim ; GINDELE, Tobias ; JAGSZENT, Daniel ; DILLMANN, Rüdiger: Path Planning for Cognitive Vehicles using Risk Maps. In: *Intelligent Vehicles Symposium, 2008 IEEE*, 2008, S. 1119–1124 [6.1]

SFB/TR28 *SFB/TR28 Kognitive Automobile*. – URL http://www.kognimobil.org/. – Zugriffsdatum 2008-12-13. [1]

Siedersberger 2003 SIEDERSBERGER, K.: *Komponenten zur automatischen Fahrzeugführung in sehenden, (semi-)autonomen Fahrzeugen*, Universität der Bundeswehr München, Fakultät für Luft- und Raumfahrttechnik, Dissertation, 2003 [2.1.4, 2.3.1]

Tanenbaum 2002 TANENBAUM, Andrew S.: *Moderne Betriebssysteme*. 2. überarbeitete Auflage. Pearson Studium, 2002. – 1021 S. – ISBN 3827370191 [4.2.3]

Thrun u. a. 2007a THRUN, Sebastian ; MONTEMERLO, Michael u. a.: Stanford's Robotic Vehicle "Junior" Interim Report / Stanford University. 2007. – DARPA Urban Challange 2007 Semifinalist Technical Report [2.3.2]

Thrun u. a. 2007b THRUN, Sebastian ; MONTEMERLO, Michael ; DAHLKAMP, Hendrick u. a.: Stanley: The Robot That Won the DARPA Grand Challenge. In: BUEHLER, Martin (Hrsg.) ; IAGNEMMA, Karl (Hrsg.) ; SINGH, Sanjiv (Hrsg.): *The 2005 DARPA Grand Challenge: The great Robot Race* Bd. 36. Springer, 2007, Kap. 1, S. 1–43. – ISBN 978-3-540-73428-4 [2.3.1]

Tsugawa u. a. 1979 Tsugawa, S. ; Yatabe, T. ; Hirose, T. ; Matsumoto, S.: An Automobile with Artificial Intelligence. In: *Proc. Sixth Int'l Joint Conf. Artificial Intelligence* (1979), S. 893–895 [2.3.1]

Turk u. a. 1988 Turk, M.A. ; Morgenthaler, D.G. ; Gremban, K.D. ; Marra, M.: VITS-A Vision System for Autonomous Land Vehicle Navigation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10 (1988), Nr. 3, S. 342–361. – ISSN 0162-8828 [2.3.1]

Vlacic 2001 Vlacic, Ljubo: *Intelligent vehicle technologies*. Butterworth-Heinemann, 2001. – ISBN 0-7506-5093-1 [2.1.1]

Werling und Groll 2008 Werling, M. ; Groll, L.: Low-level controllers realizing high-level decisions in an autonomous vehicle. In: *Intelligent Vehicles Symposium, 2008 IEEE*, 2008, S. 1113–1118 [3.5]

Whittaker u. a. 2007 Whittaker, William ; Urmson, Chris u. a.: Tartan Racing: A Multi-Modal Approach to the DARPA Urban Chalenge / Carnegie Mellon University. 2007. – DARPA Urban Challange 2007 Semifinalist Technical Report [2.3.2, 2.14]

WHO 2004 World Health Organisation: *Global Burden of Disease Estimates*. 2004. – URL http://www.who.int/healthinfo/global_burden_disease/en/index.html. – Zugriffsdatum: 2008-12-09 [1]

Ziegler u. a. 2008 Ziegler, Julius ; Werling, Moritz ; Schröder, Joachim: Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. In: *Intelligent Vehicles Symposium, 2008 IEEE*, 2008, S. 787–791 [6.1]